Practical Algorithms
- Webpage Ranking → PageRank
- Stable Marriage → Gale-Shapley
- String Matching → Rabin-Karp, Boyer-Moore-Horspool, Aho-Corasick
- Bit Tricks
- Polynomial Multiplication → Cooley-Tukey

Probabilistic Algorithms
- Primality → Miller-Rabin
- Membership → Bloom Filter
- Frequency → Count-Min Sketch
- Cardinality → Hyperloglog

External-Memory Algorithms
- Merging $k$ Sorted Arrays
- Sorting → Merge Sort
Contents

Quantum Algorithms
• GO Fundamentals
• GO ❤ Random Number Generator

Technical Problems
• GO ❤ Majority Vote
• GO ❤ Longest Palindromic Substring
• GO Selection Two Sorted Arrays
• GO Largest Subarray Sum
• GO Loop in a Linked List
• GO Y-shaped Linked List
• GO Search Sorted Matrix
• GO First Missing Positive
• GO Celebrity Problem
• GO Random Permutation
• GO Count Distinct Pairs
• GO Maximum and Minimum
Contents

**Sorting Algorithms**
- Permutation Sort
- Slow Sort
- Pancake Sort
- Stooge Sort
- Counting Sort
- Radix Sort
- Bitonic Sort
Algorithmic-problem-solving template

Step 1. Problem

Step 2. Solutions

Step 3. Complexity

Step 4. Performance

Step 5. Extensions

Step 6. References
Webpage Ranking
Problem

• Design an algorithm to rank web pages efficiently.
• Input: A directed graph with transition probability
  Output: [0.310945, 0.415423, 0.248756, 0.0248756]

Internet = directed graph
Web pages = nodes
Hyperlinks = edges
Transitions are probabilistic
What is the meaning of ranking?

Meaning of page rank.
- Rank = relative importance
- Rank = number of times a user visits a page when they keep visiting pages via hyperlinks for a long time
- Rank = proportion of time a user spends in that page if the user spends long enough time

Stationary/stable distribution (SD).
- If a user visits pages of the Internet for a long period of time as per transition probabilities, the distribution stabilizes and this is called stationary/stable distribution (SD)
Page rank algorithm

- Algorithm discovered by Sergey Brin and Lawrence Page
- Core idea behind Google
- A billion-dollar algorithm
- Ranks billions of pages efficiently
- **Static page ranking** = ranking of all web pages on the Internet
  **Dynamic page ranking** = ranking of web pages on the Internet related to the search terms
- The relative ranks of web pages returned for search queries might be very different from their relative ranks when they are measured statically (without search terms).
- Here, we will only learn about **static ranking of all web pages**
Modeling page rank using linear algebra

$$T[i, j] = \text{Probability of a user transitioning from page } i \text{ to page } j$$

![Diagram of a web graph with probabilities]

Transition matrix $T = \begin{bmatrix} 0 & 0.6 & 0.4 & 0 \\ 0.7 & 0 & 0.3 & 0 \\ 0 & 0.9 & 0 & 0.1 \\ 0.8 & 0.2 & 0 & 0 \end{bmatrix}$

- Row sum (sum of outgoing prob.) should be 1
- Col sum (sum of incoming prob.) does not mean anything
Modeling page rank using linear algebra

\[ P_t[i] = \text{Probability of a user being at page } i \text{ at time } t \]
Modeling page rank using linear algebra

\[ P_t[i] = \text{Probability of a user being at page } i \text{ at time } t \]

\[
P_t[i] = \left\{ \begin{array}{c}
P_{t-1}[1] \times T[1,i] \\
+ P_{t-1}[2] \times T[2,i] \\
\vdots \\
+ P_{t-1}[n] \times T[n,i]
\end{array} \right\} = \sum_{j=1}^{n} (P_{t-1}[j] \times T[j,i])
\]
Modeling page rank using linear algebra

\[ P_t[i] = \text{Probability of a user being at page } i \text{ at time } t \]

\[
P_t[i] = \begin{cases} 
  P_{t-1}[1] \times T[1, i] \\
  + P_{t-1}[2] \times T[2, i] \\
  \vdots \\
  + P_{t-1}[n] \times T[n, i] 
\end{cases} = \sum_{j=1}^{n} (P_{t-1}[j] \times T[j, i])
\]

\[
P_t = [P_t[1] \ P_t[2] \ \cdots \ P_t[n]]
\]

\[
= \begin{bmatrix} 
  \sum_{j=1}^{n} (P_{t-1}[j] \times T[j, 1]) & \cdots & \sum_{j=1}^{n} (P_{t-1}[j] \times T[j, n]) 
\end{bmatrix}
\]

\[
= P_{t-1} \times T
\]
Modeling page rank using linear algebra

\[ P_t[i] = \text{Probability of a user being at page } i \text{ at time } t \]

\[
P_t[i] = \left\{ \begin{array}{l}
P_{t-1}[1] \times T[1,i] \\
+ P_{t-1}[2] \times T[2,i] \\
\vdots \\
+ P_{t-1}[n] \times T[n,i]
\end{array} \right\} = \sum_{j=1}^{n} (P_{t-1}[j] \times T[j,i])
\]

\[
P_t = [P_t[1] \ P_t[2] \ \cdots \ P_t[n]]
\]

\[
= \begin{bmatrix} \sum_{j=1}^{n} (P_{t-1}[j] \times T[j,1]) & \cdots & \sum_{j=1}^{n} (P_{t-1}[j] \times T[j,n]) \end{bmatrix}
\]

\[
= P_{t-1} \times T
\]

\[
P_t = \begin{cases} 
\begin{bmatrix} \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \end{bmatrix} & \text{if } t = 0, \\
P_{t-1} \times T & \text{if } t > 0.
\end{cases}
\]
Page ranks or stable distribution $P$ is computed as:

$$P = P \times T$$

Questions...

- Do we always get a stable distribution (convergence)?
- If there is a stable distribution, is it always unique?
- Does every initial distribution converge to a stable distribution?
There are three major algorithms to compute page ranks or SD $P$, if it exists:

- Brute force
- System of linear equations
- Eigenvector
Solutions $\rightarrow$ Brute force

![Graph with nodes 1, 2, 3, 4 and transition probabilities]

$T = \begin{bmatrix} 0 & 0.6 & 0.4 & 0 \\ 0.7 & 0 & 0.3 & 0 \\ 0 & 0.9 & 0 & 0.1 \\ 0.8 & 0.2 & 0 & 0 \end{bmatrix}$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>1</td>
<td>0.29166667</td>
<td>0.30833333</td>
<td>0.225</td>
<td>0.175</td>
</tr>
<tr>
<td>2</td>
<td>0.31305556</td>
<td>0.34305556</td>
<td>0.21972222</td>
<td>0.12416667</td>
</tr>
<tr>
<td>3</td>
<td>0.32186111</td>
<td>0.36550926</td>
<td>0.22252778</td>
<td>0.09010185</td>
</tr>
<tr>
<td>4</td>
<td>0.32388673</td>
<td>0.38081019</td>
<td>0.22781759</td>
<td>0.06748549</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>19</td>
<td>0.310945</td>
<td>0.415423</td>
<td>0.248756</td>
<td>0.0248756</td>
</tr>
</tbody>
</table>

Page ranks $P = [0.310945, 0.415423, 0.248756, 0.0248756]$
Solutions $\rightarrow$ System of linear equations

\[
T = \begin{bmatrix}
0 & 0.6 & 0.4 & 0 \\
0.7 & 0 & 0.3 & 0 \\
0 & 0.9 & 0 & 0.1 \\
0.8 & 0.2 & 0 & 0 \\
\end{bmatrix}
\]

\[
P_t[1] = 0.7 \cdot P_t[2] + 0.8 \cdot P_t[4] \\
P_t[2] = 0.6 \cdot P_t[1] + 0.9 \cdot P_t[3] + 0.2 \cdot P_t[4] \\
P_t[3] = 0.4 \cdot P_t[1] + 0.3 \cdot P_t[2] \\
P_t[4] = 0.1 \cdot P_t[3] \\
\]

Page ranks $P = [0.310945, 0.415423, 0.248756, 0.0248756]$
\[ T = \begin{bmatrix} 0 & 0.6 & 0.4 & 0 \\ 0.7 & 0 & 0.3 & 0 \\ 0 & 0.9 & 0 & 0.1 \\ 0.8 & 0.2 & 0 & 0 \end{bmatrix} \]

\[
\begin{align*}
&\begin{bmatrix} 0 & 0.7 & 0 & 0.8 \\ 0.6 & 0 & 0.9 & 0.2 \\ 0.4 & 0.3 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \end{bmatrix} \times \\
\end{align*}
\]
\[ P = P \cdot T \quad \Rightarrow \quad \text{col}(P) = T^{\text{transpose}} \cdot \text{col}(P) \]
which is equivalent to the formula
\[ \lambda \vec{v} = A \cdot \vec{v} \]
where \( \lambda = 1 \), \( A = T^{\text{transpose}} \), and \( \vec{v} = \text{col}(P) \)
and \( \vec{v} \) is the eigenvector of \( A \) corresponding to eigenvalue 1. So,
Page ranks \( P = [0.310945, 0.415423, 0.248756, 0.0248756] \)
Problems

There can be two types of problems.

- Page ranks can be zero
- Page ranks can be unstable
Problem → Page rank being zero

A sink subgraph is the subgraph of the given digraph that has no outgoing edges from it to the rest of the graph. Example: The set \{1, 2, 3\} is a sink subgraph.

A strongly connected graph is a digraph such that there is a directed path between every two nodes. Example: There is no path from any of \{1, 2, 3\} to 4.

If the pagerank of at least one of the nodes in the digraph is zero, then the graph is not strongly connected.

How can the rank of a page be equal to zero?

\[ P = [0.376147, 0.422018, 0.201835, 0] \]

\[ T = \begin{bmatrix} 0 & 0.8 & 0.2 & 0 \\ 0.7 & 0 & 0.3 & 0 \\ 0.4 & 0.6 & 0 & 0 \\ 0.7 & 0.2 & 0.1 & 0 \end{bmatrix} \]
Problem → Page rank being zero

How can the rank of a page be equal to zero?

A sink subgraph is the subgraph of the given digraph that has no outgoing edges from it to the rest of the graph.

Example: The set \{1, 2, 3\} is a sink subgraph.

A strongly connected graph is a digraph such that there is a directed path between every two nodes.

Example: There is no path from any of \{1, 2, 3\} to 4.

If the pagerank of at least one of the nodes in the digraph is zero, then the graph is not strongly connected.
Problem → Unstable page ranks

How can the ranks of pages be unstable?

A periodic Markov chain is a Markov chain such that the distribution oscillates between multiple distributions periodically. An aperiodic Markov chain is a Markov chain that is not periodic. Pageranks do not converge for a periodic Markov chain.
How can the ranks of pages be unstable?

A periodic Markov chain is a Markov chain such that the distribution oscillates between multiple distributions periodically.

An aperiodic Markov chain is a Markov chain that is not periodic.

Pageranks do not converge for a periodic Markov chain.
For a Markov chain that is strongly connected (i.e., no sink subgraph) and aperiodic:
1. A unique stable distribution $P$ exists
2. All initial distributions $P_0$ converge to $P$
• What should we do if a Markov chain is not strongly connected or periodic?
   Idea: Transform it to Markov chain that is strongly connected and periodic

• How do you do this conversion/transformation?
   Idea: Transform the given digraph into a complete graph by making each edge cost positive using uniform randomization (via a concept called teleportation factor)
   Intuition: When a user is stuck at a page (or a subgraph) there is a tiny nonzero probability that she will be teleported to a random web page
• How do you transform the digraph into a complete graph?
Transform the given transition matrix $T$ using a chosen teleportation factor $\alpha \in (0, 1)$ as:

$$T[i, j] \leftarrow (1 - \alpha)T[i, j] + \alpha \left(\frac{1}{n}\right)$$
or

$$T \leftarrow (1 - \alpha)T + \frac{\alpha}{n}S$$

where $S$ is an $n \times n$ matrix with all 1s
How do you transform the digraph into a complete graph?

Transform the given transition matrix $T$ using a chosen teleportation factor $\alpha \in (0, 1)$ as:

$$T[i, j] \leftarrow (1 - \alpha)T[i, j] + \alpha \left( \frac{1}{n} \right)$$  

or

$$T \leftarrow (1 - \alpha)T + \frac{\alpha}{n} S$$

where $S$ is an $n \times n$ matrix with all 1s

What does this transformation mean?

This transformation means that

$$T[i, j] \begin{cases} \text{incremented} & \text{if } T[i, j] \in [0, 1/n) \\ \text{the same} & \text{if } T[i, j] = 1/n \\ \text{decremented} & \text{if } T[i, j] \in (1/n, 1] \end{cases}$$

After the transformation (for $n \geq 2$), $T[i, j] \in (0, 1)$ for all $i, j$. 
Algorithms

- Brute force
- Power method
- System of equations
- Eigenvector
Solutions → Brute force

BruteForce($T[1 \ldots n, 1 \ldots n]$)

TransformTransitionMatrix($T$)
Create the pagerank arrays $P_{old}[1 \ldots n]$ and $P_{new}[1 \ldots n]$

$P_{old}[1 \ldots n] \leftarrow [1/n, 1/n, \ldots, 1/n]$  
$P_{new}[1 \ldots n] \leftarrow P_{old}[1 \ldots n]$

$maxerror \leftarrow 10^{-6}; error \leftarrow \infty$

while $error > maxerror$ do
  $P_{new} \leftarrow P_{old} \cdot T$
  $error \leftarrow |P_{new} - P_{old}|$
  $P_{old} \leftarrow P_{new}$

return $P_{new}[1 \ldots n]$
Solutions → System of equations

<table>
<thead>
<tr>
<th>SystemOfEquations($T[1 \ldots n, 1 \ldots n]$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransformTransitionMatrix($T$)</td>
</tr>
<tr>
<td>Create the pagerank array $P[1 \ldots n]$</td>
</tr>
<tr>
<td>$P \leftarrow$ SolveSystemOfEquations($P = P \cdot T$)</td>
</tr>
<tr>
<td>return $P[1 \ldots n]$</td>
</tr>
</tbody>
</table>
### Eigenvector Method

**EigenvectorMethod**($T[1 \ldots n, 1 \ldots n]$)

1. **TransformTransitionMatrix**($T$)
2. Create the pagerank array $P[1 \ldots n]$ where $P \leftarrow \text{SolveForFirstEigenvector}(\text{col}(P) = T^{\text{transpose}} \cdot \text{col}(P))$
3. **return** $P[1 \ldots n]$
Which is the best algorithm?

- Brute force takes time till convergence
  System of linear equations takes $O(n^3)$
  Eigenvector method takes $O(n^3)$
- Which algorithm is the fastest?
  Brute force takes the least time for Internet-type graphs
  ($\approx 50$ iterations)

Theoretically fast algos might not always be the best in practice
References

- Page rank video by Reducible
- AMS description of page rank
- Cornell tutorial for page rank
- Brin and Page’s paper
- Linear algebra behind Google
Given the preference/priority list of $n$ guys and $n$ gals, design an algorithm to determine if a set of stable marriages exists and find one such set.
A matching is a 1-to-1 correspondence b/w $n$ guys and $n$ gals.

An unstable pair is a pair $(m, w)$ who would have a love affair:
- Man $m$ prefers woman $w$ to his matched partner and
- Woman $w$ prefers man $w$ to her matched partner

An unstable matching is a set of marriages which has an unstable pair $(C, p)$.

<table>
<thead>
<tr>
<th>Guy</th>
<th>Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st</td>
</tr>
<tr>
<td>A</td>
<td>p</td>
</tr>
<tr>
<td>B</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>r</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gal</th>
<th>Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st</td>
</tr>
<tr>
<td>p</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>A</td>
</tr>
</tbody>
</table>

This matching is unstable due to the unstable pair $(C, p)$:
- C prefers p over C’s matched partner q
- p prefers C over p’s matched partner A
### Example 1

<table>
<thead>
<tr>
<th>Guy</th>
<th>Preference</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>q</td>
<td>r</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>r</td>
<td>p</td>
<td>q</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gal</th>
<th>Preference</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

**Answer**

There are three sets of stable marriages.
- Give each man his first choice: \{ (A,p), (B,q), (C,r) \}  
  (Each woman gets her last choice)
- Give each woman her first choice: \{ (A,r), (B,p), (C,q) \}  
  (Each man gets his last choice)
- Give each man his second choice: \{ (A,q), (B,r), (C,p) \}  
  (Each woman gets her second choice)

All other sets are unstable.
Example 2

<table>
<thead>
<tr>
<th>Guy</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
<td>s</td>
<td>r</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
<td>p</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
<td>q</td>
<td>r</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gal</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

Answer

There is only one set of stable marriages.

- \{ (A,r), (B,s), (C,p), (D,q) \}
  - All men get their second choice, except A who gets third choice
  - All women get their second choice
  - All other sets are unstable.
In 1962, David Gale and Lloyd Shapley discovered an algorithm based on deferred acceptance that guarantees that

- (Matching) each man and each woman gets matched
- (Stability) the matching is stable
- (Optimality) the matching is always best for the group that proposes and worst for the group that handles proposals

Applications:
Matching hospitals and medical residents
Matching roommates

Shapley and Roth were awarded 2012 Nobel Memorial Prize in Economic Sciences (Gale died in 2008)

Exists as functions in Python, MATLAB, and R
Gale-Shapley’s algorithm

High level description of the algorithm
1. All individuals rank the members of the opposite set in order of preference
2. One of the two sets is chosen to make proposals
3. In a loop, run
   i. An individual from the proposing group who is not already engaged will propose to their most preferable option who has not already rejected them
   ii. The person being proposed to will:
      • Accept if this is their first offer
      • Reject if this is worse than their current offer
      • Accept if this is better than their current offer
4. When all members of the proposing group are matched, terminate. The current pairs represents stable set of marriages.
Gale-Shapley’s algorithm

Let’s understand the working of the algorithm on an example.

<table>
<thead>
<tr>
<th>Guy</th>
<th>Preference</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
<td>s</td>
<td>r</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
<td>p</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
<td>q</td>
<td>r</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gal</th>
<th>Preference</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

A proposes to p and p accepts the proposal
A is p’s first offer/proposal
### Gale-Shapley’s algorithm

<table>
<thead>
<tr>
<th>Guy</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
<td>s</td>
<td>r</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
<td>p</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
<td>q</td>
<td>r</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gal</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

B proposes to p and p rejects the proposal. p’s current partner A is better than B.
### Gale-Shapley’s algorithm

**Guy**

<table>
<thead>
<tr>
<th>Guy</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
<td>s</td>
<td>r</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
<td>p</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
<td>q</td>
<td>r</td>
<td>p</td>
</tr>
</tbody>
</table>

**Gal**

<table>
<thead>
<tr>
<th>Gal</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

B proposes to p and p rejects the proposal  

p’s current partner A is better than B
**Gale-Shapley’s algorithm**

<table>
<thead>
<tr>
<th>Guy</th>
<th>Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st</td>
</tr>
<tr>
<td>A</td>
<td>p</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gal</th>
<th>Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st</td>
</tr>
<tr>
<td>p</td>
<td>D</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
</tr>
</tbody>
</table>

C proposes to q and q accepts the proposal
C is q’s first offer/proposal
Gale-Shapley’s algorithm

<table>
<thead>
<tr>
<th>Guy</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
<td>s</td>
<td>r</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
<td>p</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
<td>q</td>
<td>r</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gal</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

D proposes to s and s accepts the proposal
D is s’s first offer/proposal
Gale-Shapley’s algorithm

<table>
<thead>
<tr>
<th>Guy</th>
<th>Preference</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st</td>
<td>2nd</td>
<td>3rd</td>
<td>4th</td>
</tr>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
<td>s</td>
<td>r</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
<td>p</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
<td>q</td>
<td>r</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gal</th>
<th>Preference</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st</td>
<td>2nd</td>
<td>3rd</td>
<td>4th</td>
</tr>
<tr>
<td>p</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

B proposes to s and s accepts the proposal
B is better than s’s current partner D
Gale-Shapley’s algorithm

<table>
<thead>
<tr>
<th>Guy</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
<td>s</td>
<td>r</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
<td>p</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
<td>q</td>
<td>r</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gal</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

D proposes to q and q accepts the proposal
D is better than q’s current partner C
### Gale-Shapley’s algorithm

<table>
<thead>
<tr>
<th>Guy</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
<td>s</td>
<td>r</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
<td>p</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
<td>q</td>
<td>r</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gal</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

C proposes to p and p accepts the proposal. C is better than p’s current partner A.
## Gale-Shapley’s algorithm

### Preference

**Guy**

<table>
<thead>
<tr>
<th>Guy</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
<td>s</td>
<td>r</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
<td>p</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
<td>q</td>
<td>r</td>
<td>p</td>
</tr>
</tbody>
</table>

**Gal**

<table>
<thead>
<tr>
<th>Gal</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

A proposes to q and q rejects the proposal. q’s current partner D is better than A.
### Gale-Shapley’s algorithm

#### Preference

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
<td>s</td>
<td>r</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
<td>p</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
<td>q</td>
<td>r</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

A proposes to r and r accepts the proposal.
A is r’s first offer/proposal.
Gale-Shapley’s algorithm

<table>
<thead>
<tr>
<th>Guy</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>B</td>
<td>p</td>
<td>s</td>
<td>r</td>
<td>q</td>
</tr>
<tr>
<td>C</td>
<td>q</td>
<td>p</td>
<td>r</td>
<td>s</td>
</tr>
<tr>
<td>D</td>
<td>s</td>
<td>q</td>
<td>r</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gal</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>q</td>
<td>B</td>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>r</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>s</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

There is no man who is not engaged.
Algorithm terminates.
Stable matching is achieved.
Matching best for men and worst for women.
If women propose and men handle proposals you get the same stable matching in this specific example.

This is because this instance has only one stable matching.

In other examples, you can get different stable matchings.
Gale-Shapley’s algorithm

```plaintext
STABLEMARRIAGE-GALESHAPLEY(menpreferences, womenpreferences)

Input: \( n \) = number of men (or women), menpreferences = preference list of men, womenpreferences = preference list of women

Output: Stable matching

engaged = dictionary with initial empty mappings for men and women

while there is a man who is not engaged do
    man = next non-engaged man and
    woman = first woman in man’s preferences list to whom man has not yet proposed

    if woman is not engaged then
        engage man and woman
    else if woman prefers man to her current partner then
        mark current partner as not engaged
        engage man and woman
    else if woman does not prefer man to her current partner then
        woman rejects man

return stable matching engaged mapping
```
Complexity

- Time $= \mathcal{O}(n^2)$
- Space $= \mathcal{O}(n)$ for notengaged dequeue and engaged map
• Stable matching even when \( \#\text{men} > \#\text{women} \) or \( \#\text{women} > \#\text{men} \)

• There is no stable matching for stable roommate matching

An even number of boys wish to divide up into pairs of roommates

Example: Boys A, B, C, D where A ranks B first, B ranks C first, C ranks A first, and A,B,C all rank D last. Then regardless of D’s preferences there can be no stable pairing, for whoever has to room with D will want to move out and one of the other two will be willing to take him in.
References

- Gale-Shapley paper
- Gale-Shapley simulation
Given a text \text{text}[1 \ldots n] and a pattern \text{pattern}[1 \ldots m], design an algorithm to find the location of the first occurrence of the pattern in the text.
### Solutions → Brute force

**text**[1…n]  
```
 a a a a a a a a a b
```

**pattern**[1…m]  
```
 a a a b
```

![Diagram showing the brute force method of pattern matching](image_url)
1. Check if the pattern matches the text starting from the 1st index of text.
2. If not, check if the pattern matches with the text starting from the 2nd index of the text.
3. Repeat this process until either the pattern is found or the end of the text is reached (without finding any pattern).

\[
\text{StringMatching-BruteForce}(text[1 \ldots n], \text{pattern}[1 \ldots m])
\]

\[
\text{for } i \leftarrow 1 \text{ to } n - m + 1 \text{ do}
\]
\[
// \text{ If text window at position } i \text{ matches with pattern, return position}
\]
\[
\text{if } text[i \ldots (i + m - 1)] = \text{pattern} \text{ then}
\]
\[
\text{return } i
\]
\[
\text{return } -1
\]

\[
\langle \text{PreprocessTime, MatchTime, Space} \rangle = \langle 0, \mathcal{O}(mn), \Theta(1) \rangle
\]
Solutions → Hashing

\[
\text{Hash}(\text{string}[1 \ldots m], b, p)
\]

// Polynomial hash: \( s_1 b^{m-1} + s_2 b^{m-2} + \cdots + s_{m-1} b^1 + s_m b^0 \)
// Use Horner's rule to compute polynomial hash
\[
\text{hash} \leftarrow 0
\]
\[
\text{for } i \leftarrow 1 \text{ to } m \text{ do}
\]
\[
| \quad \text{hash} \leftarrow (\text{hash} \times b + \text{string}[i]) \mod p
\]
\[
\text{return } \text{hash}
\]

Time = \( \Theta(m) \)
1. Check if patternhash matches the texthash at index 1.
2. If not, check if patternhash matches the texthash at index 2.
3. Repeat this process until either the pattern is found or the end of the text is reached (without finding any pattern).

```plaintext
STRING_MATCHING-HASHING(text[1...n], pattern[1...m])

p ← a good prime // e.g.: 101
b ← size of ASCII set // i.e., 256
patternhash ← HASH(pattern, b, p)
texthash ← HASH(text[1...m], b, p)

for i ← 1 to n – m + 1 do
    // If hash value of text window matches the hash value of pattern and if the text window matches the pattern then there is a match
    if texthash = patternhash and text[i...(i+m-1)] = pattern then
        return i
    // Compute hash value of the next text window in Θ(m) time
    if i ≠ n – m + 1 then
        texthash ← HASH(text[i+1...i+m])

return −1
```

⟨PreprocessTime, MatchTime, Space⟩ = ⟨Θ(m), O(mn), Θ(1)⟩
Solutions → RabinKarp (rolling hash)

\[
\text{StringMatching-RabinKarp}(text[1 \ldots n], pattern[1 \ldots m])
\]

\[
p \leftarrow \text{a good prime} \quad \text{// e.g.: 101}
\]
\[
b \leftarrow \text{size of ASCII set} \quad \text{// i.e., 256}
\]
\[
h \leftarrow b^{m-1} \mod p \quad \text{// highest term in the polynomial hash}
\]
\[
patternhash \leftarrow \text{Hash}(pattern, b, p)
\]
\[
texthash \leftarrow \text{Hash}(text[1 \ldots m], b, p)
\]

\[
\text{for } i \leftarrow 1 \text{ to } n - m + 1 \text{ do}
\]
\[
\quad \text{if } texthash = patternhash \text{ and } text[i \ldots (i + m - 1)] = pattern \text{ then}
\quad \quad \text{return } i
\]
\[
\quad \text{// Rolling hash: Compute hash value of the next text window using}
\quad \text{the current text window in } \Theta(1) \text{ time}
\quad \text{if } i \neq n - m + 1 \text{ then}
\quad \quad \text{texthash } \leftarrow \text{RollingHash}(texthash, text[i \ldots i + m])
\]

\[
\text{return } -1
\]
RabinKarp (rolling hash)

\[
\text{RollingHash}(\text{texthash, string}[1 \ldots m']) \quad (m' = m + 1)
\]

\[
\text{texthash} \leftarrow ((\text{texthash} - \text{string}[1] \times h) \times b + \text{string}[m']) \mod p
\]

\text{return texthash}

Time = \Theta (1)
Solutions → RabinKarp (rolling hash)

\[ \text{pattern}[1 \ldots m] \]

\[
\begin{array}{cccc}
6 & 3 & 2 & 4 \\
3 & 2 & 4 & 1 \\
2 & 4 & 1 & 2 \\
4 & 1 & 2 & 3 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{align*}
\text{1} \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0 & = 1234 \\
6 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0 & = 6324 \\
(6324 - 6 \cdot 10^3) \cdot 10 + 1 & = 3241 \\
(3241 - 3 \cdot 10^3) \cdot 10 + 2 & = 2412 \\
(2412 - 2 \cdot 10^3) \cdot 10 + 3 & = 4123 \\
(4123 - 4 \cdot 10^3) \cdot 10 + 4 & = 1234
\end{align*}
\]
\[ \text{pattern}[1 \ldots m] \]

\[ \begin{array}{cccccccc}
6 & 3 & 2 & 4 & 1 & 2 & 3 & 4 & 2 & 6 & 8 & 9 & 7
\end{array} \]

\[ \text{text}[1 \ldots n] \]

\[ \begin{array}{cccc}
6 & 3 & 2 & 4 \\
3 & 2 & 4 & 1 \\
2 & 4 & 1 & 2 \\
4 & 1 & 2 & 3 \\
1 & 2 & 3 & 4
\end{array} \]

\[(1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0) \mod 31 = 2\]

\[(6 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0) \mod 31 = 8\]

\[((8 - 6 \cdot 10^3 \mod 31) \cdot 10 + 1) \mod 31 = 25\]

\[((25 - 3 \cdot 10^3 \mod 31) \cdot 10 + 2) \mod 31 = 2\]

\[((2 - 2 \cdot 10^3 \mod 31) \cdot 10 + 3) \mod 31 = 8\]

\[((8 - 4 \cdot 10^3 \mod 31) \cdot 10 + 4) \mod 31 = 2\]
Solutions  →  Boyer-Moore-Horspool

\[ skip[\alpha] = \begin{cases} 
\text{distance from the end of the pattern of } \alpha\text{'s last occurrence} & \text{if } \alpha \neq \text{pattern}[m] \\
\text{distance from the end of the pattern of } \alpha\text{'s last but one occurrence} & \text{if } \alpha = \text{pattern}[m] 
\end{cases} \]
### Solutions → Boyer-Moore-Horspool

#### StringMatching-BMH\( (text[1 \ldots n], pattern[1 \ldots m]) \)

\[
\text{skip}[0 \ldots 255] \leftarrow \text{ConstructSkipTable}(pattern)
\]

\[
i \leftarrow m
\]

\[
\text{while } i \leq n \text{ do}
\]

\[
\quad \text{if } text[(i - m + 1) \ldots i] = pattern \text{ comparing from right to left then}
\]

\[
\quad \quad \text{return } i - m + 1
\]

\[
\quad \text{else}
\]

\[
\quad \quad i \leftarrow i + \text{skip}[text[i]]
\]

\[
\text{return } -1
\]

#### ConstructSkipTable\( (pattern[1 \ldots m]) \)

\[
// \text{Initialize the skip table of ASCII characters to } m
\]

\[
\text{skip}[0 \ldots 255] \leftarrow [m \ldots m]
\]

\[
\text{for } i \leftarrow 1 \text{ to } m - 1 \text{ do}
\]

\[
\quad \text{skip}[pattern[i]] \leftarrow m - i
\]

\[
\text{return } \text{skip}[0 \ldots 255]
\]

\[
\langle \text{PreprocessTime, MatchTime, Space} \rangle = \langle \Theta (m + |\Sigma|), O(mn), \Theta (|\Sigma|) \rangle
\]
Aho-Corasick

\(text[1 \ldots n]\)

\(pattern[1 \ldots m]\)

\[
\begin{align*}
&\text{State} & a & b & c & \sum - \{a, b, c\} \\
&0 & 1 & 0 & 0 & 0 \\
&1 & 1 & 2 & 0 & 0 \\
&2 & 3 & 0 & 0 & 0 \\
&3 & 1 & 4 & 0 & 0 \\
&4 & 5 & 0 & 0 & 0 \\
&5 & 1 & 4 & 6 & 0 \\
&6 & - & - & - & - \\
\end{align*}
\]
### StringMatching-AhoCorasick\( (text[1 \ldots n], pattern[1 \ldots m]) \)

\[ \text{transitionTable}[0 \ldots m, 0 \ldots 255] \leftarrow \text{BuildTransitionTable}(pattern) \]

\[
\begin{align*}
\text{state} & \leftarrow 0 \\
\text{for } i & \leftarrow 1 \text{ to } n \text{ do} \\
& \quad \text{state} \leftarrow \text{transitionTable}[\text{state}, text[i]] \\
& \quad \text{if } \text{state} = m \text{ then} \\
& \quad \quad \text{return } i - m + 1 \\
\text{return } -1
\end{align*}
\]

\[ \langle \text{PreprocessTime}, \text{MatchTime}, \text{Space} \rangle = \langle \Theta (m), O (n), \Theta (m) \rangle \]
## Solutions → Aho-Corasick

**BuildTransitionTable**(pattern[1...m])

// Stage 1. Construct array X such that X[i] represents the length of the longest proper suffix at index i which is also the prefix at index 1

<table>
<thead>
<tr>
<th>X[0...m] ← [0...0]; len ← 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>for i ← 1 to m do</td>
</tr>
<tr>
<td>if i &lt; m and pattern[i + 1] = pattern[len + 1] then</td>
</tr>
<tr>
<td>len ← len + 1; X[i] ← len</td>
</tr>
<tr>
<td>else if len ≠ 0 then</td>
</tr>
<tr>
<td>len ← X[len − 1]; i ← i − 1</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>X[i] ← 0</td>
</tr>
</tbody>
</table>

// Stage 2. Compute table from X array

| table[0...m, 1...|Σ|] ← [0...0, 0...0]; table[0, pattern[1]] ← 1 |
|-----------------------------|
| for i ← 1 to m do |
| for j ← 1 to |Σ| do |
| if i < m and j = pattern[i + 1] then table[i, j] ← i + 1 |
| else table[i, j] ← table[X[i − 1], j] |

return table

\[
\langle \text{Time, Space} \rangle = \langle \Theta(m\Sigma), \Theta(m\Sigma) \rangle
\]
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Preprocess time</th>
<th>Matching time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>$-$</td>
<td>$\mathcal{O}(mn)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Rabin Karp</td>
<td>$\Theta(m)$</td>
<td>$\mathcal{O}(mn)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Horspool</td>
<td>$\Theta(m +</td>
<td>\Sigma</td>
<td>)$</td>
</tr>
<tr>
<td>Aho-Corasick</td>
<td>$\Omega(m</td>
<td>\Sigma</td>
<td>)$</td>
</tr>
</tbody>
</table>
Why care?

Problem

- Why care for bit tricks?

- Extensively used by compilers and programmers for achieving high performance
- Easily extends to Bit vector; Instead of working with 32 or 64 bits, bit vector can use an arbitrary size of bits and the operations and concepts remain the same
- Many algorithms make use these bit tricks
  Example: most-widely used HyperLogLog++ algorithm requires counting the number of trailing zeros in a word
Binary representation

- Let $x = \langle x_{w-1}x_{w-2} \ldots x_0 \rangle$ be a $w$-bit word
- Unsigned integer value stored in $x$ is
  $$x = \sum_{i=0}^{w-1} x_i 2^i$$
- Signed integer value stored in $x$ is
  $$x = \sum_{i=0}^{w-1} x_i 2^i$$
- Prefix $0B$ represents a binary number in programming languages
- Examples:
  Unsigned int $x = 0B10010110 = 128 + 16 + 4 + 2 = 150$
  Signed int $x = 0B10010110 = -128 + 16 + 4 + 2 = -106$
Bitwise operators

\[ A = 0B10110011 \]
\[ B = 0B01101001 \]

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>AND</td>
<td>[ A &amp; B = 0B00100001 ]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ A</td>
</tr>
<tr>
<td>⊕</td>
<td>XOR</td>
<td>[ A \oplus B = 0B11011010 ]</td>
</tr>
<tr>
<td>∼</td>
<td>NOT</td>
<td>[ \sim A = 0B01001100 ]</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>shift right</td>
<td>[ A &gt;&gt; 1 = 0B01011001 ]</td>
</tr>
<tr>
<td></td>
<td>shift right</td>
<td>[ A &gt;&gt; 2 = 0B00101100 ]</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>shift left</td>
<td>[ A &gt;&gt; 1 = 0B01100110 ]</td>
</tr>
<tr>
<td></td>
<td>shift left</td>
<td>[ A &gt;&gt; 2 = 0B11001100 ]</td>
</tr>
</tbody>
</table>
Complementation

Problem

- Take the complement of a word $x$

1’s complement $\sim x$

2’s complement $\sim x + 1 = -x$

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>$\sim x$</td>
<td>0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 0</td>
</tr>
<tr>
<td>$\sim x + 1$</td>
<td>0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1</td>
</tr>
</tbody>
</table>
Odd or even

Problem
- Check if an integer is odd or even

\[ A = x \& 1 \]

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>( A = x &amp; 1 )</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
</tbody>
</table>
**Problem**

- Extract the $k$th bit in a word $x$.

\[
\text{mask} = 1 \ll k \\
A = (x \& \text{mask}) \gg k
\]

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>1 0 1 1 1 1 0 1 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>$\text{mask} = 1 \ll 7$</td>
<td>0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$x &amp; \text{mask}$</td>
<td>0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$A = (x &amp; \text{mask}) \gg 7$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
</tbody>
</table>
Set a bit

**Problem**

- Set the $k$th bit in a word $x$.

\[
\begin{align*}
\text{mask} &= 1 \ll k \\
A &= x \mid \text{mask}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>$\text{mask} = 1 \ll 7$</td>
<td>0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$A = x \mid \text{mask}$</td>
<td>1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1</td>
</tr>
</tbody>
</table>
Clear a bit

### Problem

- Clear the $k$th bit in a word $x$.

\[
\text{mask} = \sim (1 << k) \\
A = x \& \text{mask}
\]

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>101111101111010111011101</td>
</tr>
<tr>
<td>$1 &lt;&lt; 7$</td>
<td>000000000010000000</td>
</tr>
<tr>
<td>$\text{mask} = \sim (1 &lt;&lt; 7)$</td>
<td>11111111110111111101111111</td>
</tr>
<tr>
<td>$A = x &amp; \text{mask}$</td>
<td>1011111010101101101101</td>
</tr>
</tbody>
</table>
Problem

- Toggle the $k$th bit in a word $x$.

$$mask = 1 \ll k$$

$$A = x \oplus mask$$

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>$mask = 1 \ll 7$</td>
<td>0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$A = x \oplus mask$</td>
<td>1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 1</td>
</tr>
</tbody>
</table>
## Problem

- Extract a bit field in a word \( x \).

\[
A = (x \& mask) >> shift
\]

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>( mask )</td>
<td>0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>( x &amp; mask )</td>
<td>0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>((x &amp; mask) &gt;&gt; shift)</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0</td>
</tr>
</tbody>
</table>
Set a bit field

Problem

- Set a bit field in a word \(x\) to a value \(y\).

\[
A = (x \& \sim \text{mask}) \mid ((y << \text{shift}) \& \text{mask})
\]

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x)</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>(\text{mask})</td>
<td>0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>(\sim \text{mask})</td>
<td>1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>(A_1 = x &amp; \sim \text{mask})</td>
<td>1 0 1 1 1 0 0 0 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>(y)</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>(A_2 = (y &lt;&lt; 7) &amp; \text{mask})</td>
<td>0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>(A = A_1 \mid A_2)</td>
<td>1 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1</td>
</tr>
</tbody>
</table>
Swap

Problem

• Swap two integers \( x \) and \( y \).

Using temporary variables: \( t = x; \ x = y; \ y = t; \)

No temporary variables: \( x = x \oplus y; \ y = x \oplus y; \ x = x \oplus y; \)

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>( y )</td>
<td>0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>( x = x \oplus y )</td>
<td>1 0 1 1 1 1 0 1 1 0 1 0 0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>( y = x \oplus y )</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>( x = x \oplus y )</td>
<td>0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>
Problem

• How does \{ x = x \oplus y; y = x \oplus y; x = x \oplus y; \} swap?

Core idea:

\[ a \oplus a = 0 \implies b \oplus a \oplus a = a \oplus b \oplus a = a \oplus a \oplus b = b \]

• How do you apply this idea to this algorithm?

Let’s keep the \( x \) and \( y \) variables unchanged

\[
\begin{align*}
  a &= x \oplus y \\
  b &= a \oplus y = x \oplus y \oplus y = x \\
  c &= a \oplus b = x \oplus y \oplus x = y
\end{align*}
\]

• Variables \( b \) and \( c \) store original values of \( x \) and \( y \), respectively
• Variables \( b \) and \( c \) are the variables \( y \) and \( x \), respectively
Detect if two integers have opposite sign

Problem

- Detect if two integers \( x \) and \( y \) have opposite sign.

\[
A = (x \oplus y) < 0
\]

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>( y )</td>
<td>0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>( x \oplus y )</td>
<td>1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0 1</td>
</tr>
<tr>
<td>( A = (x \oplus y) &lt; 0 )</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>( y )</td>
<td>1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>( x \oplus y )</td>
<td>1 0 1 0 0 0 0 1 0 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>( A = (x \oplus y) &lt; 0 )</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>
## Sign of an integer

### Problem

- Determine the sign of an integer \( x \) (return +1/0/-1).

\[
A = (x > 0) - (x < 0)
\]

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>( x &gt; 0 )</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>( x &lt; 0 )</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>( A = (x &gt; 0) - (x &lt; 0) )</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>0 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>( A = (x &gt; 0) - (x &lt; 0) )</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>( A = (x &gt; 0) - (x &lt; 0) )</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>
Check if number is a power of 2

Problem

- Check if unsigned integer $x$ is a power of 2.

$$A = x \land \neg(x \land (x - 1))$$

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 0</td>
</tr>
<tr>
<td>$(x - 1)$</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 0 1 1</td>
</tr>
<tr>
<td>$(x \land (x - 1))$</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 0 0 0</td>
</tr>
<tr>
<td>$\neg(x \land (x - 1))$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$A = x \land \neg(x \land (x - 1))$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0</td>
</tr>
<tr>
<td>$(x - 1)$</td>
<td>0 0 0 0 0 0 0 0 0 0 1 1 1 1 1</td>
</tr>
<tr>
<td>$(x \land (x - 1))$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\neg(x \land (x - 1))$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>$A = x \land \sim (x \land (x - 1))$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
</tbody>
</table>

Create the table for $x = 0$. 
Minimum of two integers

Problem

- Find the minimum of two integers \( x \) and \( y \).

\[
A = y \oplus ((x \oplus y) \& - (x < y))
\]

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 0 1 1 0 1</td>
</tr>
<tr>
<td>( y )</td>
<td>0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>( (x &lt; y) )</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>(- (x &lt; y) )</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>( (x \oplus y) )</td>
<td>1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0</td>
</tr>
<tr>
<td>( (x \oplus y) &amp; -(x &lt; y) )</td>
<td>1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0</td>
</tr>
<tr>
<td>( A = y \oplus ((x \oplus y) &amp; -(x &lt; y)) )</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
</tr>
</tbody>
</table>

- \( x < y \implies -(x < y) = -1 = 1 \ldots 1 \implies A = y \oplus (x \oplus y) = x \)
- \( x \geq y \implies -(x < y) = 0 = 0 \ldots 0 \implies A = y \oplus 0 = y \)
Maximum of two integers

Problem

• Find the maximum of two integers $x$ and $y$.

$$A = x \oplus ((x \oplus y) \& \neg (x < y))$$

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>$y$</td>
<td>0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>$(x &lt; y)$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$-(x &lt; y)$</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>$(x \oplus y)$</td>
<td>1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0</td>
</tr>
<tr>
<td>$(x \oplus y) &amp; \neg (x &lt; y)$</td>
<td>1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0</td>
</tr>
<tr>
<td>$A = x \oplus ((x \oplus y) &amp; \neg (x &lt; y))$</td>
<td>0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

$$x < y \implies -(x < y) = -1 = 1 \ldots 1 \implies A = x \oplus (x \oplus y) = y$$

$$x \geq y \implies -(x < y) = 0 = 0 \ldots 0 \implies A = x \oplus 0 = x$$
Problem

- Count set bits in unsigned int $x$.

```
for (A = 0; x; A++) x ← x & (x − 1)
```

<table>
<thead>
<tr>
<th>Count</th>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 0</td>
<td>$x$</td>
<td>100011101011100101</td>
</tr>
<tr>
<td>A = 1</td>
<td>$x = x &amp; (x − 1)$</td>
<td>100011101011100100</td>
</tr>
<tr>
<td>A = 2</td>
<td>$x = x &amp; (x − 1)$</td>
<td>100011101011100000</td>
</tr>
<tr>
<td>A = 3</td>
<td>$x = x &amp; (x − 1)$</td>
<td>100011101010100000</td>
</tr>
<tr>
<td>A = 4</td>
<td>$x = x &amp; (x − 1)$</td>
<td>100011101010000000</td>
</tr>
<tr>
<td>A = 5</td>
<td>$x = x &amp; (x − 1)$</td>
<td>100011100000000000</td>
</tr>
<tr>
<td>A = 6</td>
<td>$x = x &amp; (x − 1)$</td>
<td>100010000000000000</td>
</tr>
<tr>
<td>A = 7</td>
<td>$x = x &amp; (x − 1)$</td>
<td>100000000000000000</td>
</tr>
<tr>
<td>A = 8</td>
<td>$x = x &amp; (x − 1)$</td>
<td>000000000000000000</td>
</tr>
</tbody>
</table>
Compute \([\log_2 x]\) for an unsigned int \(x\).

for \((A = 0; x >>= 1; A++)\);

<table>
<thead>
<tr>
<th>Log</th>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A = 0)</td>
<td>(x)</td>
<td>00000000000101101</td>
</tr>
<tr>
<td>(A = 1)</td>
<td>(x &gt;&gt;= 1)</td>
<td>00000000000101110</td>
</tr>
<tr>
<td>(A = 2)</td>
<td>(x &gt;&gt;= 1)</td>
<td>00000000000101111</td>
</tr>
<tr>
<td>(A = 3)</td>
<td>(x &gt;&gt;= 1)</td>
<td>00000000000001011</td>
</tr>
<tr>
<td>(A = 4)</td>
<td>(x &gt;&gt;= 1)</td>
<td>00000000000000010</td>
</tr>
<tr>
<td>(A = 5)</td>
<td>(x &gt;&gt;= 1)</td>
<td>000000000000000001</td>
</tr>
<tr>
<td>(A = 6)</td>
<td>(x &gt;&gt;= 1)</td>
<td>000000000000000000</td>
</tr>
</tbody>
</table>
Round to next power of 2

Problem

- Round to next power of 2, i.e., $2^\lceil \log_2 x \rceil$ of an unsigned int $x$.

$x --; x| = x >> 1; x| = x >> 2; x| = x >> 4; x| = x >> 8; x| = x >> 16; x| = x >> 32; x ++;

<table>
<thead>
<tr>
<th>Term</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0</td>
</tr>
<tr>
<td>$x --$</td>
<td>0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1</td>
</tr>
<tr>
<td>$x</td>
<td>= x &gt;&gt; 1$</td>
</tr>
<tr>
<td>$x</td>
<td>= x &gt;&gt; 2$</td>
</tr>
<tr>
<td>$x</td>
<td>= x &gt;&gt; 4$</td>
</tr>
<tr>
<td>$x</td>
<td>= x &gt;&gt; 8$</td>
</tr>
<tr>
<td>$x</td>
<td>= x &gt;&gt; 16$</td>
</tr>
<tr>
<td>$x</td>
<td>= x &gt;&gt; 32$</td>
</tr>
<tr>
<td>$x ++$</td>
<td>0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

$x --$ is used to correctly handle powers of 2.
Step 1. Problem

Problem

- Multiply two \((n - 1)\)-degree polynomials.
  For simplicity, we assume \(n\) is a power of 2.
- Formally, let \(A(x)\) and \(B(x)\) be \((n - 1)\)-degree polynomials. Compute \((2n - 2)\)-degree polynomial \(C(x)\) such that

\[
C(x) = A(x) \times B(x)
\]

where

\[
A(x) = a_0 + a_1 x^1 + \cdots + a_{n-1} x^{n-1}
\]

\[
B(x) = b_0 + b_1 x^1 + \cdots + b_{n-1} x^{n-1}
\]

\[
C(x) = c_0 + c_1 x^1 + \cdots + c_{2n-2} x^{2n-2}
\]
Step 2. Subproblem

$$\text{Mult}(A[\ell..h], B[\ell..h]) = \text{Multiply two } (h - \ell) \text{ degree polynomials } A[\ell..h] \text{ and } B[\ell..h].$$

Compute \( \text{Mult}(A[0..n - 1], B[0..n - 1]) \).
Step 3. Core idea

\[ A \times B = (A_L A_R) \times (B_L B_R) \]
\[ = (A_L + A_R \cdot x^{n/2}) \times (B_L + B_R \cdot x^{n/2}) \]
\[ = (A_L \times B_L) + (A_L \times B_R + A_R \times B_L) \cdot x^{n/2} \]
\[ + (A_R \times B_R) \cdot x^n \]
\[ = (A_L \times B_L) \]
\[ + \left( (A_L + A_R) \times (B_L + B_R) \right) \cdot x^{n/2} \]
\[ + (A_R \times B_R) \cdot x^n \]
Step 3. Core idea

\[ \text{MULT}(n) \]

\[
A_L A_R, \quad B_L B_R
\]

\[
A_L + A_R, \quad B_L + B_R
\]

\[
A_R, \quad B_R
\]

\[
P_1\]

\[
P_2\]

\[
P_3\]

\[
C
\]
Step 4. Example

Consider

\[ A(x) = [-6, 11, -6, 1] = -6 + 11x - 6x^2 + x^3 \]
\[ B(x) = [-120, 74, -15, 1] = -120 + 74x - 15x^2 + x^3 \]

Now consider \( A(x) \cdot B(x) \):

\[
[-6, 11, -6, 1] \times [-120, 74, -15, 1] \\
= ([-6, 11] + [-6, 1]x^2) \times ([-120, 74] + [-15, 1]x^2) \\
= [-6, 11] \times [-120, 74] \\
\hspace{1cm} + ([-6, 11] \times [-15, 1] + [-6, 1] \times [-120, 74])x^2 \\
\hspace{1cm} + ([-6, 1] \times [-15, 1])x^4 \\
= [-6, 11] \times [-120, 74] + \\
\hspace{1cm} + \left( ([-6, 11] + [-6, 1]) \times ([-120, 74] + [-15, 1]) - ([6, 1] \times [-15, 1]) \right) \cdot x^2 \\
\hspace{1cm} + ([-6, 1] \times [-15, 1])x^4 \]
Step 5. Algorithm

**KaratsubaProduct**(*A[ℓ...h], B[ℓ...h]*)

**Input:** Two \((h - ℓ)\)-degree polynomials \(A\) and \(B\), where \(ℓ\) and \(h\) are the lower and higher order coefficients

**Output:** Product of polynomials \(A\) and \(B\)

**if** \(ℓ = h\) **then** **return** \(A[ℓ] \times B[ℓ]\)

\(\text{mid} \leftarrow \lfloor (h + ℓ)/2 \rfloor; n \leftarrow h - ℓ + 1\)

\(A_L \leftarrow A[ℓ...\text{mid}], A_R \leftarrow A[\text{mid} + 1...h]\)

\(B_L \leftarrow B[ℓ...\text{mid}], B_R \leftarrow B[\text{mid} + 1...h]\)

**parallel:**

\(P_1 \leftarrow \text{KaratsubaProduct}(A_L, B_L)\)

\(P_2 \leftarrow \text{KaratsubaProduct}((A_L + A_R), (B_L + B_R))\)

\(P_3 \leftarrow \text{KaratsubaProduct}(A_R, B_R)\)

**return** \((P_1 + (P_2 - P_1 - P_3) \cdot x^{n/2} + P_3 \cdot x^n)\)
Step 6. Complexity

Work $T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1, \\
3T(n/2) + \Theta(n) & \text{if } n > 1. 
\end{cases} \in \Theta(n^{\log_2 3})$

Depth $D(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1, \\
D(n/2) + \Theta(n) & \text{if } n > 1. 
\end{cases} \in \Theta(n)$

Space $S(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1, \\
3S(n/2) + \Theta(n) & \text{if } n > 1. 
\end{cases} \in \Theta(n^{\log_2 3})$

Cache $Q(n) = \begin{cases} 
O(M/B) & \text{if } n \leq \gamma M, \\
3Q(n/2) + \Theta(n/B) & \text{if } n > \gamma M. 
\end{cases} \in O\left(\frac{n^{\log_2 3}}{MB}\right)
1. Coefficient representation
- $(n - 1)$-degree polynomial can be represented using $n$ coefficients
- $A(x) = a_0 + a_1 x^1 + \cdots + a_{n-1} x^{n-1} = \sum_{i=0}^{n-1} a_i x^i$
- $A(x) = [a_0, a_1, \ldots, a_{n-1}]$ \(\triangleright\) coefficient vector

2. Root representation
- $(n - 1)$-degree polynomial can be represented using $n - 1$ roots
- $A(x) = c(x - r_1)(x - r_1) \cdots (x - r_{n-1})$
- $A(x) = [c, \{r_1, r_1, \ldots, r_{n-1}\}]$\(\triangleright\) set of roots

3. Point representation
- $(n - 1)$-degree polynomial can be represented using $n$ points
- $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ where $y_i = A(x_i)$
- $A(x)$ is the set of these sample points \(\triangleright\) set of samples
1. Coefficient representation
   • 3-degree polynomial can be represented using 4 coefficients
   • \( A(x) = -6 + 11x - 6x^2 + x^3 \)
   • \( A(x) = [-6, 11, -6, 1] \) ▷ coefficient vector

2. Root representation
   • 3-degree polynomial can be represented using 3 roots
   • \( A(x) = 1(x - 1)(x - 2)(x - 3) \)
   • \( A(x) = [1, \{1, 2, 3\}] \) ▷ set of roots

3. Point representation
   • 4-degree polynomial can be represented using 4 points
   • \( \{(0, -6), (10, 504), (20, 5814), (30, 21924)\} \)
   • \( A(x) \) is the set of these sample points ▷ set of samples
Operations on polynomials

- **Coeff.**
  - add: $\Theta(n)$
  - mult: $\Theta\left(n^{\log_2 3}\right)$

- **Root**
  - add: $\Theta(n^2)$
  - mult: $\Theta(n^2)$

- **Point**
  - add: $\Theta(n)$
  - mult: $\Theta(n)$

- Connections:
  - $\Theta(n^2)$ from Coeff. to Point
  - $\Theta(n^2)$ from Root to Coeff.
  - $\Theta(n^2)$ from Root to Point
  - $\Theta(n)$ from Coeff. to Root
  - $\Theta(n)$ from Point to Root
  - $\infty$ from Coeff. to Point
  - $\infty$ from Root to Point
  - $\infty$ from Point to Coeff.
• Root representation is not very useful. Let’s remove it.
• Polynomial multiplication can be done in two different ways:
  1. Multiply in coefficient representation using Karatsuba’s idea
  2. Convert coefficient to point representation
     Multiply in point representation
     Convert point to coefficient representation
**Evaluation and interpolation**

\[
\begin{bmatrix}
A(x_0) \\
A(x_1) \\
A(x_2) \\
\vdots \\
A(x_{n-1})
\end{bmatrix}
= 
\begin{bmatrix}
1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\
1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\
1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{bmatrix}
\]

\[
X = \{x_0, x_1, \ldots, x_{n-1}\} \quad \text{and} \quad Y = V_X A
\]

- **Evaluation.**
  Convert coefficient to point representation.
  \(A\) is known, \(X\) is chosen, \(Y\) is computed.
  \(Y\) can be computed in \(\Theta(n^2)\) time using **Horner’s formula**.

- **Interpolation.**
  Convert point to coefficient representation.
  \(X\) and \(Y\) are known, \(A\) is computed.
  \(A\) can be computed in \(\Theta(n^2)\) time using **Lagrange’s formula**.
## Evaluation and interpolation

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Can we perform evaluation &amp; interpolation better than $\Theta(n^2)$?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Great idea</th>
</tr>
</thead>
<tbody>
<tr>
<td>• We can perform evaluation and interpolation in $\Theta(n \log n)$ using roots of 1 and divide-and-conquer.</td>
</tr>
<tr>
<td>• Evaluation of $(n - 1)$-degree polynomial $A(x)$ at $n$ roots of unity can be done in $\Theta(n \log n)$ using divide-and-conquer.</td>
</tr>
<tr>
<td>• Interpolation of $n$ roots of unity to an $(n - 1)$-degree polynomial can be done in $\Theta(n \log n)$ using divide-and-conquer.</td>
</tr>
</tbody>
</table>
Step 3. Core idea

evaluation: $\Theta(n \log n)$

interpolation: $\Theta(n \log n)$

mult: $\Theta(n)$
Step 3. Core idea

- Core idea.

\[
A(x) = A^{\text{even}}(x^2) + xA^{\text{odd}}(x^2)
\]

\[
A(-x) = A^{\text{even}}(x^2) - xA^{\text{odd}}(x^2)
\]

- Example.

\[
7 + 3x + 2x^2 + 6x^3 = (7 + 2x^2) + x(3 + 6x^2)
\]

\[
7 + 3(-x) + 2(-x)^2 + 6(-x)^3 = (7 + 2x^2) - x(3 + 6x^2)
\]

- Interpretation.

If we have the results of \(A^{\text{even}}(x^2)\) and \(A^{\text{odd}}(x^2)\), we can compute \(A(x)\) and \(A(-x)\) in constant time. Because we take square roots repeatedly, we use roots of unity. This leads to the amalgamation of ideas from mathematics (roots of unity) and computation (divide-and-conquer).
Roots of unity

- \( n \) roots of unity are the \( n \) solutions to equation \( x^n = 1 \).
- \( n \) roots of unity are \( \omega_0^n, \omega_1^n, \ldots, \omega_{n-1}^n \), where
  \[
  \omega_k^n = e^{k(2\pi i)/n} = \cos\left(k\left(\frac{2\pi}{n}\right)\right) + i \sin\left(k\left(\frac{2\pi}{n}\right)\right)
  \]
  and \( i = \sqrt{-1} \).
Step 3. Core idea

- \( \Theta(n \log n) \) evaluation: Use \( X = \{ \omega_0^n, \omega_1^n, \ldots, \omega_{n-1}^n \} \).

\[
\begin{bmatrix}
A(\omega_1^n) \\
A(\omega_2^n) \\
\vdots \\
A(\omega_{n-1}^n)
\end{bmatrix}
= \begin{bmatrix}
1 & \omega_0^n & (\omega_0^n)^2 & \cdots & (\omega_0^n)^{n-1} \\
1 & \omega_1^n & (\omega_1^n)^2 & \cdots & (\omega_1^n)^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega_{n-1}^n & (\omega_{n-1}^n)^2 & \cdots & (\omega_{n-1}^n)^{n-1}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{bmatrix}
\]

- \( \Theta(n \log n) \) interpolation: Use \( X = \frac{1}{n} \{ \omega_0^{-1}, \omega_1^{-1}, \ldots, \omega_{n-1}^{-(n-1)} \} \).

\[
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{bmatrix}
= \frac{1}{n}
\begin{bmatrix}
1 & \omega_0^{-1} & (\omega_0^{-1})^2 & \cdots & (\omega_0^{-1})^{n-1} \\
1 & \omega_1^{-1} & (\omega_1^{-1})^2 & \cdots & (\omega_1^{-1})^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega_{n-1}^{-(n-1)} & (\omega_{n-1}^{-(n-1)})^2 & \cdots & (\omega_{n-1}^{-(n-1)})^{n-1}
\end{bmatrix}
\begin{bmatrix}
A(\omega_1^n) \\
A(\omega_2^n) \\
\vdots \\
A(\omega_{n-1}^n)
\end{bmatrix}
\]
Step 3. Core idea

Evaluation FFT($n$)

Divide

Combine

$[a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7]$

$[y'_0, y'_1, y'_2, y'_3]$

$[y''_0, y''_1, y''_2, y''_3]$

$[y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7]$
Step 3. Core idea

Interpolation $\text{INVERSEFFT}(n)$

- $[y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7]$
- Divide
  - $[y_0, y_2, y_4, y_6]$
    - $[a'_0, a'_1, a'_2, a'_3]$
  - $[y_1, y_3, y_5, y_7]$
    - $[a''_0, a''_1, a''_2, a''_3]$
- Combine
  - $[a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7]$
Step 4. Example (Evaluation)

\[ [a_0, a_1, a_2, a_3] = [7, 3, 2, 6] \]

\[ [a_0, a_1] = [7, 2] \]

\[ a_0 = 7 \]

\[ y_0 = 7 \]

\[ y_0 = 7 + 2\omega_2^0 \]

\[ y_1 = 7 - 2\omega_2^0 \]

\[ y_0 = (7 + 2\omega_2^0) + (3 + 6\omega_2^0)\omega_4^0 \]

\[ y_1 = (7 - 2\omega_2^0) + (3 - 6\omega_2^0)\omega_4^1 \]

\[ y_2 = (7 + 2\omega_2^0) - (3 + 6\omega_2^0)\omega_4^0 \]

\[ y_3 = (7 - 2\omega_2^0) - (3 - 6\omega_2^0)\omega_4^1 \]

\[ y_0 = 7 + 3\omega_4^0 + 2(\omega_4^0)^2 + 6(\omega_4^0)^3 \]

\[ y_1 = 7 + 3\omega_4^1 + 2(\omega_4^1)^2 + 6(\omega_4^1)^3 \]

\[ y_2 = 7 + 3\omega_4^2 + 2(\omega_4^2)^2 + 6(\omega_4^2)^3 \]

\[ y_3 = 7 + 3\omega_4^3 + 2(\omega_4^3)^2 + 6(\omega_4^3)^3 \]
Step 5. Algorithm

\[
\text{FFT}([a_0, a_1, \ldots, a_{n-1}])
\]

**Input:** Coefficients of polynomial \( A(x) \): \([a_0, a_1, \ldots, a_{n-1}]\)

**Output:** Point values vector \( Y \) for \( X \) values \([\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}]\)

If \( n = 1 \) then return \( a_0 \)

\( \omega_n \leftarrow e^{2\pi i/n} \)

\( \omega \leftarrow 1 \)

// [Stage 1. Divide] ...........................................................
\( A^{\text{even}} \leftarrow [a_0, a_2, \ldots, a_{n-2}] \)
\( A^{\text{odd}} \leftarrow [a_1, a_3, \ldots, a_{n-1}] \)

// [Stage 2. Conquer] ..................................................
parallel: \( Y^{\text{even}} \leftarrow \text{FFT}(A^{\text{even}}) \)
\( Y^{\text{odd}} \leftarrow \text{FFT}(A^{\text{odd}}) \)

// [Stage 3. Combine] ..................................................
for \( k \leftarrow 0 \) to \( n/2 - 1 \) do

\( y_k \leftarrow Y_k^{\text{even}} + \omega Y_k^{\text{odd}} \)
\( y_{n/2+k} \leftarrow Y_k^{\text{even}} - \omega Y_k^{\text{odd}} \)
\( \omega \leftarrow \omega \omega_n \)

return \([y_0, y_1, \ldots, y_{n-1}]\)
**Step 5. Algorithm**

\[ \text{INVERSEFFT}([y_0, y_1, \ldots, y_{n-1}]) \]

**Input:** Point values vector \( Y \) for \( X \) values \([\omega_0^n, \omega_1^n, \ldots, \omega_{n-1}^n]\)

**Output:** Coefficients of polynomial \( A(x) : [a_0, a_1, \ldots, a_{n-1}] \)

**if** \( n = 1 \) **then** return \( y_0 \)

\( \omega_n \leftarrow \left( 1 / n \right) e^{-2\pi i / n} \)

\( \omega \leftarrow 1 \)

**// [Stage 1. Divide]..................................................**

\( Y^{\text{even}} \leftarrow [y_0, y_2, \ldots, y_{n-2}] \)

\( Y^{\text{odd}} \leftarrow [y_1, y_3, \ldots, y_{n-1}] \)

**// [Stage 2. Conquer]...............................................

**parallel:** 
\( A^{\text{even}} \leftarrow \text{INVERSEFFT}(Y^{\text{even}}) \)

\( A^{\text{odd}} \leftarrow \text{INVERSEFFT}(Y^{\text{odd}}) \)

**// [Stage 3. Combine]..............................................**

**for** \( k \leftarrow 0 \) **to** \( n/2 - 1 \) **do**

\( a_k \leftarrow A^{\text{even}}_k + \omega A^{\text{odd}}_k \)

\( a_{n/2+k} \leftarrow A^{\text{even}}_k - \omega A^{\text{odd}}_k \)

\( \omega \leftarrow \omega \omega_n \)

**return** \([a_0, a_1, \ldots, a_{n-1}]\)
CooleyTukeyProduct\((A(x), B(x))\)

**Input:** Polynomials \(A(x)\) and \(B(x)\) of same degree

**Output:** Polynomial product \(C(x) = A(x) \times B(x)\)

\[
\begin{align*}
\{a_0, a_1, \ldots, a_{n-1}\} & \leftarrow \text{Coefficients}(A(x)) \\
\{b_0, b_1, \ldots, b_{n-1}\} & \leftarrow \text{Coefficients}(B(x))
\end{align*}
\]

// [Stage 1. Add high-order coefficients] ........................................

\[
\begin{align*}
\{a_n, a_{n+1}, \ldots, a_{2n-1}\} & \leftarrow [0, 0, \ldots, 0] \\
\{b_n, b_{n+1}, \ldots, b_{2n-1}\} & \leftarrow [0, 0, \ldots, 0]
\end{align*}
\]

// [Stage 2. Evaluate] ......................................................................

**parallel:** \[
\begin{align*}
\{y_0^A, y_1^A, \ldots, y_{2n-1}^A\} & \leftarrow \text{FFT}([a_0, a_1, \ldots, a_{2n-1}]) \\
\{y_0^B, y_1^B, \ldots, y_{2n-1}^B\} & \leftarrow \text{FFT}([b_0, b_1, \ldots, b_{2n-1}])
\end{align*}
\]

// [Stage 3. Pointwise multiply] .........................................................

**parallel:** \[
\begin{align*}
\text{for } k & \leftarrow 0 \text{ to } 2n - 1 \text{ do} \\
\{y_k^C\} & \leftarrow \{y_k^A\} \times \{y_k^B\}
\end{align*}
\]

// [Stage 4. Interpolate] ....................................................................

\[
\begin{align*}
\{c_0, c_1, \ldots, c_{2n-1}\} & \leftarrow \text{InverseFFT}([y_0^C, y_1^C, \ldots, y_{2n-1}^C]) \\
C(x) & \leftarrow [c_0, c_1, \ldots, c_{2n-1}]
\end{align*}
\]

**return** \(C(x)\)
Step 6. Complexity

**Work** $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases} \in \Theta(n \log n)$

**Depth** $D(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ D(n/2) + \Theta(n) & \text{if } n > 1. \end{cases} \in \Theta(n)$

**Space** $S(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2S(n/2) + \Theta(n) & \text{if } n > 1. \end{cases} \in \Theta(n \log n)$

**Cache** $Q(n) = \begin{cases} O(M/B) & \text{if } n \leq \gamma M, \\ 2Q(n/2) + \Theta(n/B) & \text{if } n > \gamma M. \end{cases} \in O\left(\frac{n}{B} \log \frac{n}{M}\right)$
Problem

- Given a positive integer greater than 1, check if the number is prime or not.
- A prime is a natural number greater than 1 that has no positive divisors other than 1 and itself.
- Input: \( n = 11 \)
  Output: prime
- Input: \( n = 15 \)
  Output: composite
If $n$ is divisible by any number in the range $[2, n - 1]$, then $n$ is composite, else, $n$ is prime

**Primality-NaiveAlgorithm**($n$)

\[
\text{for } i \leftarrow 2 \text{ to } n - 1 \text{ do} \\
\quad \text{if } n \mod i = 0 \text{ then} \\
\quad \quad \text{return composite} \\
\text{return prime}
\]

$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n), \Theta(1) \rangle$
If \( n \) is divisible by any number in the range \([2, n - 1]\), then \( n \) is composite, else, \( n \) is prime.

This is because a larger factor of \( n \) must be a multiple of a smaller factor that has been already checked.

\[
\text{Primality-SchoolAlgorithm}(n) \\
\text{for } i \leftarrow 2 \text{ to } \left\lfloor \sqrt{n} \right\rfloor \text{ do} \\
\quad \text{if } n \mod i = 0 \text{ then} \\
\quad \quad \text{return composite} \\
\text{return prime}
\]

\[
\langle \text{Time}, \text{Space} \rangle = \langle \mathcal{O}(\sqrt{n}), \Theta(1) \rangle
\]
Solutions → Optimized school algorithm

- All integers can be expressed as \( (6k + i) \), where \( i \in \{-1, 0, 1, 2, 3, 4\} \).
- Test whether \( n \) is divisible by 2 or 3. But 2 divides \( (6k + 0), (6k + 2), (6k + 4) \) and 3 divides \( (6k + 3) \). So, simply check if \( n \) is divisible by any number in the form \( (6k \pm 1) \) not greater than \( \sqrt{n} \).

```
Primality-OptimizedSchoolAlgorithm(n)

if n = 2 or n = 3 then return prime
if n mod 2 = 0 or n mod 3 = 0 then return composite

// Check if n is divisible by a number of the form 6k ± 1
for i ← 5 to (\lceil \sqrt{n} \rceil − 2) increment 6 do
    if n mod i = 0 then
        return composite // i = 6k − 1
    if n mod (i + 2) = 0 then
        return composite // i = 6k + 1
return prime
```

\( \langle \text{Time, Space} \rangle = \langle \mathcal{O} (\sqrt{n}), \Theta (1) \rangle \)
### Primality-SieveOfEratosthenes(n)

\[ \text{last} \leftarrow \lfloor \sqrt{n} \rfloor \]

Create a Boolean array \( P[2 \ldots \text{last}] \) to indicate prime numbers

```plaintext
for i ← 2 to last do
    P[i] ← true

for j ← 2 to last do
    if P[j] = true then
        for k ← 2 to \( \lfloor \frac{\text{last}}{j} \rfloor \) do
            i ← j × k
        P[i] ← false
        if n mod j = 0 then
            return composite
    
return prime
```

\[ \langle \text{Time, Space} \rangle = \langle \mathcal{O} (\sqrt{n} \log \log n) , \Theta (\sqrt{n}) \rangle \]
Wilson’s theorem: A positive integer $n > 1$ is prime iff $((n - 1)! + 1) \mod n = 0$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$(n - 1)!$</th>
<th>$((n - 1)! + 1) \mod n$</th>
<th>Is Prime?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>✅</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>✅</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>2</td>
<td>✗</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
<td>0</td>
<td>✅</td>
</tr>
<tr>
<td>6</td>
<td>120</td>
<td>1</td>
<td>✗</td>
</tr>
<tr>
<td>7</td>
<td>720</td>
<td>0</td>
<td>✅</td>
</tr>
<tr>
<td>8</td>
<td>5040</td>
<td>1</td>
<td>✗</td>
</tr>
<tr>
<td>9</td>
<td>40320</td>
<td>1</td>
<td>✗</td>
</tr>
<tr>
<td>10</td>
<td>362880</td>
<td>1</td>
<td>✗</td>
</tr>
<tr>
<td>11</td>
<td>3628800</td>
<td>0</td>
<td>✅</td>
</tr>
<tr>
<td>12</td>
<td>39916800</td>
<td>1</td>
<td>✗</td>
</tr>
<tr>
<td>13</td>
<td>479001600</td>
<td>0</td>
<td>✅</td>
</tr>
</tbody>
</table>
• Wilson’s theorem: A positive integer $n > 1$ is prime iff
$((n - 1)! + 1) \mod n = 0$

```plaintext
Primality-WilsonTheorem(n)

factorial ← 1
for $i \leftarrow 2$ to $n - 1$ do
    factorial ← (factorial × $i$) mod $n$
if (factorial + 1) = $n$ then return prime
return composite
```

$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$
• $n \geq 4$ is prime iff for all $a \in [2, n - 2]$, we have $(a^{n-1} - 1) \mod n = 0.$

<table>
<thead>
<tr>
<th>$n : a$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>


Solutions $\rightarrow$ Fermat’s theorem

<table>
<thead>
<tr>
<th><strong>Primality-FermatTheorem</strong>($n$)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>if</strong> $n = 2$ or $n = 3$ <strong>then</strong> return <strong>prime</strong></td>
</tr>
<tr>
<td><strong>for</strong> $a \leftarrow 2$ <strong>to</strong> $n - 2$ <strong>do</strong></td>
</tr>
<tr>
<td>// If $(a^{n-1} - 1) \mod n \neq 0$, then $n$ is definitely composite</td>
</tr>
<tr>
<td><strong>if</strong> <strong>POWER</strong>($a, n - 1, n$) $\neq 1$ <strong>then</strong></td>
</tr>
<tr>
<td><strong>return</strong> <strong>composite</strong></td>
</tr>
<tr>
<td><strong>return</strong> <strong>prime</strong></td>
</tr>
</tbody>
</table>

$\langle \text{Time, Space} \rangle = \langle O(n \log n), \Theta(1) \rangle$
Power function using repeated squaring

\[ \text{Output: Computes } (a^b) \mod c \text{ in } \Theta (\log b) \text{ time} \]
\[ result \leftarrow 1 \]
\[ \textbf{while } b > 0 \textbf{ do} \]
\[ \quad \textbf{if } b \mod 2 = 1 \textbf{ then } result \leftarrow (result \times a) \mod c \]
\[ \quad b = b/2; \; a = (a \times a) \mod c \]
\[ \textbf{return } result \]

\[ \langle \text{Time, Space} \rangle = \langle \Theta (\log b) , \Theta (1) \rangle \]
Solutions → Fermat’s test

If a cell in the $n$th row of the table is nonzero, then $n$ is definitely composite.

Bad news.

- If a cell in the $n$th row of the table is $0$, then $n$ may or may not be prime.
- Formally, for all $n \geq 4$, for some $a \in [2, n - 2]$, if $(a^{n-1} - 1) \mod n = 0$, then $n$ may or may not be prime.
- Example: Cell in $n = 13$, $a = 8$ is zero and $n$ is prime
  - Example: Cell in $n = 15$, $a = 11$ is zero but $n$ is composite

Good news.

- There are very few cases when $n$ is composite and it has some cells as zeros in its row
- So, we run this check multiple times to increase our success probability of guessing whether $n$ is prime or composite
Solutions → Fermat’s test

\[
\textbf{Primality-FermatTest}(n)
\]

\begin{itemize}
\item \textbf{if} $n = 2$ \textbf{or} $n = 3$ \textbf{then} return prime
\item \textbf{// More trials increases the probability of success}
\item \textbf{for} \hspace{0.5em} \texttt{count} ← 1 \textbf{to} \hspace{0.5em} \#trials \textbf{do}
\item \hspace{1em} \texttt{a} ← \texttt{RandomNumber}($\{2, 3, 4, \ldots, n−2\}$)
\item \hspace{1em} \textbf{// If} $(a^{n-1} - 1) \mod n \neq 0$, \textbf{then} \hspace{0.5em} $n$ \textbf{is definitely composite}
\item \hspace{1em} \textbf{if} \hspace{0.5em} \texttt{POWER}(a, \hspace{0.5em} n−1, \hspace{0.5em} n) \neq 1 \textbf{then}
\item \hspace{1.5em} \textbf{return} composite
\item \textbf{return} prime \hspace{1.6em} \texttt{// may or may not be prime}
\end{itemize}

\[\langle \text{Time, Space} \rangle = \langle O(\#\text{trials} \cdot \log n), \Theta(1) \rangle\]
Miller’s theorem:
Suppose \( p \) is an odd prime. Let \( p - 1 = 2^k \cdot m \), where \( m \) is odd. Then, for every \( a \in [2, p - 2] \), either
- \( a^m \equiv 1 \pmod{p} \) or
- \( a^{2^i \cdot m} \equiv -1 \pmod{p} \) for some \( i \in [0, k - 1] \).
For odd integer \( n > 1 \), \( n - 1 = 2^k m \), where \( k \geq 1 \) and \( m \) is odd

\[
(x^{2^km} - 1) = ((x^{2^{k-1}m})^2 - 1)
\]

\[
= (x^{2^{k-1}m} - 1)(x^{2^{k-1}m} + 1)
\]

\[
= (x^{2^{k-2}m} - 1)(x^{2^{k-2}m} - 1)(x^{2^{k-1}m} + 1)
\]

\[
\vdots
\]

\[
= (x^m - 1)(x^m + 1)(x^{2m} + 1)(x^{4m} + 1) \cdots (x^{2^{k-1}m} + 1)
\]

If \( n \) is prime and \( a \in [1, n - 1] \), then \( a^{n-1} - 1 \equiv 0 \pmod{n} \) by Fermat’s theorem, so, using the factorization above we get

\[
(a^m - 1)(a^m + 1)(a^{2m} + 1)(a^{4m} + 1) \cdots (a^{2^{k-1}m} + 1) \equiv 0 \pmod{n}
\]

When \( n \) is odd prime, one of these factors must be \( 0 \pmod{n} \), so

\[
a^m \equiv 1 \pmod{n} \text{ or } a^{2^i m} \equiv -1 \pmod{n} \text{ for some } i \in [0, \ldots k - 1]
\]
## Solutions → Miller’s theorem

$a \in [2, n - 2]$

<table>
<thead>
<tr>
<th>$n$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>−1</td>
<td>0</td>
<td>−1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>−1</td>
<td>−1</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>−1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>−1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>−1</td>
<td>−1</td>
<td>−1</td>
<td>0</td>
<td>−1</td>
<td>0</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>−1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td>23</td>
<td>−1</td>
<td>−1</td>
<td>−1</td>
<td>0</td>
<td>−1</td>
<td>0</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>1</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
</tbody>
</table>

### Comments

- Rows for prime numbers have no dashes
- Rows for composite numbers have at least one dash

<table>
<thead>
<tr>
<th>Cell</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>−1</td>
<td>✓</td>
<td>?</td>
<td>Test 1 passed; We don’t care about Test 2</td>
</tr>
<tr>
<td>≥ 0</td>
<td>×</td>
<td>✓</td>
<td>Cell = least $i \in [0, k - 1]$ that passes Test 2</td>
</tr>
<tr>
<td>−</td>
<td>×</td>
<td>×</td>
<td>Test 1 failed; Test 2 failed</td>
</tr>
</tbody>
</table>
Solutions → Miller’s theorem

PRIMALITY-MILLERTHEOREM(n)

\[
\text{if } n > 2 \text{ and } n \text{ is even then return composite}
\]

// Find exponent k and odd number m such that \((n - 1) = 2^k \times m\)
\(k \leftarrow 0; m \leftarrow n - 1\)

\[\textbf{while } m \mod 2 = 0 \textbf{ do } \{ m \leftarrow m/2; k \leftarrow k + 1 \} \]

// Apply Miller’s theorem. \(a = 1 \& a = n - 1\) are redundant.
\[\textbf{for } a \leftarrow 2 \textbf{ to } n - 2 \textbf{ do} \]
\[T \leftarrow \text{POWER}(a, m, n)\]

// Check test 1; If test 1 fails, check test 2 for \(i = 0\)
\[\textbf{if } T = 1 \textbf{ or } T = n - 1 \textbf{ then continue}\]

// Check test 2 for \(i \in [1, k - 1]\)
\[\textbf{for } i \leftarrow 1 \textbf{ to } k - 1 \textbf{ do} \]
\[T \leftarrow \text{POWER}(T, 2, n)\]

// If \(T = 1\), we only get 1’s for future values of \(i\)
\[\textbf{if } T = 1 \textbf{ then return composite}\]
\[\textbf{if } T = n - 1 \textbf{ then break}\]
\[\textbf{if } T \neq n - 1 \textbf{ then return composite}\]

\[\text{return prime}\]

\(\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n \log^2 n), \Theta(1) \rangle\)
Solutions → Miller-Rabin’s test

\[ \text{Primality-MillerRabinTest}(n) \]

\[
\begin{align*}
\text{if } n > 2 \text{ and } n \text{ is even then return composite} \\
\text{// Find exponent } k \text{ and odd number } m \text{ such that } (n - 1) = 2^k \times m \\
k \leftarrow 0; \ m \leftarrow n - 1 \\
\text{while } m \text{ mod } 2 = 0 \text{ do } \{ \ m \leftarrow m/2; \ k \leftarrow k + 1 \ \} \\
\text{// Apply Miller's constraints in a randomized way as suggested by Rabin} \\
\text{for count} \leftarrow 1 \text{ to } \#\text{trials} \text{ do} \\
\quad a \leftarrow \text{RandomNumber}(\{2, 3, 4, \ldots, n - 2\}) \\
\quad T \leftarrow \text{Power}(a, m, n) \\
\quad \text{// Check test 1; If test 1 fails, check test 2 for } i = 0 \\
\quad \text{if } T = 1 \text{ or } T = n - 1 \text{ then continue} \\
\quad \text{// Check test 2 for } i \in [1, k - 1] \\
\quad \text{for } i \leftarrow 1 \text{ to } k - 1 \text{ do} \\
\quad \quad T \leftarrow \text{Power}(T, 2, n) \\
\quad \quad \text{if } T = 1 \text{ then return composite} \\
\quad \quad \text{if } T = n - 1 \text{ then break} \\
\quad \quad \text{if } T \neq n - 1 \text{ then return composite} \\
\text{return prime}
\end{align*}
\]

\[ \langle \text{Time, Space} \rangle = \langle \mathcal{O}(\#\text{trials} \cdot \log^2 n), \Theta(1) \rangle \]
• $n \geq 2$ is prime iff all coefficients, except first and last, of the $n$th row in the Pascal’s triangle are multiples of $n$
• $n \geq 2$ is prime iff for all $i \in [1, n - 1]$, $nC_i$ is a multiple of $n$.

<table>
<thead>
<tr>
<th>Primality-NaiveAKSTest($n$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>// Step 1. Compute all required binomial coefficients</td>
</tr>
<tr>
<td>$r \leftarrow \lfloor n/2 \rfloor$ // binomial coefficients are symmetric</td>
</tr>
<tr>
<td>Create an array $C[0 \ldots r]$</td>
</tr>
<tr>
<td>for $i \leftarrow 0$ to $n$ do</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>// Step 2. Check if the binomial coefficients are multiples of $n$</td>
</tr>
<tr>
<td>for $j \leftarrow 1$ to $r$ do</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>return prime</td>
</tr>
</tbody>
</table>

$\langle$Time, Space$\rangle = \langle \Theta \left(n^2\right), \Theta \left(n\right)\rangle$
## Complexity

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
<th>Probabilistic?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive algorithm</td>
<td>$O(n)$</td>
<td>$\Theta(1)$</td>
<td>$\times$</td>
</tr>
<tr>
<td>School algorithm</td>
<td>$O(\sqrt{n})$</td>
<td>$\Theta(1)$</td>
<td>$\times$</td>
</tr>
<tr>
<td>Opt. school algo.</td>
<td>$O(\sqrt{n})$</td>
<td>$\Theta(1)$</td>
<td>$\times$</td>
</tr>
<tr>
<td>SieveOfEratosthenes</td>
<td>$O(\sqrt{n} \log \log n)$</td>
<td>$\Theta(\sqrt{n})$</td>
<td>$\times$</td>
</tr>
<tr>
<td>Wilson’s theorem</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
<td>$\times$</td>
</tr>
<tr>
<td>Fermat’s theorem</td>
<td>$O(n \log n)$</td>
<td>$\Theta(1)$</td>
<td>$\times$</td>
</tr>
<tr>
<td>Fermat’s test</td>
<td>$O(#\text{trials} \cdot \log n)$</td>
<td>$\Theta(1)$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>Miller’s theorem</td>
<td>$O(n \log^2 n)$</td>
<td>$\Theta(1)$</td>
<td>$\times$</td>
</tr>
<tr>
<td>Miller-Rabin’s test</td>
<td>$O(#\text{trials} \cdot \log^2 n)$</td>
<td>$\Theta(1)$</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>Naive AKS test</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n)$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>
Problem

- Design a data structure to implement a set with add and search operations.

<table>
<thead>
<tr>
<th>Solutions</th>
<th>Add</th>
<th>Search</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balanced tree</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>works for sort-related ops.</td>
</tr>
<tr>
<td>Hash table</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>worst case is worse</td>
</tr>
<tr>
<td></td>
<td>$O(1)$*</td>
<td>$O(1)$*</td>
<td>amortized case is awesome</td>
</tr>
<tr>
<td>Bloom filter</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>has false positive errors</td>
</tr>
</tbody>
</table>

- There are many more solutions.
A probabilistic data structure to check set membership discovered by Burton Howard Bloom in 1970.

- Takes less space than a hash table and answers approximately
- **Has false positives but no false negatives**, i.e.,
  - If BF returns found/present, then there is a small chance that the item is not present
  - If BF returns not found, then the item is definitely not present

- Bloom filter has two main components:
  - A bit array $A[0 \ldots N - 1]$
  - Independent hash functions $h_1, h_2, \ldots, h_k$ such that $h_i : \Sigma^* \rightarrow \{0, 1, 2, \ldots, N - 1\}$ such that the mapping is uniform
**Bloom filter class**

<table>
<thead>
<tr>
<th>class</th>
<th><code>BloomFilter(n, p)</code></th>
</tr>
</thead>
</table>
| **Input:** | $n \leftarrow$ number of elements; $p \leftarrow$ desired false probability  
| | $k \leftarrow$ number of hash functions; $N \leftarrow$ Bloom filter/table size  
| | $A \leftarrow$ Bloom filter bit array/table of size $N$  
| | **Initialize(); Add(x); Search(x)** |
| **Initialize()** | $k \leftarrow \left\lceil \frac{-\ln p}{\ln 2} \right\rceil$; $N \leftarrow \left\lceil n \cdot \frac{k}{\ln 2} \right\rceil$  
| | $A[0 \ldots N - 1] \leftarrow [0 \ldots 0]$ |
| **Add(x)** | **for** $i \leftarrow 1$ **to** $k$ **do**  
| | | $A[h_i(x)] \leftarrow 1$ |
| **Search(x)** | **for** $i \leftarrow 1$ **to** $k$ **do**  
| | | **if** $A[h_i(x)] \neq 1$ **then**  
| | | | **return** false  
| | **return** true |
\textbf{ADD}(x)

\begin{itemize}
\item \textbf{for} \( i \leftarrow 1 \) \textbf{to} \( k \) \textbf{do}
\item \( A[h_i(x)] \leftarrow 1 \)
\end{itemize}

Source: Medjedovic-Tahirovic-Dedovic’s Algorithms and Data Structures for Massive Datasets
**Search**

\[ \text{Search}(x) \]

\[ \text{for } i \leftarrow 1 \text{ to } k \text{ do} \]
\[ \quad \text{if } A[h_i(x)] \neq 1 \text{ then} \]
\[ \quad \quad \text{return false} \]
\[ \text{return true} \]

Source: Medjedovic-Tahirovic-Dedovic’s Algorithms and Data Structures for Massive Datasets
Complexity analysis

- Given number of elements $n$
- Given the false positive rate $p$
- Compute the number of hash functions as
  \[ k \leftarrow \left\lceil -\frac{\ln p}{\ln 2} \right\rceil \]
- Compute the Bloom filter bit array size $N$ as
  \[ N \leftarrow \left\lfloor n \cdot \frac{k}{\ln 2} \right\rfloor \]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>$O(k) = O(\ln (1/p))$</td>
</tr>
<tr>
<td>Search</td>
<td>$O(k) = O(\ln (1/p))$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(N) = O(nk) = O(n \ln (1/p))$</td>
</tr>
</tbody>
</table>
Error analysis

- Incorrect formula for computing the false positive prob. was given by [Bloom 1970] as

\[
\text{False positive prob. } = p = \left(1 - \left(1 - \frac{1}{N}\right)^{nk}\right)^k
\]

- Incorrect formula for computing the false positive prob. was given by [Bose et al. 2008] as

\[
\text{False positive prob. } = p^* = \left(\frac{1}{N^k(n+1)}\right) \cdot \sum_{i=1}^{m} i^k i! \cdot mC_i \cdot knC_i
\]

- Fortunately, the incorrect \( p \) gives a very good approximation to correct \( p^* \) for practical values.

- We show Bloom’s derivation to derive the incorrect \( p \) so that you can be careful when you do probabilistic analysis.
Error analysis

Prob. that a bit will be 0 after 1 insertion = \((1 - 1/N)^k\)
Prob. that a bit will be 0 after \(n\) insertions = \((1 - 1/N)^{nk}\)
Prob. that a bit will be 1 after \(n\) insertions = \((1 - (1 - 1/N)^{nk})\)
Prob. that \(k\) bits are 1 after \(n\) insertions = \((1 - (1 - 1/N)^{nk})^k\)

Simplify.

\[
\text{Prob. of false positives} = \text{Prob. that } k \text{ bits are 1 after } n \text{ insertions} = \left(1 - \left(1 - \frac{1}{N}\right)^{nk}\right)^k = \left(1 - \left(\left(1 - \frac{1}{N}\right)^N\right)^{\frac{nk}{N}}\right)^k \\
\approx \left(1 - e^{\frac{nk}{N}}\right)^k \quad (\because (1 - 1/x)^x \approx e)
\]

So,

False positive probability = \(p = \left(1 - e^{\frac{nk}{N}}\right)^k\)
## Error analysis

<table>
<thead>
<tr>
<th>$n$</th>
<th>$p = 0.0001$</th>
<th>$p = 0.001$</th>
<th>$p = 0.01$</th>
<th>$p = 0.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^1$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 13$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 10$</td>
<td>$N = \lfloor N_{0.01} n \rfloor, k = 7$</td>
<td>$N = \lfloor N_{0.1} n \rfloor, k = 3$</td>
</tr>
<tr>
<td>$10^2$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 13$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 10$</td>
<td>$N = \lfloor N_{0.01} n \rfloor, k = 7$</td>
<td>$N = \lfloor N_{0.1} n \rfloor, k = 3$</td>
</tr>
<tr>
<td>$10^3$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 13$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 10$</td>
<td>$N = \lfloor N_{0.01} n \rfloor, k = 7$</td>
<td>$N = \lfloor N_{0.1} n \rfloor, k = 3$</td>
</tr>
<tr>
<td>$10^4$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 13$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 10$</td>
<td>$N = \lfloor N_{0.01} n \rfloor, k = 7$</td>
<td>$N = \lfloor N_{0.1} n \rfloor, k = 3$</td>
</tr>
<tr>
<td>$10^5$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 13$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 10$</td>
<td>$N = \lfloor N_{0.01} n \rfloor, k = 7$</td>
<td>$N = \lfloor N_{0.1} n \rfloor, k = 3$</td>
</tr>
<tr>
<td>$10^6$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 13$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 10$</td>
<td>$N = \lfloor N_{0.01} n \rfloor, k = 7$</td>
<td>$N = \lfloor N_{0.1} n \rfloor, k = 3$</td>
</tr>
<tr>
<td>$10^7$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 13$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 10$</td>
<td>$N = \lfloor N_{0.01} n \rfloor, k = 7$</td>
<td>$N = \lfloor N_{0.1} n \rfloor, k = 3$</td>
</tr>
<tr>
<td>$10^8$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 13$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 10$</td>
<td>$N = \lfloor N_{0.01} n \rfloor, k = 7$</td>
<td>$N = \lfloor N_{0.1} n \rfloor, k = 3$</td>
</tr>
<tr>
<td>$10^9$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 13$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 10$</td>
<td>$N = \lfloor N_{0.01} n \rfloor, k = 7$</td>
<td>$N = \lfloor N_{0.1} n \rfloor, k = 3$</td>
</tr>
<tr>
<td>$10^{10}$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 13$</td>
<td>$N = \lfloor N_{0.0001} n \rfloor, k = 10$</td>
<td>$N = \lfloor N_{0.01} n \rfloor, k = 7$</td>
<td>$N = \lfloor N_{0.1} n \rfloor, k = 3$</td>
</tr>
</tbody>
</table>

- $N_{0.0001} = 19.170116754734877$, $N_{0.001} = 14.37758756605116$
- $N_{0.01} = 4.792529188683719$, $N_{0.1} = 0.9965784284662087$
- Note that Bloom filter bit array size $N$ is in bits
References

- BF simulator
- BF parameter calculator
- BF extensions
- BF applications
- BF false positive prob. analysis in [Bose et al. 2008]
## Problem

- Design a data structure that can estimate the frequencies of items.

## Solutions

<table>
<thead>
<tr>
<th></th>
<th>Update</th>
<th>Estimate</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balanced tree</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>works for sort-related ops.</td>
</tr>
<tr>
<td>Hash table</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>worst case is worse</td>
</tr>
<tr>
<td></td>
<td>$O(1)\ast$</td>
<td>$O(1)\ast$</td>
<td>amortized case is awesome</td>
</tr>
<tr>
<td>Count-min sketch</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>has false positive errors with approximations</td>
</tr>
</tbody>
</table>

- There are many more solutions.
Count-min sketch

- A probabilistic data structure to estimate frequencies, discovered by Graham Cormode and Shan Muthukrishnan in 2005
- Takes less space than a hash table and answers approximately and probabilistically
- **Always overestimates, never underestimates**, i.e., If CMS returns $x$, then $x$ is greater than or equal to actual frequency
- Count-min sketch has two main components:
  A 2-D count matrix $A[1 \ldots k, 1 \ldots w]$
  Independent hash functions $h_1, h_2, \ldots, h_k$ such that $h_i : \Sigma^* \rightarrow \{1, 2, \ldots, m\}$ such that the mapping is uniform
class \texttt{CountMinSketch}(\epsilon, \delta) \\

**Input:** \( \epsilon \leftarrow \) approximation parameter; \( \delta \leftarrow \) error probability parameter \\
\( k \leftarrow \) number of hash functions; \( w \leftarrow \) width of CMS \\
\( A \leftarrow \) 2-D CMS matrix of size \( k \times w \) \\

\textbf{Initialize()}; \textbf{Update}(x, c_x); \textbf{Estimate}(x) \\

**Initialize()** \\
\[
k \leftarrow \lceil \ln \frac{1}{\delta} \rceil; \ w \leftarrow \lceil \frac{\epsilon}{\epsilon} \rceil \\
A[1 \ldots k, 1 \ldots w] \leftarrow [0 \ldots 0, 0 \ldots 0]
\]

**Update\((x, c_x)\)** \\
\[
\text{for } i \leftarrow 1 \text{ to } k \text{ do} \\
\quad \quad A[i, h_i(x)] \leftarrow A[i, h_i(x)] + c_x
\]

**Estimate\((x)\)** \\
\[
\text{min} \leftarrow A[1, h_1(x)] \\
\text{for } i \leftarrow 2 \text{ to } k \text{ do} \\
\quad \quad \text{if } A[i, h_i(x)] < \text{min} \text{ then} \\
\quad \quad \quad \quad \text{min} \leftarrow A[i, h_i(x)] \\
\text{return } \text{min}
\]
Update

**UPDATE**($x, c_x$)

```plaintext
for i ← 1 to k do
    A[i, h_i(x)] ← A[i, h_i(x)] + c_x
```

Source: Medjedovic-Tahirovic-Dedovic’s Algorithms and Data Structures for Massive Datasets
**Estimate**

**Estimate**$(x)$

\[
\text{min} \leftarrow A[1, h_1(x)]
\]

\[
\text{for } i \leftarrow 2 \text{ to } k \text{ do}
\]

\[
\quad \text{if } A[i, h_i(x)] < \text{min} \text{ then}
\]

\[
\quad \text{min} \leftarrow A[i, h_i(x)]
\]

\[
\text{return } \text{min}
\]

Source: Medjedovic-Tahirovic-Dedovic’s Algorithms and Data Structures for Massive Datasets
Complexity analysis

- Given approximation fixed parameter $\epsilon$
- Given error probability fixed parameter $\delta$
- Compute the number of hash functions $k$ as
  $$k \leftarrow \lceil \ln \frac{1}{\delta} \rceil$$

- Compute the CMS matrix width size $w$ as
  $$w \leftarrow \lceil \frac{e}{\epsilon} \rceil$$

<table>
<thead>
<tr>
<th>Feature</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>$\mathcal{O}(k) = \mathcal{O}(1)$</td>
</tr>
<tr>
<td>Estimate</td>
<td>$\mathcal{O}(k) = \mathcal{O}(1)$</td>
</tr>
<tr>
<td>Space</td>
<td>$\mathcal{O}(mk) = \mathcal{O}(1)$</td>
</tr>
</tbody>
</table>
Let the data stream be \((a_1, c_1), (a_2, c_2), \ldots, (a_t, c_t)\)

Let \(N = \text{sum of all frequencies} = c_1 + c_2 + \cdots + c_t\)

Let \(f_{x}^{\text{true}} = \text{true frequency of item } x \text{ in CMS}\)

Let \(f_{x}^{\text{est}} = \text{estimated frequency of item } x \text{ in CMS}\)

Then

\[
\begin{align*}
\text{\(f_{x}^{\text{est}}\) is in} & \left\{ \begin{array}{l}
\left[ f_{x}^{\text{true}}, f_{x}^{\text{true}} + \epsilon \cdot N \right] \quad \text{with probability} \geq 1 - \delta \\
\left( f_{x}^{\text{true}} + \epsilon \cdot N, \infty \right) \quad \text{with probability} \leq \delta
\end{array} \right. \\
\end{align*}
\]

where, \(\epsilon, \delta \in (0, 1)\)
## Differences between Bloom filter and CMS

<table>
<thead>
<tr>
<th>Feature</th>
<th>Bloom filter</th>
<th>CMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicates</td>
<td>Set</td>
<td>Multiset</td>
</tr>
<tr>
<td>Array</td>
<td>1-D</td>
<td>2-D</td>
</tr>
<tr>
<td>Hash function</td>
<td>Maps to entire array</td>
<td>Maps to portions of 2-D array</td>
</tr>
<tr>
<td>Query</td>
<td>Existential queries</td>
<td>Counting queries</td>
</tr>
<tr>
<td>Size</td>
<td>Typ. linear w.r.t set size</td>
<td>Typ. sublinear w.r.t total freq</td>
</tr>
<tr>
<td>Randomness</td>
<td>Uniformly random</td>
<td>Uni. random &amp; pairwise ind.</td>
</tr>
</tbody>
</table>
Cardinality
Problem

- Given a data stream, compute the number of distinct elements efficiently.
- Input: [4, 8, 9, 4, 4, 8]
  Output: 3
Solutions → Brute force

1. Check all previous values for duplicates
2. Count a value only if no previous duplicate

BruteForce\((A[1\ldots n])\)

\[
\begin{align*}
\text{distinct} & \leftarrow 1 \\
\text{for } i & \leftarrow 2 \text{ to } n \text{ do} \\
& \quad j \leftarrow 1 \\
& \quad \text{while } j \leq i \text{ do} \\
& \quad \quad \text{// Check current element with previous } (j < i) \text{ or current } (j = i) \\
& \quad \quad \text{if } A[i] = A[j] \text{ then break} \\
& \quad \quad j \leftarrow j + 1 \\
& \quad \text{// If there is no previous duplicate, then increment distinct} \\
& \quad \text{if } j = i \text{ then} \\
& \quad \quad \text{distinct} \leftarrow \text{distinct} + 1 \\
\text{return } \text{distinct}
\end{align*}
\]

\[
\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n^2), \Theta(1) \rangle
\]
1. Sort the array
2. Equal values are together in the sorted input
3. Count the first occurrence of each value

\[ \text{\texttt{SortAndCount}}(A[1 \ldots n]) \]

\[
distinct \leftarrow 1 \\
\text{Sort}(A[1 \ldots n]) \\
\text{for } i \leftarrow 2 \text{ to } n \text{ do} \\
\quad \text{// Current value is different from previous value} \\
\quad \text{if } A[i] \neq A[i-1] \text{ then} \\
\quad \quad \text{distinct } \leftarrow \text{distinct } + 1 \\
\text{return distinct}
\]

\( \langle \text{Time, Space} \rangle = \langle \Theta(n \log n), \Theta(n) \rangle \)
1. Create a bit vector $B[1 \ldots U]$, where $U$ is the maximum value in the universe or in the array
2. For value $A[i]$, set $B[A[i]]$ to true
3. Count the number of true values in the bit vector

```
BitVector(A[1 \ldots n])

distinct \leftarrow 0
U \leftarrow \text{Max}(A[1 \ldots n]) \quad // \text{max element in the array/universe}
B[1 \ldots U] \leftarrow [0 \ldots 0]
// For value $A[i]$, set $B[A[i]]$ to true
\text{for } i \leftarrow 1 \text{ to } n \text{ do } B[A[i]] \leftarrow 1
// Count the number of true values in the bit vector
\text{for } i \leftarrow 1 \text{ to } U \text{ do }
| \quad \text{if } B[i] = 1 \text{ then } \text{distinct} \leftarrow \text{distinct} + 1
\text{return distinct}
```

$\langle \text{Time, Space} \rangle = \langle \Theta(n + U), \Theta(U) \rangle$, where $U \geq \text{Max}(A[1 \ldots n])$
Solutions → Hash set

1. Create a hash set to store unique values
2. Add each element to the hash set
3. Return the size of the hash set

\[
\text{HashSet}(A[1 \ldots n])
\]

Create a hash set \( H \) to store unique values

\[
\text{for } i \leftarrow 1 \text{ to } n \text{ do}
\]
\[
| \quad H.\text{Add}(A[i])
\]
\[
distinct \leftarrow H.\text{Size}() \quad // \quad \#\text{elements in the hash set}
\]

\[
\text{return } distinct
\]

\[
\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle
\]
• Like Bloom filter, linear counter is a bit vector of size $m$ and does not store hash keys
• Linear counter length $m$ is proportional to $n$ but requires 1 bit per element
• There can be hard collisions that are not handled
• #Distinct elements is estimated based on the observed fraction of empty bits in the set
1. Use a hash function $h$
2. Create and initialize a bit array of size proportional to $n$ to zeros
3. Set $h(A[1])$ index in the bit array to 1
4. Set $h(A[2])$ index in the bit array to 1
5. so on...
6. Set $h(A[n])$ index in the bit array to 1

#Distinct elements = #zeros in the bit array
\textbf{Solutions $\rightarrow$ Linear counting}

\begin{Verbatim}
\textbf{LinearCounting}(A[1 \ldots n])
\end{Verbatim}

\begin{itemize}
\item $m \leftarrow n$ \hspace{1cm} // assuming $n$ is known
\item $linearcounter[1 \ldots m] \leftarrow [0 \ldots 0]$
\item // Adding elements to the linear counter
\begin{itemize}
\item \textbf{for} $i \leftarrow 1$ \textbf{to} $n$ \textbf{do}
\item \hspace{0.5cm} $linearcounter[h(A[i])] \leftarrow 1$
\end{itemize}
\item // Compute the number of zeros in the linear counter
\begin{itemize}
\item $zerocount \leftarrow 0$
\item \textbf{for} $i \leftarrow 1$ \textbf{to} $n$ \textbf{do}
\item \hspace{0.5cm} $\textbf{if} linearcounter[i] = 0 \textbf{then}$
\item \hspace{1cm} $zerocount \leftarrow zerocount + 1$
\end{itemize}
\item // Estimate \#distinct elements using probabilistic analysis
\begin{itemize}
\item $distinct \leftarrow \left\lfloor -m \times \ln \left( \frac{zerocount}{m} \right) \right\rfloor$
\end{itemize}
\end{itemize}
\begin{itemize}
\item \textbf{return} $distinct$
\end{itemize}

$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$
1. Use a hash function $h$
2. Let $z_1 = (\text{#trailing zeros in } h(A[1]) ) + 1$
3. Let $z_2 = (\text{#trailing zeros in } h(A[2]) ) + 1$
4. so on...
5. Let $z_n = (\text{#trailing zeros in } h(A[n]) ) + 1$
6. Let $z_{\text{max}} = \text{Max}(z_1, z_2, \ldots, z_n)$

\[
\text{#Distinct elements} = 2^{z_{\text{max}}}
\]
#distinct=7, 16-bit hashes, estimated #distinct= \(2^5 = 32\)

Source: Medjedovic-Tahirovic-Dedovic’s Algorithms and Data Structures for Massive Datasets
Probabilistic Counting

\[
\text{ProbabilisticCounting}(A[1 \ldots n])
\]

\[
\begin{align*}
    z_{\text{max}} & \leftarrow 0 \quad \text{// denotes max \# trailing zeros in a hash value} \\
    \text{for } i & \leftarrow 1 \text{ to } n \text{ do} \\
        & z \leftarrow \text{CountTrailingZeros}(h(A[i])) \\
        & \text{if } z > z_{\text{max}} \text{ then } z_{\text{max}} \leftarrow z \\
    \text{distinct} & \leftarrow 2^{z_{\text{max}}} \\
    \text{return } \text{distinct} \\
\end{align*}
\]

\[
\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle
\]
Why does the algorithm work?

Among $k$ random generated bit strings
- $\approx k/2$ bit strings have 0 as the last digit
- $\approx k/2^2$ bit strings have 00 as the last digits
- $\approx k/2^3$ bit strings have 000 as the last digits
- $\approx k/2^j$ bit strings have $0 \ldots 0$ as the last digits
- $\approx 1/2^j$ occurring with probability
- $\approx 1/2^{z_{\max}}$ occurring with probability
Why does the algorithm work?

Among $k$ random generated bit strings

- $\approx k/2$ bit strings have 0 as the last digit
- $\approx k/2^2$ bit strings have 00 as the last digits
- $\approx k/2^3$ bit strings have 000 as the last digits
- $\approx k/2^j$ bit strings have $0\ldots0$ as the last digits

Probability of generating a hash value for item $A[i]$

- having $z_1 = 1$ (hash ends with 1) is $1/2$
- having $z_2 = 2$ (hash ends with 10) is $1/2^2$
- having $z_3 = 3$ (hash ends with 100) is $1/2^3$
- having $z_j = j$ (hash ends with $10\ldots0$) is $1/2^{j-1}$
Why does the algorithm work?

Among $k$ random generated bit strings

- $\approx k/2$ bit strings have 0 as the last digit
- $\approx k/2^2$ bit strings have 00 as the last digits
- $\approx k/2^3$ bit strings have 000 as the last digits
- $\approx k/2^j$ bit strings have $0\ldots0$ as the last digits

Probability of generating a hash value for item $A[i]$

- having $z_1 = 1$ (hash ends with 1) is $1/2$
- having $z_2 = 2$ (hash ends with 10) is $1/2^2$
- having $z_3 = 3$ (hash ends with 100) is $1/2^3$
- having $z_j = j$ (hash ends with 10$\ldots$0) is $1/2^j$

An event having prob. $1/2^j$ occurs if on avg. $2^j$ trials are performed

An event having prob. $1/2^{z_{\max}}$ occurs if on avg. $2^{z_{\max}}$ trials are performed
Problem:
- Probabilistic counting does not approximate well
- Idea: Use $m$ hash functions and take the average
- Flaw: But, using $m$ hash functions is very expensive
Problem:
- Probabilistic counting does not approximate well
- Idea: Use $m$ hash functions and take the average
- Flaw: But, using $m$ hash functions is very expensive

Idea: Bucketing
- Have $m = 2^b$ buckets
- Find the bucket using the last $b$ bits of the hash value
- Perform probabilistic counting with the remaining bits.
- Add the distinct items in all buckets
1. Let $z_1 = \text{estimator/prediction for bucket 1}$
2. Let $z_2 = \text{estimator/prediction for bucket 2}$
3. so on...
4. Let $z_m = \text{estimator/prediction for bucket } m$
5. Let $z_{\text{avg}} = \frac{z_1 + z_2 + \cdots + z_m}{m}$
   = average estimator/prediction of all buckets

\[ \#\text{Distinct elements} = \text{Round} \left( m \cdot 2^{z_{\text{avg}}} \right) \]
#distinct = 7, 16-bit hashes, 4 buckets, estimated

#distinct = Round \( (4 \cdot 2^{2.5}) = 23 \)

Source: Medjedovic-Tahirovic-Dedovic’s Algorithms and Data Structures for Massive Datasets
### Stochastic Averaging \( (A[1 \ldots n]) \)

\[
\text{STOCHASTIC_AVERAGING}(A[1 \ldots n])
\]

// Initialize estimators in \( m \) buckets
Create an array \( z_{\text{max}}[0, 1, \ldots, m - 1] \leftarrow [0, 0, \ldots, 0] \)

// Compute estimators in \( m \) buckets
for \( i \leftarrow 1 \) to \( n \) do

\[
\text{bucket} \leftarrow h(A[i]) \mod m
\]

\[
\text{buckethash} \leftarrow \lceil h(A[i]) / m \rceil
\]

\[
z \leftarrow \text{CountTrailingZeros}(\text{buckethash})
\]

if \( z > z_{\text{max}}[\text{bucket}] \) then

\[
z_{\text{max}}[\text{bucket}] \leftarrow z
\]

// Find the average of estimators in \( m \) buckets
\[
z_{\text{avg}} \leftarrow \frac{1}{m} \cdot (z_{\text{max}}[0] + z_{\text{max}}[1] + \cdots + z_{\text{max}}[m - 1])
\]

// Estimate the #distinct elements in all buckets
\[
distinct \leftarrow \text{Round} \left( m \cdot 2^{z_{\text{avg}}} \right)
\]

return distinct

\[\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle\]
1. Let $z_1 = \text{estimator/prediction for bucket 1}$
2. Let $z_2 = \text{estimator/prediction for bucket 2}$
3. so on...
4. Let $z_m = \text{estimator/prediction for bucket } m$
5. Let $z_{\text{avg}} = \frac{z_1 + z_2 + \cdots + z_m}{m}$
   $= \text{average estimator/prediction of all buckets}$

#Distinct elements $= \text{Round} \left( \alpha_m \cdot m \cdot 2^{z_{\text{avg}}} \right)$, where,

$$\alpha_m = \begin{cases} 
0.39701 - \frac{2\pi^2 + \ln^2 2}{48m} & \text{if } m < 64, \\
0.39701 & \text{if } m \geq 64.
\end{cases}$$
#distinct = 7, 16-bit hashes, 4 buckets, estimated

#distinct = \text{Round}\left(0.29169926137 \cdot 4 \cdot 2^{2.5}\right) = 7

Source: Medjedovic-Tahirovic-Dedovic's Algorithms and Data Structures for Massive Datasets
## Solutions → LogLog

<table>
<thead>
<tr>
<th>LogLog($A[1 \ldots n]$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>// Initialize estimators in $m$ buckets</td>
</tr>
<tr>
<td>Create an array $z_{\text{max}}[0, 1, \ldots, m - 1] \leftarrow [0, 0, \ldots, 0]$</td>
</tr>
<tr>
<td>// Compute estimators in $m$ buckets</td>
</tr>
<tr>
<td>for $i \leftarrow 1$ to $n$ do</td>
</tr>
<tr>
<td>$\quad$ bucket $\leftarrow h(A[i]) \mod m$ // determines bucket</td>
</tr>
<tr>
<td>$\quad$ buckethash $\leftarrow \lceil h(A[i])/m \rceil$ // determines hash in bucket</td>
</tr>
<tr>
<td>$\quad$ $z \leftarrow \text{CountTrailingZeros}(\text{buckethash})$</td>
</tr>
<tr>
<td>$\quad$ if $z &gt; z_{\text{max}}[\text{bucket}]$ then $z_{\text{max}}[\text{bucket}] \leftarrow z$</td>
</tr>
<tr>
<td>// Find the average of estimators in $m$ buckets</td>
</tr>
<tr>
<td>$z_{\text{avg}} \leftarrow \frac{1}{m} \cdot (z_{\text{max}}[0] + z_{\text{max}}[1] + \cdots + z_{\text{max}}[m - 1])$</td>
</tr>
<tr>
<td>// Estimate the #distinct elements in all buckets</td>
</tr>
<tr>
<td>$\text{distinct} \leftarrow \text{Round}(\alpha_m \cdot m \cdot 2^{z_{\text{avg}}})$</td>
</tr>
<tr>
<td>return $\text{distinct}$</td>
</tr>
</tbody>
</table>

$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$
1. Let \( z_1 \) = estimator/prediction for bucket 1
2. Let \( z_2 \) = estimator/prediction for bucket 2
3. so on...
4. Let \( z_m \) = estimator/prediction for bucket \( m \)
5. Let \( B = \frac{1}{2^{z_1} + \frac{1}{2^{z_2}} + \ldots + \frac{1}{2^{z_m}}} \)
   \[
   = \text{average estimator/prediction of all buckets}
   \]

\#Distinct elements = Round \( (\beta_m \cdot m \cdot B) \)

\[
\beta_m = \begin{cases} 
0.541 & \text{if } m = 4, \\
0.627 & \text{if } m = 8, \\
0.673 & \text{if } m = 16, \\
0.697 & \text{if } m = 32, \\
0.709 & \text{if } m = 64, \\
\frac{0.723}{1+1.079/m} & \text{if } m \geq 128.
\end{cases}
\]
#distinct=7, 16-bit hashes, 4 buckets, estimated

#distinct = Round (0.541 \cdot 4 \cdot 3.88) = 8

Source: Medjedovic-Tahirovic-Dedovic’s Algorithms and Data Structures for Massive Datasets
HyperLogLogLog\((A[1\ldots n])\)

// Initialize estimators in \(m\) buckets
Create an array \(z_{\text{max}}[0, 1, \ldots, m-1] \leftarrow [0, 0, \ldots, 0]\)

// Compute estimators in \(m\) buckets
for \(i \leftarrow 1\) to \(n\) do
    \(\text{bucket} \leftarrow h(A[i]) \mod m\)  // determines bucket
    \(\text{buckethash} \leftarrow \lceil h(A[i])/m \rceil\)  // determines hash in bucket
    \(z \leftarrow \text{CountTrailingZeros}(\text{buckethash})\)
    if \(z > z_{\text{max}}[\text{bucket}]\) then \(z_{\text{max}}[\text{bucket}] \leftarrow z\)

// Find the harmonic average of estimators in \(m\) buckets
\(B \leftarrow \left( \frac{1}{2^{z_{\text{max}}[0]}} + \frac{1}{2^{z_{\text{max}}[1]}} + \cdots + \frac{1}{2^{z_{\text{max}}[m-1]}} \right)\)

// Estimate the #distinct elements in all buckets
\(\text{distinct} \leftarrow \text{Round}(\beta_m \cdot m \cdot B)\)
return \(\text{distinct}\)

\(\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle\)
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>$\mathcal{O}(n^2)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Sort and count</td>
<td>$\Theta(n \log n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Bit vector</td>
<td>$\Theta(n + U)$</td>
<td>$\Theta(U)$</td>
</tr>
<tr>
<td>Hash table</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Linear counting</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Probabilistic counting</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Stochastic averaging</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>LogLog</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>HyperLogLog</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>
References

- Book: Algorithms and Data Structures for Massive Datasets
- Book: Probabilistic Data Structures and Algorithms for Big Data Applications
Merge $k$ Sorted Arrays

HOME
• Merge $k$ sorted arrays each with size $n$.
• Input: Sorted arrays $A_1, A_2, \ldots, A_k$ each with size $n$
  Output: Sorted array consisting of $kn$ elements
• Input: $A_1 = [3, 5, 8]$, $A_2 = [4, 6, 7]$, $A_3 = [1, 2, 9]$
  Output: $[1, 2, 3, 4, 5, 6, 7, 8, 9]$
Solutions → Naive solution

- Copy elements in all arrays to a single array
- Sort the array using merge sort

**NaiveSolution**($A_1[1...n],...,A_k[1...n]$)

Create a dynamic array $A ← []$

for $i ← 1$ to $k$ do
    for $j ← 1$ to $n$ do
        $A$.Add($A_i[j]$)
    SORT($A$)

return $A$

$\langle$Time, Space$\rangle = \langle \Theta (kn \log (kn)), \Theta (kn) \rangle$
Solutions → Naive merging

- Create an empty dynamic array
- Merge each array with the evolving array one-by-one

\[
\text{\textsc{NaiveMerging}}(A_1[1 \ldots n], \ldots, A_k[1 \ldots n])
\]

Create two dynamic arrays \(B_0 \leftarrow []\) and \(B_1 \leftarrow []\)

\[j \leftarrow 0\]

\[\text{for } i \leftarrow 1 \text{ to } k \text{ do}\]
\[\quad j \leftarrow (j + 1) \mod 2\]
\[\quad B_j \leftarrow \text{MERGE}(B_{(j+1) \mod 2}, A_i)\]

return \(B_j\)

\[\langle \text{Time, Space} \rangle = \langle \Theta \left( k^2 n \right), \Theta \left( kn \right) \rangle\]
$B_0$ and $B_1$ are empty and $j = 0$

Return $B_j$, where $j = 1$
Solutions → Divide-and-conquer

- Merge arrays in groups of two to get \( k/2 \) arrays
- Merge arrays in groups of two to get \( k/4 \) arrays
- Repeat this process until there is only one array
- That single array is the merged sorted array

\[
\text{MERGEKSortedArrays}(A_1[1 \ldots n], \ldots, A_k[1 \ldots n])
\]
\[
\text{return } \text{MERGE-D&C}(A_1[1 \ldots n], \ldots, A_k[1 \ldots n])
\]
\[
\text{MERGE-D&C}(A_{low}, \ldots, A_{high})
\]
\[
\begin{align*}
n &\leftarrow (\text{high} - \text{low} + 1) \\
\text{if } n = 1 \text{ then return } A_{low} \\
\text{mid} &\leftarrow (\text{low} + \text{high})/2 \\
\text{// Split the arrays into left and right sets} \\
A_{left} &\leftarrow \text{MERGE-D&C}([A_{low}, A_{low+1}, \ldots, A_{mid}]) \\
A_{right} &\leftarrow \text{MERGE-D&C}([A_{mid+1}, A_{mid+2}, \ldots, A_{high}]) \\
\text{// Merge the left and right sets} \\
A_{merged} &\leftarrow \text{MERGE}(A_{left}, A_{right}) \\
\text{return } A_{merged}
\end{align*}
\]
\[
\langle \text{Time, Space} \rangle = \langle \Theta (kn \log k), \Theta (kn) \rangle
\]
Solutions \rightarrow \text{Divide-and-conquer}

\begin{itemize}
\item \textbf{A}_1 = [1 2 3]
\item \textbf{A}_2 = [2 3 4]
\item \textbf{A}_3 = [3 4 5]
\item \textbf{A}_4 = [4 5 6]
\item \textbf{A}_5 = [5 6 7]
\item \textbf{A}_6 = [6 7 8]
\item \textbf{A}_7 = [7 8 9]
\item \textbf{A}_8 = [8 9 10]
\end{itemize}
Solutions → Naive $k$-way merge

Core idea.
- In 2-way merge, we take the minimum of elements from the two sorted arrays and add it to the merged array.
- In $k$-way merge, we take the minimum of elements from all the $k$ sorted arrays and add it to the merged array.
- We compute the minimum of $k$ elements using a naive approach of scanning all these $k$ elements and taking the minimum.

Implementation details.
- Maintain a $pointer[1 \ldots k]$ array where $pointer[i]$ points to the next element in $A_i$ to be compared for finding the minimum element.
- We take the minimum element of $\{ A_1[pointer[1]], A_2[pointer[2]], \ldots, A_k[pointer[k]] \}$. If all elements of an array are exhausted, i.e., $pointer[i] = n + 1$ then we do not consider that array for computing the minimum element.
Naive $k$-way merge

NaiveKWayMerge($A_1[1...n], ..., A_k[1...n]$)

$A_{merged} \leftarrow []$

pointer[1...k] $\leftarrow [1...1]$  // initialize a pointer for each array

// Get all the elements of $A_{merged}$ in the sorted order

while true do

    // Find the minimum element among the current pointers of the $k$ arrays
    minval $\leftarrow \infty$; minindex $\leftarrow -1$

    for $i \leftarrow 1$ to $k$ do

        if pointer[$i$] $\leq n$ and $A_i[pointer[i]] < minval$ then

            minval $\leftarrow A_i[pointer[i]]$

            minindex $\leftarrow i$

    // If no minimum element is found, we are done
    if minindex $= -1$ then break

    $A_{merged}$.Append(minval)

    // Move the pointer of the array from which the minimum element was taken
    pointer[minindex] $\leftarrow pointer[minindex] + 1$

return $A_{merged}$

$\langle\text{Time, Space}\rangle = \langle\Theta(k^2n)\rangle, \Theta(kn)\rangle$
## Solutions → Naive $k$-way merge

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_{merged}$</th>
<th>pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_{merged}$</th>
<th>pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_{merged}$</th>
<th>pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_{merged}$</th>
<th>pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_{merged}$</th>
<th>pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_{merged}$</th>
<th>pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_{merged}$</th>
<th>pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Solutions $\to$ Naive $k$-way merge (continued)

<table>
<thead>
<tr>
<th></th>
<th>$A_{merged}$</th>
<th>$pointer$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>1 2 3</td>
<td>4</td>
</tr>
<tr>
<td>$A_2$</td>
<td>2 3 4</td>
<td>3</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3 4 5</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$A_{merged}$</th>
<th>$pointer$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>1 2 3</td>
<td>4</td>
</tr>
<tr>
<td>$A_2$</td>
<td>2 3 4</td>
<td>3</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3 4 5</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$A_{merged}$</th>
<th>$pointer$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>1 2 3</td>
<td>4</td>
</tr>
<tr>
<td>$A_2$</td>
<td>2 3 4</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3 4 5</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$A_{merged}$</th>
<th>$pointer$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>1 2 3</td>
<td>4</td>
</tr>
<tr>
<td>$A_2$</td>
<td>2 3 4</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3 4 5</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$A_{merged}$</th>
<th>$pointer$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>1 2 3</td>
<td>4</td>
</tr>
<tr>
<td>$A_2$</td>
<td>2 3 4</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3 4 5</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$A_{merged}$</th>
<th>$pointer$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>1 2 3</td>
<td>4</td>
</tr>
<tr>
<td>$A_2$</td>
<td>2 3 4</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3 4 5</td>
<td>4</td>
</tr>
</tbody>
</table>
Core idea.

- In the naive $k$-way merge algorithm, we compute the minimum of $k$ elements using a naive approach of scanning all these $k$ elements and taking the minimum. This takes exactly $k$ operations per element, even if some arrays are fully processed.
- In the improved $k$-way merge algorithm, we compute the minimum of $k$ elements in a hash map. This takes at most $k$ operations per element. If arrays are fully processed, then there will be no comparisons for their elements.

Implementation details.

- Maintain a pointer hash map where $\text{pointer}[\text{key}]$ points to the next element in $A_{\text{key}}$ to be compared for finding minimum.
- We take the minimum element of $A_{\text{key}}[\text{pointer}[\text{key}]]$ for all keys in the hash map. If all elements of an array are exhausted, i.e., $\text{pointer}[\text{key}] = n + 1$ then that array number will not be in the hash map.
Improved $k$-way merge

ImprovekwayMerge($A_1[1...n], ..., A_k[1...n]$)

$A_{merged} \leftarrow []$; Create a hash map $pointer$ where the key is the array number and the value is the index inside that array

for $i \leftarrow 1$ to $k$ do $pointer.Add(\langle i, 1 \rangle)$

// Get all the elements of $A_{merged}$ in the sorted order

while true do

// Find the minimum element among the current pointers of the keys (i.e., array numbers) in the hash map

$minval \leftarrow \infty$; $minindex \leftarrow -1$

foreach $key$ in $pointer.Keys()$ do

if $A_{key}[pointer[key]] < minval$ then

$minval \leftarrow A_{key}[pointer[key]]$

$minindex \leftarrow key$

// If no minimum element is found, we are done

if $minindex = -1$ then break

$A_{merged}.Append(minval)$

// Move the pointer of the array from which the minimum element was taken

$pointer[minindex] \leftarrow pointer[minindex] + 1$

// If the array is fully processed, remove its entry from the hash map

if $pointer[minindex] > n$ then $pointer.Remove(minindex)$

return $A_{merged}$

$\langle Time, Space \rangle = \langle \mathcal{O}(k^2n), \Theta(kn) \rangle$
Solutions → Best $k$-way merge

- We create a min-heap of size $k$ and add the first elements of all arrays. When we poll, we check if the element on the right is available, if so, then add it to the heap. Once the heap is empty, $A_{merged}$ is the answer.

```
BESTkWAYMERGE($A_1[1 \ldots n], \ldots, A_k[1 \ldots n]$)

$A_{merged} \leftarrow []$
Create a min-heap $H$ and initialize with the first element of each array
for $i \leftarrow 1$ to $k$ do
| $H$.$Add((A_i[1], i, 1))$ // (element, arrayindex, elementindex)
while min-heap $H$ is not empty do
  // Extract the minimum element from the min-heap
  ($minelement, arrayindex, elementindex$) $\leftarrow H$.RemoveMin()
  $A_{merged}$.$Append(minelement)$
  // Move to the next element in the array that contributed the min element
  elementindex $\leftarrow elementindex + 1$
  // If there are more elements in the current array, insert the next element into the heap
  if $elementindex \leq n$ then
    | $H$.$Add((A_{arrayindex}[elementindex], arrayindex, elementindex))$
return $A_{merged}$
```

$\langle \text{Time, Space} \rangle = \langle \Theta (nk \log k) , \Theta (kn) \rangle$
Solutions → Best $k$-way merge

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**min-heap $H$**

$A_{merged}$

$\phi$

$1$

$A_1$

$A_2$

$A_3$

$1$

$1$

$2$

$1$

$2$

$2$

$A_1$

$A_2$

$A_3$

$1$

$1$

$2$

$3$

$3$

$A_1$

$A_2$

$A_3$

$3$

$3$

$3$

$A_1$

$A_2$

$A_3$

$3$

$3$

$3$

$A_1$

$A_2$

$A_3$

$3$

$3$

$3$

$A_1$

$A_2$

$A_3$

$3$

$3$

$3$
Solutions → **Best $k$-way merge (continued)**

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**min-heap $H$**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{merged}$</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**min-heap $H$**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{merged}$</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**min-heap $H$**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{merged}$</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**min-heap $H$**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{merged}$</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$A_3$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**min-heap $H$**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{merged}$</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Solutions $\rightarrow$ External-memory $k$-way merge

Source: Medjedovic-Tahirovic-Dedovic’s Algorithms and Data Structures for Massive Datasets
Solutions → External-memory $k$-way merge

$k$WAYMERGE($F_1, F_2, \ldots, F_k, M, B$)

For each file $F_i$, create an input buffer $[i]$ of size $B$ elements in RAM
Initialize each input buffer with the first data block from its input file
Create a min-heap $H$ to keep track of the minimum from each input buffer, and an empty output buffer to store the result
Initialize min-heap $H$ with the first element of every input buffer
for $i \leftarrow 1$ to $k$ do
\begin{itemize}
  \item element $\leftarrow$ ReadNextElementFromInputBuffer(inputbuffer$[i]$)
  \item if element is not none then $H$.Add(element, $i$)
\end{itemize}
while min-heap $H$ is not empty do
\begin{itemize}
  \item // Remove from min-heap $H$ the min and input buffer index; and write to output buffer
  \item minelement, bufferindex $\leftarrow$ H.RemoveMin()
  \item WriteElementToOutputBuffer(minelement, outputbuffer)
  \item if output buffer is full then
  \item \hspace{1em}WriteOutputBufferToDisk(outputbuffer, merged file $F_{merged}$)
  \item \hspace{1em}Clear contents of outputbuffer
  \item // Add to min-heap $H$ the next element from input buffer having index bufferindex
  \item element $\leftarrow$ ReadNextElementFromInputBuffer(inputbuffer$[bufferindex]$)
  \item if element is none then
  \item \hspace{1em}inputbuffer$[bufferindex]$ $\leftarrow$ ReadNextDataBlockFromDisk($F_{bufferindex}$)
  \item \hspace{1em}element $\leftarrow$ ReadNextElementFromInputBuffer(inputbuffer$[bufferindex]$)
  \item $H$.Add(element, bufferindex)
\end{itemize}
return $F_{merged}$

$\langle\text{Time, Space}\rangle = \left\langle \Theta \left( \frac{kn}{B} \right) \text{ I/Os}, \Theta (kn) \right\rangle$
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Internal-memory algorithms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Naive solution</td>
<td>$\Theta (kn \log (kn))$</td>
<td>$\Theta (kn)$</td>
</tr>
<tr>
<td>Naive merging</td>
<td>$\Theta (k^2n)$</td>
<td>$\Theta (kn)$</td>
</tr>
<tr>
<td>Divide-and-conquer</td>
<td>$\Theta (kn \log k)$</td>
<td>$\Theta (kn)$</td>
</tr>
<tr>
<td>Naive $k$-way merge</td>
<td>$\Theta (k^2n)$</td>
<td>$\Theta (kn)$</td>
</tr>
<tr>
<td>Improved $k$-way merge</td>
<td>$\Theta (k^2n)$</td>
<td>$\Theta (kn)$</td>
</tr>
<tr>
<td>Best $k$-way merge</td>
<td>$\Theta (nk \log k)$</td>
<td>$\Theta (kn)$</td>
</tr>
<tr>
<td><strong>External-memory algorithms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External-memory $k$-way merge</td>
<td>$\Theta \left(\frac{kn}{B}\right)$ I/Os</td>
<td>$\Theta (kn)$</td>
</tr>
</tbody>
</table>
**Merge Sort (Recursive)**

---

**MergeSort** *(unsortedfile, M, B)*

// Step 1: Divide ..........................................................
Divide *unsortedfile* into $\lceil n/M \rceil$ chunks, each of size at most *M*

foreach unsorted chunk do
  Load the unsorted chunk into RAM
  Sort the chunk using 2-way merge sort
  Write back the sorted chunk to the hard disk
Create *sortedchunks* to contain pointers to all sorted chunks

// Step 2: Conquer and combine (*k*-way merge), where $k = M/B$ ...........

*sortedfile* ← **RecursiveMerge**(*sortedchunks*, *k*)

return *sortedfile*

---

**RecursiveMerge** *(sortedchunks*, *k*)

if there is only one chunk in *sortedchunks* then
  return the only sorted chunk

Create *newsortedchunks* ← [] to contain pointers to merged chunks

while *sortedchunks* has more than one chunk do
  Divide all sorted chunks into groups of *k*, except possibly the last group
  foreach group of at most *k* sorted chunks do
    Merge the *k* chunks into a single sorted chunk using *k*-way merge
    Add this merged chunk to *newsortedchunks*
  return **RecursiveMerge**(*newsortedchunks*, *k*)

---

$$\langle \text{Time, Space} \rangle = \Theta \left( \frac{n}{B} \log \frac{M}{B} \frac{n}{B} \right) \ I/Os, \Theta \left( n \right)$$
Solutions → Merge sort (Non-recursive)

\[
\text{MergeSortNonRecursive}(\text{unsortedfile}, M, B)
\]

// Step 1: Divide ..................................................
Divide \(\text{unsortedfile}\) into \(\lceil n/M \rceil\) chunks, each of size at most \(M\)

\[
\text{foreach \ unsorted\ chunk \ do}
\]
\[
\text{Load the unsorted chunk into RAM}
\]
\[
\text{Sort the chunk using 2-way merge sort}
\]
\[
\text{Write back the sorted chunk to hard disk}
\]
Create \(\text{sortedchunks}\) to contain pointers to all sorted chunks

// Step 2: Conquer \((k\text{-way merge}), \text{ where } k = \frac{M}{B}\) .................
\[
\text{while \(\text{sortedchunks}\) has more than one sorted file \ do}
\]
\[
\text{Create \(\text{newsortedchunks}\) to contain pointers to merged chunks}
\]
\[
\text{Divide all sorted chunks in groups of } k, \text{ except possibly the last group}
\]
\[
\text{foreach \ group \ of \ } k \ \text{sorted \ chunks \ do}
\]
\[
\text{Merge the } k \text{ chunks into a single sorted chunk using } k\text{-way merge}
\]
\[
\text{Append this merged chunk to \(\text{newsortedchunks}\)}
\]
\[
\text{sortedchunks} \leftarrow \text{newsortedchunks}
\]
Let the only file in \(\text{sortedchunks}\) be called \(\text{sortedfile}\)

\[
\text{return } \text{sortedfile}
\]

\[
\langle \text{Time, Space} \rangle = \left\langle \Theta \left( \frac{n}{B} \log \frac{M}{B} \frac{n}{B} \right) \text{ I/Os}, \Theta (n) \right\rangle
\]
Quantum Algorithms
What is Hilbert space?

Definition

- The **inner/dot/scalar product** $\vec{v} \cdot \vec{w}$ of two vectors $\vec{v}$ and $\vec{w}$ is a mathematical operation between two vectors of the same dimension that returns a scalar number.

Suppose $\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$ and $\vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$. Then,

$$\vec{v} \cdot \vec{w} = v_1w_1 + v_2w_2 + \cdots + v_nw_n.$$

- **Hilbert space** is a complex vector Euclidean space with well-defined inner product.
What is superposition?

- $|0\rangle$ (Ground state)
- $|1\rangle$ (1st excited state)
- $|2\rangle$ (2nd excited state)
- $|k-1\rangle$ ($(k-1)$th excited state)

Nucleus

**Energy of an electron in an atom**

- This is a $k$-level quantum mechanical system
- After measuring, electron is in exactly one of the states.
- Before measuring, electron is in all $k$ quantum states.
Superposition is when a quantum particle is in multiple states simultaneously.
What is superposition?

<table>
<thead>
<tr>
<th>Energy of an electron in an atom</th>
</tr>
</thead>
<tbody>
<tr>
<td>After measuring, the electron can be in any one of (</td>
</tr>
</tbody>
</table>
| \[
|0\rangle = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad |k - 1\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}
| are the **computational basis** and they represent the orthonormal basis of a \( k \)-dimensional vector space |
What is superposition?

Energy of an electron in an atom

- **Before measuring**, the electron is in a superposition of all $k$ quantum energy states i.e.,

$$|\psi\rangle = c_0|0\rangle + c_1|1\rangle + \cdots + c_{k-1}|k-1\rangle$$

$$\therefore |\psi\rangle = c_0 \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + c_1 \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} + \cdots + c_{k-1} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{k-1} \end{bmatrix},$$

where $c_i$'s are complex numbers and $|\psi\rangle$ is a unit vector, i.e., $|c_0|^2 + |c_1|^2 + \cdots + |c_{k-1}|^2 = 1$. 


What is a qubit?

A qubit represents the superpositioned state of a 2-state quantum system.

Example: A qubit can be made from a photon being polarized either horizontally or vertically:

\[ |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \]

\[ |\psi\rangle = a |0\rangle + b |1\rangle = a \begin{bmatrix} 1 \\ 0 \end{bmatrix} + b \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} \]

where \( |a|^2 + |b|^2 = 1 \)

Qubit \( |\psi\rangle \) is invalid if \( |a|^2 + |b|^2 \neq 1 \)
A qubit when measured collapses to one of the two basis states

Suppose $|\psi\rangle = a|0\rangle + b|1\rangle = a \begin{bmatrix} 1 \\ 0 \end{bmatrix} + b \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$

Probability of measuring qubit $|\psi\rangle$ as $|0\rangle$ is $|a|^2$

Probability of measuring qubit $|\psi\rangle$ as $|1\rangle$ is $|b|^2$

Observe that the sum of all collapsing probabilities must be 1
Comparison between classical and quantum bit

<table>
<thead>
<tr>
<th>Feature</th>
<th>Bit</th>
<th>Qubit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation</td>
<td>Transistor</td>
<td>Quantum system</td>
</tr>
<tr>
<td>Exclusive states</td>
<td>0 and 1</td>
<td></td>
</tr>
<tr>
<td>State after measuring</td>
<td>0 or 1</td>
<td></td>
</tr>
<tr>
<td>State before measuring</td>
<td>0 or 1</td>
<td>Superposition of</td>
</tr>
<tr>
<td>Representation</td>
<td>bit ∈ {0, 1}</td>
<td>qubit = a</td>
</tr>
</tbody>
</table>
Schrödinger’s cat

Schrödinger’s cat

Problem
- Suppose the probabilities of cat being alive and the cat being dead are the same. Then, what are the values of $a$ and $b$?
Suppose $|\psi\rangle = (a + ib)|0\rangle + (c + id)|1\rangle$

There are 4 variables! However, $\sqrt{a^2 + b^2} + \sqrt{c^2 + d^2} = 1$

So, we can say there are only 3 independent variables

This implies we can visualize this state on a 3-D unit sphere

What should be the three basis vectors or axes?
How to visualize a qubit? Block sphere

Source: https://www.quantum-inspire.com/kbase/bloch-sphere/
How to visualize a qubit? Block sphere

Source: https://logosconcarne.com/2021/03/15/qm-101-bloch-sphere/

<table>
<thead>
<tr>
<th>Axis</th>
<th>Basis</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z$</td>
<td>$</td>
<td>0\rangle$ and $</td>
</tr>
<tr>
<td>$Y$</td>
<td>$</td>
<td>i\rangle$ and $</td>
</tr>
<tr>
<td>$X$</td>
<td>$</td>
<td>+\rangle$ and $</td>
</tr>
</tbody>
</table>
How does a quantum system evolve?

- Both classical and quantum systems evolve through state transformations.
- Arbitrary transformations of a quantum state are not possible.

Time evolution of a quantum system happens through a series of unitary transformations.

- A unitary transformation simply means multiplying by a unit matrix.
- Multiplying by any unitary matrix $U$ is a valid quantum state transformation.

$$U \cdot |\psi_1\rangle = |\psi_2\rangle$$
What is a unitary matrix?

A matrix $U$ is a unitary matrix if $UU^\dagger = U^\dagger U = I$

A matrix $U$ is a unitary matrix if $U^\dagger = U^{-1}$

where $U^\dagger$ is the conjugate transpose of $U$.

Examples

- **Pauli matrices**
  
  $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$,  
  $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$,  
  $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$,  
  $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

- **Hadamard matrix**

  $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
The conjugate transpose of a matrix $M$ is a matrix $M^\dagger$ obtained by taking the complex conjugate of all elements of $M$ and taking the transpose of the resulting matrix.

Examples

- Suppose $M = \begin{bmatrix} 1 + i & 2 + 2i & 3 + 3i \\ 10 - 10i & 20 - 20i & 30 - 30i \end{bmatrix}$.

  Then, $M^\dagger = \begin{bmatrix} 1 - i & 10 + 10i \\ 2 - 2i & 20 + 20i \\ 3 - 3i & 30 + 30i \end{bmatrix}$. 
**What is a quantum operation?**

**Definition**

- A **quantum operation** transforms a quantum state to another quantum state.
- Any quantum operation can be represented by a **unitary matrix**. Similarly, any unitary matrix represents a possible quantum operation.
- Every quantum operation can be thought as a rotation in the Block sphere

**Examples**

- All unitary matrices
Quantum operations are reversible

Every quantum operation, except measurement, is reversible.

- If $U|\psi_1\rangle = |\psi_2\rangle$, then it is possible to reverse the transformation, i.e., $U^\dagger|\psi_2\rangle = |\psi_1\rangle$
- Suppose you have a sequence of quantum operations $U_1U_2U_3 \cdots U_k|\psi_1\rangle = |\psi_2\rangle$, then it is possible to reverse the transformation by using $U_k^\dagger U_{k-1}^\dagger U_{k-2}^\dagger \cdots U_1|\psi_2\rangle = |\psi_1\rangle$
Single-qubit operations (quantum-algorithm level)

Non-Clifford gate

- \( T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \) (45° rotation around Z axis)

Clifford gates

- \( H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \) (create equal superposition of \(|0\rangle\) and \(|1\rangle\))
- \( S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = T^2 \) (90° rotation around Z axis)
- \( X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = HT^4H \) (NOT; 180° rotation around X axis)
- \( Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = T^2HT^4HT^6 \) (180° rotation around Y axis)
- \( Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = T^4 \) (180° rotation around Z axis)

- Pauli operators = \( \{X, Y, Z\} \)

These operations can be composed to approximate any unitary transformation on a single qubit.
Single-qubit operations (function-description level)

- $R_z(\theta) = e^{-i\theta Z/2} = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$
- $R_x(\theta) = e^{-i\theta X/2} = HR_z(\theta)H = \begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$
- $R_y(\theta) = e^{-i\theta Y/2} = SHR_z(\theta)HS^\dagger = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$

These rotations can be composed to perform any unitary transformation on a single qubit.

For every unitary matrix $U$, there exists $\alpha, \beta, \gamma, \delta$ such that

$$U = e^{i\alpha} R_x(\beta) R_z(\gamma) R_x(\delta)$$
Single-qubit operations

- $|0\rangle \rightarrow X \rightarrow |1\rangle$
- $H \rightarrow |+\rangle \rightarrow Z \rightarrow |-\rangle$
- $|+\rangle \rightarrow H \rightarrow |1\rangle$
- $|1\rangle \rightarrow H \rightarrow |+\rangle$
- $|0\rangle \rightarrow |\rangle$
Single-qubit gates

- Classical Boolean computing consists of circuits of NOT, AND, and, OR gates
- Quantum computing consists of circuits of quantum gates

A quantum gate is a quantum operation
A quantum circuit is a model to visualize operations on qubits

\[
\begin{align*}
    &a|0\rangle + b|1\rangle &\quad &X\quad &b|0\rangle + a|1\rangle \\
    &a|0\rangle + b|1\rangle &\quad &Y\quad &-ib|0\rangle + ia|1\rangle \\
    &a|0\rangle + b|1\rangle &\quad &Z\quad &a|0\rangle - b|1\rangle \\
    &a|0\rangle + b|1\rangle &\quad &H\quad &a|+\rangle - b|-\rangle \\
    &a|0\rangle + b|1\rangle &\quad &S\quad &a|0\rangle + be^{i\pi/2}|1\rangle \\
    &a|0\rangle + b|1\rangle &\quad &T\quad &a|0\rangle + be^{i\pi/4}|1\rangle 
\end{align*}
\]
## Random number generation

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate a truly random bit.</td>
<td>Can be solved in classical computing</td>
</tr>
<tr>
<td></td>
<td>Can be solved in quantum computing</td>
</tr>
</tbody>
</table>
Random number generation

$|0\rangle \xrightarrow{H} \text{[Quantum Gate]}$
### Problem

- Generate 37 truly random bits when your quantum computer has only 5 qubits.

### Solution

- Generate 5 random bits in parallel for 7 times and then generate 2 random bits in parallel.
Multi-qubit operations

- 1-qubit introduces superposition
- >1-qubits introduces interference and entanglement

\[
\text{\#dimensions is directly proportional to } 2\text{\#qubits}
\]

- Increase in 1 qubit doubles the computational power. This exponential speedup is the reason that a quantum computer with 100 qubits can surpass the most powerful supercomputers
Multi-qubit states

- First qubit is in the state $|\psi_1\rangle = \begin{bmatrix} a \\ b \end{bmatrix}$
- Second qubit is in the state $|\psi_2\rangle = \begin{bmatrix} c \\ d \end{bmatrix}$
- Corresponding 2-qubit state is given by the tensor product or Kronecker product

$$|\psi_1\rangle \otimes |\psi_2\rangle = \begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix}$$

- This idea can be generalized to $n$-qubits which gives a normalized vector of size $2^n$
Multi-qubit states

A 2-qubit state is written as

$$|\psi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

where

- $$|00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
- $$|01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$
- $$|10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$
- $$|11\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$
Small vs. Large quantum systems

Transformation always possible

\begin{align*}
\text{$n$ small systems of 2-D} & \\
\text{$n$ one-qubits} & \\
|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle & \\
\end{align*}

Transformation NOT always possible

\begin{align*}
\text{1 large system of $2^n$-D} & \\
\text{1 $n$-qubit} & \\
|\Psi\rangle & \\
\end{align*}
Small vs. Large quantum systems

Transformation always possible

2-D × 2-D

Transformation NOT always possible

4-D
Quantum system transformation: Small $\rightarrow$ Large

**Definition**
- A joint system of $n$ small 2-D quantum systems, each having 2 quantum states can be thought as a large $2^n$-D quantum mechanical system having $2^n$ quantum states.
- The **tensor product** $\otimes$ (or Kronecker product) of $n$ one-qubits can be thought to denote a quantum mechanical system having $2^n$ quantum states.

**Examples**
- 3 qubits can be thought to denote an 8-D quantum system.

E.g.: $|1\rangle \otimes |0\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |101\rangle.$
Quantum system transformation: Small $\rightarrow$ Large

Examples

- Alice has quantum state $|\psi\rangle = a|0\rangle + b|1\rangle$.
- Bob has quantum state $|\phi\rangle = c|0\rangle + d|1\rangle$.

Then, their combined quantum state is

$$|\psi\phi\rangle = |\psi\rangle \otimes |\phi\rangle = (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle)$$

$$= \begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix}.$$
Quantum system transformation: Large $\rightarrow$ Small

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A large $2^n$-D quantum mechanical system having $2^n$ quantum states can be thought as a joint system of $n$ small 2-D quantum systems, each having 2 quantum states.</td>
</tr>
<tr>
<td>• A quantum mechanical system having $2^n$ quantum states can be thought as a <strong>tensor product</strong> $\otimes$ (or Kronecker product) of $n$ one-qubits.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>• An 8-D quantum system can be represented using 3 qubits.</td>
</tr>
</tbody>
</table>

E.g.: $|101\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \otimes |0\rangle \otimes |1\rangle$. |
What is a separable state?

**Definition**

- The two-qubit state $|\Psi\rangle$ is **separable** if
  
  $|\Psi\rangle = |\psi\rangle \otimes |\phi\rangle$
  
  for some one-qubit states $|\psi\rangle$ and $|\phi\rangle$. 
What is a separable state?

The combined state of Alice and Bob is $|\Psi\rangle = \frac{1}{2}(|00\rangle + |01\rangle - |10\rangle - |11\rangle)$. Find the individual states of Alice and Bob.

Let $|\Psi\rangle = |\psi\rangle \otimes |\phi\rangle = \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix} \otimes \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} \psi_1 \phi_1 \\ \psi_1 \phi_2 \\ \psi_2 \phi_1 \\ \psi_2 \phi_2 \end{bmatrix}$.

Solving the system of equations, we find that Alice’s state $|\psi\rangle = |\!\!-\rangle$ and Bob’s state $|\phi\rangle = |\!\!+\rangle$. 

**Examples**
What is an entangled state?

Definition

- The two-qubit state $|\Psi\rangle$ is **entangled** if

  \[
  |\Psi\rangle \neq |\psi\rangle \otimes |\phi\rangle
  \]

  for any one-qubit states $|\psi\rangle$ and $|\phi\rangle$.

- A two-qubit state is called an **entangled state** if it cannot be written as the tensor product of single-qubit states.

- A two-qubit gate is called an **entangled gate** if it cannot be written as the tensor product of single-qubit gates.
What is an entangled state?

Examples

- The combined state of Alice and Bob is \( |\Psi\rangle = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) \).

  Find the individual states of Alice and Bob.

  Let \( |\Psi\rangle = |\psi\rangle \otimes |\phi\rangle = \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix} \otimes \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \psi_1 \phi_1 \\ \psi_1 \phi_2 \\ \psi_2 \phi_1 \\ \psi_2 \phi_2 \end{bmatrix} \).

  The system of equations is not solvable.

  Hence, the state \( |\Psi\rangle \) is entangled. This implies that it is impossible to obtain the individual states of Alice and Bob.
What are Bell states?

Definition

- The following two-qubit states are known as the **Bell states**. They represent an orthonormal, entangled basis for two qubits.

<table>
<thead>
<tr>
<th>Bell states</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
</tr>
<tr>
<td>$</td>
</tr>
<tr>
<td>$</td>
</tr>
<tr>
<td>$</td>
</tr>
</tbody>
</table>
2-qubit gates

Single-qubit gate.

\[ a|0\rangle + b|1\rangle \quad \underbrace{U} \quad a'|0\rangle + b'|1\rangle \]

\[
\begin{bmatrix} a' \\ b' \end{bmatrix} = U \begin{bmatrix} a \\ b \end{bmatrix}
\]

Two-qubit gate.

\[ a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle \quad \underbrace{U} \quad a'|00\rangle + b'|01\rangle + c'|10\rangle + d'|11\rangle \]

\[
\begin{bmatrix} a' \\ b' \\ c' \\ d' \end{bmatrix} = U \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}
\]
2-qubit gates

Here are some 2-qubit gates

- \( H_2 = H \otimes H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \)

- Controlled NOT: \( CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \)

- \( SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \)
An election was held in a democratic nation to elect their next leader. The citizens of the nation voted for their favorite candidates. It is now time to find whether someone won the election. Winning the election means getting a majority of votes. Given a set of elements, an element is a majority in that set if that element occurs greater than 50% of the number of elements in that set. If there is no majority in an election, there will be a re-election and the process repeats until there is a majority. So, how do you find whether someone won an election?
Problem


- Input: Array of natural numbers
  Output: Majority if it exists, -1 if there is no majority
- Input: \([3, 3, 4, 2, 4, 4, 2, 4, 4]\)
  Output: 4
- Input: \([3, 3, 4, 2, 4, 4, 2, 4]\)
  Output: -1
Solutions → Brute force

1. Count occurrences of each element
2. Find the majority

**MAJORITY-BRUTEFORCE**($A[1...n]$)

**Input:** Array $A[1...n]$ of natural numbers.
**Output:** Majority element if it exists and $-1$ otherwise.

```plaintext
for i ← 1 to ⌊n/2⌋ do
    count ← COUNTOCCURRENCES($A[i...n], A[i]$)
    if count > ⌊n/2⌋ then
        return $A[i]$
return $-1$
```

**COUNTOCCURRENCES**($A[ℓ...h], k$)

```plaintext
count ← 0
for i ← ℓ to h do
    if $A[i] = k$ then
        count ← count + 1
return count
```

$\langle \text{Time, Space} \rangle = \langle \Theta(n^2), \Theta(1) \rangle$
Solutions → Sorting

1. Sort the array
2. Count occurrences of each element
3. Find the majority

\[
\text{MAJORITY-SORT}(A[1 \ldots n])
\]

\[
A[1 \ldots n] \leftarrow \text{SORT}(A[1 \ldots n])
\]

\[
i \leftarrow 1
\]

\[
\text{for } j \leftarrow 2 \text{ to } n \text{ do}
\]

\[
\text{if } A[j] \neq A[i] \text{ then}
\]

\[
\text{if } (j - i) > \lfloor n/2 \rfloor \text{ then}
\]

\[
\text{return } A[i]
\]

\[
i \leftarrow j
\]

\[
\text{if } (n - i + 1) > \lfloor n/2 \rfloor \text{ then}
\]

\[
\text{return } A[i]
\]

\[
\text{return } -1
\]

\[
\langle \text{Time, Space} \rangle = \langle \Theta (n \log n) , \Theta (n) \rangle
\]
Solutions → Divide-and-conquer

1. Split the array into two halves
2. \( \ell_{majority} \leftarrow \) majority in the left half
3. \( rmajority \leftarrow \) majority in the right half
4. Check if \( \ell_{majority} \) or \( rmajority \) is the array majority

### Majority-D&C(\(A[1 \ldots n]\))

<table>
<thead>
<tr>
<th>return</th>
<th>D&amp;C(A[1 \ldots n])</th>
</tr>
</thead>
</table>

### D&C(A[low \ldots high])

<table>
<thead>
<tr>
<th>if low = high then return A[low]</th>
</tr>
</thead>
<tbody>
<tr>
<td>size ← (high - low + 1); mid ← [(low + high)/2]</td>
</tr>
<tr>
<td>( \ell_{majority} \leftarrow ) D&amp;C(A[low \ldots mid])</td>
</tr>
<tr>
<td>( rmajority \leftarrow ) D&amp;C(A[(mid + 1) \ldots high])</td>
</tr>
<tr>
<td>( \ellcount \leftarrow ) CountOccurrences(A[low \ldots high], ( \ell_{majority} ))</td>
</tr>
<tr>
<td>( rcount \leftarrow ) CountOccurrences(A[low \ldots high], ( rmajority ))</td>
</tr>
<tr>
<td>if ( \ellcount &gt; \lfloor size/2 \rfloor ) then return ( \ell_{majority} )</td>
</tr>
<tr>
<td>if ( rcount &gt; \lfloor size/2 \rfloor ) then return ( rmajority )</td>
</tr>
<tr>
<td>return (-1)</td>
</tr>
</tbody>
</table>

\(\langle Time, Space\rangle = \langle \Theta \left(n \log n\right), \Theta \left(\log n\right)\rangle\)
1. Store $\langle$unique element, frequency$\rangle$ pairs in hash map
2. Find majority

**Majority-Hashing($A[1\ldots n]$)**

Create hash map $H$ to insert (element, frequency) pairs

```python
for i ← 1 to n do
    if $H$.ContainsKey($A[i]$) then
        $H$.Add($\langle A[i] , H$.GetValue($A[i]) + 1\rangle$
    else
        $H$.Add($\langle A[i] , 1\rangle$
    if $H$.GetValue($A[i]$) $> \lfloor n/2 \rfloor$ then
        return $A[i]$
return $-1$
```

$\langle$Time, Space$\rangle$ = $\langle \Theta (n), \Theta (n) \rangle$
Solutions → Median

1. Find the median element
2. Check if the median is the majority

**Majority-Median**$(A[1\ldots n])$

\[
\begin{align*}
\text{median} & \leftarrow \text{Selection}(A[1\ldots n], \lfloor n/2 \rfloor) \\
\text{count} & \leftarrow \text{CountOccurrences}(A[1\ldots n], \text{median}) \\
\text{if } \text{count} > \lfloor n/2 \rfloor \text{ then} \\
& \quad \text{return median} \\
\text{return } -1
\end{align*}
\]

\[
\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle
\]
1. Select a random element and check if it is majority
2. Repeat Step 1 for at most $\left\lfloor \log_2 \frac{1}{\epsilon} \right\rfloor$ number of times
3. Return majority

### MAJORITY-PROBABILISTIC($A[1 \ldots n]$)

```plaintext
def MAJORITY-PROBABILISTIC($A[1 \ldots n]$):
    for $i \leftarrow 1$ to $\left\lfloor \log_2 \frac{1}{\epsilon} \right\rfloor$ do
        random $\leftarrow$ RANDOM($A[1 \ldots n]$)
        count $\leftarrow$ COUNTOCCURRENCES($A[1 \ldots n]$, random)
        if count $>$ $\lfloor n/2 \rfloor$ then
            return random
    return -1
```

$\langle$Time, Space$\rangle = \Theta \left( n \log \frac{1}{\epsilon} \right), \Theta (1)$
Remove two unequal elements from $A$ to get $B$

If $A$ has a majority $M$, then $B$ also has majority $M$

If $B$ has a majority $M$, then $A$ need not have majority $M$
1. If a pair is different, then discard
   If a pair is same, then keep one copy
2. Repeat step 1 until only one element if left
3. If array has majority, then final element is majority
   If array has no majority, then final element has no meaning
Solutions → BoyerMoore-Multipass

**Majority-BoyerMoore-Multipass**($A[1\ldots n]$)

**Majority-Multipass**($A[1\ldots n], -1$)

**Majority-Multipass**($A[1\ldots n], tiebreaker$)

Create a dynamic array $B \leftarrow []$

for $i \leftarrow 1$ to $n - 1$ increment 2 do

| if $A[i] = A[i + 1]$ then
| $B$.Add($A[i]$)

if $n \mod 2 = 1$ then $tiebreaker \leftarrow A[n]$

if $B$ is empty then $C \leftarrow tiebreaker$

else $C \leftarrow$ Majority-Multipass($B, tiebreaker$)

if $C = -1$ then return $-1$

$count \leftarrow$ CountOccurrences($A[1\ldots n], C$)

if $count > \lfloor n/2 \rfloor$ or ($count = \lfloor n/2 \rfloor$ and $C = tiebreaker$) then

| return $C$

return $-1$

\[
\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle
\]
Array has a majority

tiebreaker = −1
candidate = c

tiebreaker = c
candidate = c

tiebreaker = c
candidate = c
1. Create a stack. Scan the elements one at a time.
2. If stack is empty or if element considered is the same as stack top, then push element. Else, pop an element from stack.
3. If stack is non-empty, check if stack top is the majority element. Else, return $-1$. 

Solutions → BoyerMoore-Twopass

Majority-BoyerMoore-Twopass(A[1 ... n])

// Stage 1. Eliminate all except one candidate C
Create a stack S
for i ← 1 to n do
    if S is empty then S.Push(A[i])
    else
        top ← S.Pop()
        if A[i] = top then S.Push(A[i])
        else S.Pop()

// Stage 2. Check whether C is the majority
if S is empty then return -1
C ← S.Pop()
count ← COUNTOccurrences(A[1 ... n], C)
if count > ⌊n/2⌋ then return C
return -1

⟨Time, Space⟩ = ⟨Θ (n) , O (n)⟩
Solutions $\rightarrow$ BoyerMoore-Twopass

<table>
<thead>
<tr>
<th>$i$</th>
<th>$A[i]$</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a$</td>
<td>$[a]$</td>
</tr>
<tr>
<td>2</td>
<td>$a$</td>
<td>$[a, a]$</td>
</tr>
<tr>
<td>3</td>
<td>$a$</td>
<td>$[a, a, a]$</td>
</tr>
<tr>
<td>4</td>
<td>$b$</td>
<td>$[a, a]$</td>
</tr>
<tr>
<td>5</td>
<td>$b$</td>
<td>$[a]$</td>
</tr>
<tr>
<td>6</td>
<td>$b$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>7</td>
<td>$b$</td>
<td>$[b]$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a$</td>
<td>$[a]$</td>
<td>1</td>
<td>$a$</td>
<td>$[a]$</td>
<td>1</td>
<td>$a$</td>
<td>$[a]$</td>
</tr>
<tr>
<td>2</td>
<td>$b$</td>
<td>$\phi$</td>
<td>2</td>
<td>$b$</td>
<td>$\phi$</td>
<td>2</td>
<td>$b$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>3</td>
<td>$a$</td>
<td>$[a]$</td>
<td>3</td>
<td>$a$</td>
<td>$[a]$</td>
<td>3</td>
<td>$a$</td>
<td>$[a]$</td>
</tr>
<tr>
<td>4</td>
<td>$b$</td>
<td>$\phi$</td>
<td>4</td>
<td>$b$</td>
<td>$\phi$</td>
<td>4</td>
<td>$b$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>5</td>
<td>$a$</td>
<td>$[a]$</td>
<td>5</td>
<td>$a$</td>
<td>$[a]$</td>
<td>5</td>
<td>$a$</td>
<td>$[a]$</td>
</tr>
<tr>
<td>6</td>
<td>$b$</td>
<td>$\phi$</td>
<td>6</td>
<td>$b$</td>
<td>$\phi$</td>
<td>6</td>
<td>$b$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>7</td>
<td>$c$</td>
<td>$[c]$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Let $C$ be majority candidate; $m$ be \#unpaired occurrences of $C$
2. In iteration 1, we set $C \leftarrow$ 1st element and $m \leftarrow 1$
3. In iteration $i \in [2, n]$, if $m$ is zero, then set $C \leftarrow i$th element and $m \leftarrow 1$. Otherwise, compare if $i$th element is same as $C$. If same, then increment $m$, else, decrement $m$.
4. If $m$ is positive, then check if $C$ is majority
\[
\text{ Majority-BoyerMoore-Twopass-Inplace}(A[1 \ldots n])
\]

\[
C \leftarrow A[1]; m \leftarrow 1
\]

// Stage 1. Eliminate all except one candidate \(c\)

\[
\text{ for } i \leftarrow 2 \text{ to } n \text{ do }
\]

\[
\begin{align*}
\text{ if } m = 0 \text{ then } \\
\quad \{ C \leftarrow A[i]; m \leftarrow 1 \} \\
\text{ else }
\end{align*}
\]

\[
\begin{align*}
\text{ if } C = A[i] \text{ then } m \leftarrow m + 1 \\
\text{ else } m \leftarrow m - 1
\end{align*}
\]

// Stage 2. Check whether \(c\) is the majority

\[
\text{ if } m \neq 0 \text{ then }
\]

\[
\begin{align*}
\text{ count } & \leftarrow \text{ CountOccurrences}(A[1 \ldots n], C) \\
\text{ if count } & > \lfloor n/2 \rfloor \text{ then return } C
\end{align*}
\]

return \(-1\)

\[
\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle
\]
1. Let $C$ be majority candidate; $m$ be \#unpaired occurrences of $C$
2. In iteration 1, we set $C \leftarrow$ 1st element and $m \leftarrow 1$
3. In iteration $i \in [2, n]$, if $m$ is zero, then set $C \leftarrow i$th element and $m \leftarrow 1$. Otherwise, compare if $i$th element is same as $C$. If same, then increment $m$, else, decrement $m$.
4. If $m$ is positive, then check if $C$ is majority

<table>
<thead>
<tr>
<th>$i$</th>
<th>$A[i]$</th>
<th>$C$</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a$</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$a$</td>
<td>$a$</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>$a$</td>
<td>$a$</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>$b$</td>
<td>$b$</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>$b$</td>
<td>$b$</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>$b$</td>
<td>$b$</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>$b$</td>
<td>$b$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>$A[i]$</th>
<th>$C$</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a$</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$b$</td>
<td>$a$</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$a$</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$b$</td>
<td>$a$</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>$a$</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>$b$</td>
<td>$a$</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>$c$</td>
<td>$c$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>$A[i]$</th>
<th>$C$</th>
<th>$m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a$</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$b$</td>
<td>$a$</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$a$</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$b$</td>
<td>$a$</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>$a$</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>$b$</td>
<td>$a$</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>$b$</td>
<td>$a$</td>
<td>0</td>
</tr>
<tr>
<td>MAJORITY-FISCHER-SALZBERG(A[1...n])</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>// Stage 1. Find the majority candidate C</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create two stacks S₁ and S₂</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>for</strong> i ← 1 to n <strong>do</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>if S₁ is empty <strong>or</strong> S₁.Top() ≠ A[i] <strong>then</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>S₁.Push(A[i])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>else</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>if S₂ is not empty <strong>then</strong> S₁.Push(S₂.Pop())</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>else S₂.Push(A[i])</td>
<td></td>
</tr>
<tr>
<td>C ← S₁.Top()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>// Stage 2. Confirm if the candidate is the majority</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>while</strong> S₁ is not empty <strong>do</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>item ← S₁.Pop()</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>if item = C <strong>then</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>if S₁ is empty <strong>then</strong> S₂.Push(C)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>else S₁.Pop()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>else</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>if S₂ is empty <strong>then</strong> return −1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>else S₂.Pop()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if S₂ is not empty <strong>then</strong> return C</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>return</strong> −1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

⟨Time, Space⟩ = ⟨Θ (n), Θ (n)⟩
<table>
<thead>
<tr>
<th>$i$</th>
<th>$A[i]$</th>
<th>$S_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a$</td>
<td>$[a]$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>2</td>
<td>$a$</td>
<td>$[a]$</td>
<td>$[a]$</td>
</tr>
<tr>
<td>3</td>
<td>$a$</td>
<td>$[a]$</td>
<td>$[a,a]$</td>
</tr>
<tr>
<td>4</td>
<td>$b$</td>
<td>$[a,b,a]$</td>
<td>$[a]$</td>
</tr>
<tr>
<td>5</td>
<td>$b$</td>
<td>$[a,b,a,b,a]$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>6</td>
<td>$b$</td>
<td>$[a,b,a,b,a,b]$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>7</td>
<td>$b$</td>
<td>$[a,b,a,b,a,b]$</td>
<td>$[b]$</td>
</tr>
<tr>
<td></td>
<td>$[a,b,a,b]$</td>
<td>$[b]$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$[a,b]$</td>
<td>$[b]$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\phi$</td>
<td>$[b]$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$i$</th>
<th>$A[i]$</th>
<th>$S_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a$</td>
<td>$[a]$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>2</td>
<td>$b$</td>
<td>$[a,b]$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>3</td>
<td>$a$</td>
<td>$[a,b,a]$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>4</td>
<td>$b$</td>
<td>$[a,b,a,b]$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>5</td>
<td>$a$</td>
<td>$[a,b,a,b,a]$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>6</td>
<td>$b$</td>
<td>$[a,b,a,b,a,b]$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>7</td>
<td>$c$</td>
<td>$[a,b,a,b,a,b,c]$</td>
<td>$\phi$</td>
</tr>
<tr>
<td></td>
<td>$[a,b,a,b,a]$</td>
<td>$\phi$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$[a,b,a,b]$</td>
<td>$\phi$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\phi$</td>
<td>$\phi$</td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>Time</td>
<td>Extra Space</td>
<td>Comments</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------</td>
<td>-------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Brute force</td>
<td>$\Theta (n^2)$</td>
<td>$\Theta (1)$</td>
<td>–</td>
</tr>
<tr>
<td>Sorting-based</td>
<td>$\Theta (n \log n)$</td>
<td>$\Theta (n)$</td>
<td>Can’t solve.</td>
</tr>
<tr>
<td>Divide-and-conquer</td>
<td>$\Theta (n \log n)$</td>
<td>$\Theta (\log n)$</td>
<td>–</td>
</tr>
<tr>
<td>Probabilistic</td>
<td>$\Theta (n \log \frac{1}{\epsilon})$</td>
<td>$\Theta (1)$</td>
<td>Can’t solve. Success $&gt; 1 - \epsilon$.</td>
</tr>
<tr>
<td>Hashing-based</td>
<td>$\Theta (n)$</td>
<td>$\Theta (n)$</td>
<td>Can’t solve.</td>
</tr>
<tr>
<td>Median-based</td>
<td>$\Theta (n)$</td>
<td>$\Theta (n)$</td>
<td>Can’t solve.</td>
</tr>
<tr>
<td>BoyerMoore multipass</td>
<td>$\Theta (n)$</td>
<td>$\Theta (n)$</td>
<td>–</td>
</tr>
<tr>
<td>BoyerMoore twopass</td>
<td>$\Theta (n)$</td>
<td>$O (n)$</td>
<td>–</td>
</tr>
<tr>
<td>BoyerMoore twopass inplace</td>
<td>$\Theta (n)$</td>
<td>$\Theta (1)$</td>
<td>–</td>
</tr>
<tr>
<td>FischerSalzberg</td>
<td>$\Theta (n)$</td>
<td>$\Theta (n)$</td>
<td>–</td>
</tr>
</tbody>
</table>
References

• Puzzle book
Longest Palindromic Substring
Problem

- Design an algorithm to compute a longest palindromic substring of a string.

  - Input: $s = bababaeabaed$
  - Output: $s[4, 10] = abaeaba$
Core data structures

- $oddp[i] = \text{largest radius of odd-sized palindrome centered at index } i$

  $s[i]$

  $oddp[i]$

- $evenp[i] = \text{largest radius of even-sized palindrome centered at indices } i \text{ and } i + 1$

  $s[i]$

  $evenp[i]$
Core data structures

• Longest odd-length palindrome whose center is at index \( i \) is
  \[ s[(i - oddp[i]) \ldots (i + oddp[i])] \]

• Longest even-length palindrome whose left center is at index \( i \) is
  \[ s[(i - evenp[i] + 1) \ldots (i + evenp[i])] \]
Compute $oddp[1\ldots n]$ and $evenp[1\ldots n]$.

Computing the LPS from $oddp[1\ldots n]$ and $evenp[1\ldots n]$ is easy.
1. Find all substrings
2. Check which of these substrings are palindromes
3. If palindromes, update $oddp$ and $evenp$ arrays accordingly

```
BruteForce(s[1...n])
```

Create arrays $oddp[1...n] \leftarrow [0...0]$ and $evenp[1...n] \leftarrow [0...0]

for $i \leftarrow 1$ to $n$ do
  for $j \leftarrow i$ to $n$ do
    if isPalindrome($s[i...j]$) then
      $length \leftarrow j - i + 1$ // palindrome length
      $radius \leftarrow \left\lfloor \frac{length}{2} \right\rfloor$ // palindrome radius
      $center \leftarrow \left\lfloor \frac{i+j}{2} \right\rfloor$ // palindrome center
      if $length$ is odd then
        $oddp[center] \leftarrow \text{Max}(oddp[center], radius)$
      else
        $evenp[center] \leftarrow \text{Max}(evenp[center], radius)$

return $(oddp, evenp)$

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O} \left( n^3 \right), \Theta \left( n \right) \right\rangle$$
Solutions → Standard algorithm

**STANDARDALGORITHM**(*s[1...n]*)

Create arrays `oddp[1...n] ← [0...0]` and `evenp[1...n] ← [0...0]`

// For each palindrome center \(i\), compute \(oddp[i]\)
for \(i ← 1\) to \(n\) do
  \(\ell ← i - oddp[i] - 1\)
  \(r ← i + oddp[i] + 1\)
  while \(\ell ≥ 1\) and \(r ≤ n\) and \(s[\ell] = s[r]\) do
    \(oddp[i] ← oddp[i] + 1\)
    \(\ell ← \ell - 1\)
    \(r ← r + 1\)

// For each palindrome center \(i\), compute \(evenp[i]\)
for \(i ← 1\) to \(n\) do
  \(\ell ← i - evenp[i]\)
  \(r ← i + evenp[i] + 1\)
  while \(\ell ≥ 1\) and \(r ≤ n\) and \(s[\ell] = s[r]\) do
    \(evenp[i] ← evenp[i] + 1\)
    \(\ell ← \ell - 1\)
    \(r ← r + 1\)

return \((oddp, evenp)\)

\(\langle\text{Time, Space}\rangle = \langle\mathcal{O}\left(n^2\right), \Theta\left(n\right)\rangle\)
BinarySearch($s[1 \ldots n]$)

- $low \leftarrow 1; high \leftarrow n$
- $longest\text{palindrome} \leftarrow$ empty string

**while** $low \leq high$ **do**

- $mid \leftarrow \left\lfloor \frac{high - low}{2} \right\rfloor$
- $palindrome \leftarrow$ FINDPALINDROME($s, mid$)

  // There is no palindrome with size $mid$ as it is too big

  // So check for palindromes with smaller sizes

  **if** palindrome is an empty string **then**
  - $high \leftarrow mid - 1$

  // There is a palindrome with size $mid$, maybe it is too small

  // So check for palindromes with larger sizes

  **else**
  - $longest\text{palindrome} \leftarrow$ palindrome
  - $low \leftarrow mid + 1$

**return** $longest\text{palindrome}$

\[
\langle \text{Time, Space} \rangle = \langle \Theta (n \cdot \log n) \text{ w.h.p, } \Theta (1) \rangle
\]

\[
\langle \text{Time, Space} \rangle = \langle \mathcal{O} (n^2 \log n) , \Theta (1) \rangle
\]
**FindPalindrome**($s[1 \ldots n], m$)

// Step 1: Initialize parameters and variables
$p \leftarrow \text{RandomLargePrime}([1 \ldots nm^2])$

$b \leftarrow \text{Size of ASCII set}$

$h \leftarrow b^{(m-1)} \mod p$, $r \leftarrow \text{Reverse of string } s$

$hash1 \leftarrow \text{HASH}(s[1 \ldots m], b, p)$

$hash2 \leftarrow \text{HASH}(r[n - m + 1 \ldots n], b, p)$

// Step 2: Create a rolling hash for substrings of size $m$
for $i \leftarrow 1$ to $(n - m + 1)$ do

$j \leftarrow n - m - i + 2$ // index of substring in $r$

// Probability of hash collision is less than $1/n$
if $hash1 = hash2$ and $s[i \ldots (i + m - 1)] = r[j \ldots (j + m - 1)]$ then

| return $s[i \ldots (i + m - 1)]$

// Rolling hash: Compute hash value of the next text window using the current text window in $\Theta(1)$ time
if $i \neq (n - m + 1)$ then

| $hash1 \leftarrow \text{ROLLINGHASH-LTOR}(hash1, s[i] \times h, s[i + m], b, p)$

| $hash2 \leftarrow \text{ROLLINGHASH-RTOLE}(hash2, r[j + m - 1], r[i - 1] \times h, b, p)$

return empty string
Solutions → Rabin-Karp + Binary search

\[ \text{RollingHash-LtoR}(hash, subtract, add, b, p) \]

\[
hash \leftarrow ((hash - subtract) \times b + add) \mod p \\
\text{return } hash
\]

\[ \text{RollingHash-RtoL}(hash, subtract, add, b, p) \]

\[
hash \leftarrow ((hash - subtract) \times b^{-1} + add) \mod p \\
\text{return } hash
\]

\[ \text{Time } = \Theta(1) \]

\[ b^{-1} \text{ is modulo multiplicative inverse of } b \]

\[ b^{-1} \text{ always exists if the modulus is w.r.t a prime} \]

\[ b^{-1} \text{ can be computed using extended Euclidean algorithm} \]

More information: Modulo multiplicative inverse
• This algorithm is an optimization over $O(n^2)$ algorithm
• Instead of computing $oddp[i]$ and $evenp[i]$ from scratch at every value of $i$, we reuse the already computed values to reduce computations
Suppose we have $oddp[1 \ldots 8]$; we want to compute $oddp[9]$

- Let $s[L \ldots R]$ = palindromic substring that ends as far to the right as possible so far
- For $i = 9$, $s[L \ldots R] = s[4 \ldots 10] = abaeaba$ with center at 7

There are two scenarios to consider:

- $i \geq R$: Use the standard algorithm logic to compute $oddp[i]$
- $i < R$: Use $oddp[\text{mirror image of } i]$ to compute $oddp[i]$
Suppose we have $oddp[1\ldots 8]$; we want to compute $oddp[9]$.

Let $s[L\ldots R] =$ palindromic substring that ends as far to the right as possible so far.

For $i = 9$, $s[L\ldots R] = s[4\ldots 10] = abaeaba$ with center at 7.

There are two scenarios to consider:

- $i \geq R$: Use the standard algorithm logic to compute $oddp[i]$
- $i < R$: Use $oddp[\text{mirror image of } i]$ to compute $oddp[i]$

When $i < R$, the mirror image of $i$ is $j = (L + R - i)$ with respect to the center of $s[L\ldots R]$. 

---

**Indices and Positions**

<table>
<thead>
<tr>
<th>indices $s[i]$</th>
<th>positions $oddp[i]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>0 1 2 2 1 0 3 0</td>
</tr>
<tr>
<td>b a b a b a e a</td>
<td>L j i R</td>
</tr>
<tr>
<td>b a e a b a</td>
<td></td>
</tr>
</tbody>
</table>

**Supplementary Diagram**

- **Manacher’s algorithm**
  - Compute $oddp[i]$ using the standard algorithm logic when $i \geq R$.
  - Use $oddp[\text{mirror image of } i]$ to compute $oddp[i]$ when $i < R$.
  - The mirror image of $i$ is $j = (L + R - i)$ with respect to the center of $s[L\ldots R]$. 

---

**Supplementary Text**

- **Solutions → Manacher’s algorithm**
- **Indices**
  - $s[i]$
- **Positions**
  - $oddp[i]$
- **Table**
  - | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
  - | b | a | b | a | b | a | e | a | b | a | e | d |
  - | L | j | i | R |
### Case 1. $i \geq R$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s[i]$</td>
<td>b</td>
<td>a</td>
<td>e</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>e</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>e</td>
<td>a</td>
</tr>
<tr>
<td>$j$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$L$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$R$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$i$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$oddp[i]$</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this case, compute $oddp[i]$ using standard algorithm logic.
Case 1. \( i \geq R \)

In this case, compute \( oddp[i] \) using standard algorithm logic

- \( i \geq R \), i.e., \( 9 \geq 8 \)
- Therefore, we compute \( oddp[9] \) as per standard algorithm logic
- We have \( oddp[9] = 3 \). We do not save any computations
Case 2. $i < R$ and largest palindrome at center $j$ is completely inside $s[L \ldots R]$

In this case, set $oddp[i]$ to $oddp[j]$, and extend $oddp[i]$. 

---

**Indices**

- $s[i]$ positions
- $oddp[i]$ positions

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s[i]$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>e</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>e</td>
<td>d</td>
</tr>
<tr>
<td>$oddp[i]$</td>
<td></td>
<td>L</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>i</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Positions</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case 2. $i < R$ and largest palindrome at center $j$ is completely inside $s[L \ldots R]$

<table>
<thead>
<tr>
<th>indices</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s[i]$</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>e</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>e</td>
<td>d</td>
</tr>
<tr>
<td>positions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$oddp[i]$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this case, set $oddp[i]$ to $oddp[j]$, and extend $oddp[i]$

- $i < R$, i.e., $9 < 10$
- $s[j - oddp[j], j + oddp[j]]$ is inside $s[L \ldots R]$, i.e., $s[4 \ldots 6]$ is inside $s[4 \ldots 10]$
- We have $oddp[9] = 2$. We save one computation
Case 3. $i < R$ and largest palindrome at center $j$ is not completely inside $s[L \ldots R]$

In this case, set $oddp[i]$ to $(R - i)$, and extend $oddp[i]$
Case 3. $i < R$ and largest palindrome at center $j$ is not completely inside $s[L \ldots R]$

In this case, set $oddp[i]$ to $(R - i)$, and extend $oddp[i]$

- $i < R$, i.e., $9 < 10$
- $s[j - oddp[j], j + oddp[j]]$ is not inside $s[L \ldots R]$, i.e., $s[1 \ldots 9]$ is not completely inside $s[4 \ldots 10]$
- Therefore, set $oddp[9]$ to $(R - i) = (j - L)$, i.e., $oddp[9] = 1$, and extend $oddp[9]$ as per standard algorithm logic
- We have $oddp[9] = 1$. We save one computation
Solutions → Manacher’s algorithm

\[
\text{MANACHERODD}(s[1 \ldots n])
\]

Create array \(oddp[1 \ldots n] \leftarrow [0 \ldots 0]\)

\(L \leftarrow 1\)

\(R \leftarrow -1\)

\[\text{for } i \leftarrow 1 \text{ to } n \text{ do}\]

\[\text{if } i < R \text{ then}\]

\[\text{oddp}[i] \leftarrow \text{Min}(oddp[L + R - i], R - i)\]

\(\ell \leftarrow i - oddp[i] - 1\)

\(r \leftarrow i + oddp[i] + 1\)

\[\text{while } \ell \geq 1 \text{ and } r \leq n \text{ and } s[\ell] = s[r] \text{ do}\]

\[\text{oddp}[i] \leftarrow oddp[i] + 1\]

\(\ell \leftarrow \ell - 1\)

\(r \leftarrow r + 1\)

\[\text{if } i + oddp[i] > R \text{ then}\]

\(L \leftarrow i - oddp[i]\)

\(R \leftarrow i + oddp[i]\)

\[\text{return } oddp\]

\[\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle\]
Solutions → Manacher’s algorithm

\[
\text{ManacherEven}(s[1 \ldots n])
\]

Create array \(evenp[1 \ldots n] \leftarrow [0 \ldots 0]\)
\[L \leftarrow 1\]
\[R \leftarrow -1\]

for \(i \leftarrow 1\) to \(n\) do

\[\text{if } i < R \text{ then}\]
\[evenp[i] \leftarrow \text{Min}(evenp[L + R - i], R - i)\]
\[\ell \leftarrow i - evenp[i]\]
\[r \leftarrow i + evenp[i] + 1\]

while \(\ell \geq 1\) and \(r \leq n\) and \(s[\ell] = s[r]\) do

\[evenp[i] \leftarrow evenp[i] + 1\]
\[\ell \leftarrow \ell - 1\]
\[r \leftarrow r + 1\]

\[\text{if } i + evenp[i] > R \text{ then}\]
\[L \leftarrow i - evenp[i] + 1\]
\[R \leftarrow i + evenp[i]\]

return \(evenP\)

\[\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle\]
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
<th>All pal. substr?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>$\mathcal{O}(n^3)$</td>
<td>$\Theta(n)$</td>
<td>✔</td>
</tr>
<tr>
<td>Standard algo</td>
<td>$\mathcal{O}(n^2)$</td>
<td>$\Theta(n)$</td>
<td>✔</td>
</tr>
<tr>
<td>Rabin-Karp + bin. search</td>
<td>$\mathcal{O}(n \log n)$ w.h.p</td>
<td>$\Theta(1)$</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>$\mathcal{O}(n^2 \log n)$</td>
<td>$\Theta(1)$</td>
<td>✗</td>
</tr>
<tr>
<td>Manacher</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
<td>✔</td>
</tr>
<tr>
<td>Suffix trees</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
<td>✔</td>
</tr>
</tbody>
</table>
Selection Two Sorted Arrays
Find the $k$th smallest element among two sorted arrays $A[1 \ldots m]$ and $B[1 \ldots n]$, where $k \in [1, (m + n)]$.

- **Input:** $[10, 30, 40, 60, 70, 80, 100]$, $[20, 50, 90, 110]$, and $k = 9$
  - **Output:** 90

- **Input:** $[10, 30, 40, 60, 70, 80, 100]$, $[20, 50, 90, 110]$, and $k = 4$
  - **Output:** 40
1. Concatenate the two arrays and sort it.
2. Return the $k$-th smallest element.

```
Selection-Concatenate&Sort(A[1...m], B[1...n], k)
```

Create an array $M[1...(m+n)]$

$M[1...m] \leftarrow A[1...m]$  
// Copy the first array to $M$

$M[(m+1)...(m+n)] \leftarrow B[1...n]$  
// Copy the second array to $M$

$\text{SORT}(M[1...(m+n)])$  
// Sort $M$

$\text{return } M[k]$  
// Return the $k$-th smallest element of $M$

$$\langle \text{Time, Space} \rangle = \langle \Theta \left((m+n) \log(m+n)\right), \Theta \left(m+n\right) \rangle$$
Solutions → Concatenate and heapify

1. Concatenate the two arrays and build a heap
2. Return the $k$th smallest element

**Selection-Concatenate&Heapify** $(A[1 \ldots m], B[1 \ldots n], k)$

Create a heap $H[1 \ldots (m + n)]$

$H[1 \ldots m] \leftarrow A[1 \ldots m]$  // Copy the first array to $H$

$H[(m + 1) \ldots (m + n)] \leftarrow B[1 \ldots n]$  // Copy the second array to $H$

$\text{HEAPIFY}(H[1 \ldots (m + n)])$  // Construct heap from $H$ in linear time

// RemoveMin from $H$ for a total of $k$ times

for $i \leftarrow 1$ to $k$ do

|  $\text{result} \leftarrow H.\text{RemoveMin}()$

return $\text{result}$

\[
\langle \text{Time, Space} \rangle = \langle \Theta ((m + n) + k \log(m + n)), \Theta (m + n) \rangle
\]
1. Merge the two sorted arrays
2. Return the \( k \)th smallest element

\[
\text{SELECTION-MERGE}(A[1 \ldots m], B[1 \ldots n], k)
\]

\[
i \leftarrow 1; \ j \leftarrow 1; \ \ell \leftarrow 1
\]

Create an array \( M[1 \ldots (m + n)] \)

// Merge the two sorted arrays to \( M \) until an array becomes empty
while \( i \leq m \) and \( j \leq n \) do

if \( A[i] \leq B[j] \) then { \( M[\ell] \leftarrow A[i]; \ i \leftarrow i + 1 \) }
else { \( M[\ell] \leftarrow B[j]; \ j \leftarrow j + 1 \) }

\( \ell \leftarrow \ell + 1 \)

// Copy the remaining elements to \( M \)
if \( i > m \) then \( M[\ell \ldots (m + n)] \leftarrow B[j \ldots n] \)
else if \( j > n \) then \( M[\ell \ldots (m + n)] \leftarrow A[i \ldots m] \)

// Return the \( k \)th smallest element of \( M \)
return \( M[k] \)

\[ \langle \text{Time, Space} \rangle = \langle \Theta (m + n), \Theta (m + n) \rangle \]
1. Merge the two sorted arrays without using extra space to find the $k$th smallest element

\[
\text{Selection-MergeOptimized}(A[1 \ldots m], B[1 \ldots n], k)
\]

\[
i \leftarrow 1; \ j \leftarrow 1
\]

\[
\text{// Iterate for } k \text{ elements}
\]

\[
\text{while } k > 0 \text{ do}
\]

\[
\text{// If one of the arrays is reached}
\]

\[
\text{if } i > m \text{ then return } B[j + k - 1]
\]

\[
\text{if } j > n \text{ then return } A[i + k - 1]
\]

\[
\text{// Merge-like operations}
\]

\[
\text{if } A[i] < B[j] \text{ then } i \leftarrow i + 1
\]

\[
\text{else if } A[i] \geq B[j] \text{ then } j \leftarrow j + 1
\]

\[
k \leftarrow k - 1
\]

\[
\text{return min}(A[i], B[j])
\]

\[
\langle \text{Time, Space} \rangle = \langle \Theta (k), \Theta (1) \rangle
\]
1. Find the middle indices $mid_1$ and $mid_2$ of the two arrays
2. Compare $mid_1 + mid_2$ and $k$ and compare $A[mid_1]$ and $B[mid_2]$. Recursively call the algorithm on a smaller subproblem depending on the four cases.
Core idea

- Step 1. Select $mid1$ and $mid2$ of arrays $A$ and $B$, respectively
- Step 2. We get 4 cases:
  - Case 1: $mid1 + mid2 < k$ and $A[mid1] > B[mid2]$
  - Case 2: $mid1 + mid2 < k$ and $A[mid1] \leq B[mid2]$
  - Case 3: $mid1 + mid2 \geq k$ and $A[mid1] > B[mid2]$
  - Case 4: $mid1 + mid2 \geq k$ and $A[mid1] \leq B[mid2]$
- Step 3. Eliminate half of an array in each case and recurse
• Case 1: \( \text{mid}_1 + \text{mid}_2 < k \) and \( A[\text{mid}_1] > B[\text{mid}_2] \)

\[
\begin{align*}
\text{A}[\text{mid}_1] &= 70 \\
\text{B}[\text{mid}_2] &= 50
\end{align*}
\]

\( \text{mid}_1 \) and \( \text{mid}_2 \)

\( k \)-th smallest can’t be in the first half of the second array \( B \)
Eliminate the first half of the second array \( B \)
Find and return \((k - \text{mid}_2)\)-th smallest in the remaining elements

\( \text{Selection-DeC}(A, B[(\text{mid}_2 + 1) \ldots n], k - \text{mid}_2) \)
Case 2: $mid_1 + mid_2 < k$ and $A[mid_1] \leq B[mid_2]$

$A[mid_1] = 30$

$B[mid_2] = 50$

$k$th smallest can't be in the first half of the first array $A$

Eliminate the first half of the first array $A$

Find and return $(k - mid_1)$th smallest in the remaining elements

$\text{SELECTION-DE&C}(A[(mid_1 + 1) \ldots m], B, k - mid_1)$
Case 3: $\text{mid}_1 + \text{mid}_2 \geq k$ and $A[\text{mid}_1] > B[\text{mid}_2]$

- $A[\text{mid}_1] = 70$
- $B[\text{mid}_2] = 50$

$k$th smallest can’t be in the second half of the first array $A$

Eliminate the second half of the first array $A$

Find and return $k$th smallest in the remaining elements

$\text{Selection-De&C}(A[1 \ldots \text{mid}_1], B, k)$
Case 4: \( mid1 + mid2 \geq k \) and \( A[mid1] \leq B[mid2] \)

- The \( k \)th smallest can’t be in the second half of the second array \( B \)
- Eliminate the second half of the second array \( B \)
- Find and return the \( k \)th smallest in the remaining elements

\[
\text{Selection-De\&C}(A, B[1 \ldots mid2], k)
\]
Selection-De&C(A[1...m], B[1...n], k)

// If an array is empty, return k'th element of other array
if m = 0 then return B[k]
if n = 0 then return A[k]

// Recursive case: Find the midpoint of each array
mid1 ← ⌊m/2⌋; mid2 ← ⌊n/2⌋
if mid1 + mid2 < k and A[mid1] > B[mid2] then
  // k'th smallest can't be in the first half of the second array B
  return Selection-De&C(A, B[(mid2 + 1)...n], k - mid2)
else if mid1 + mid2 < k and A[mid1] ≤ B[mid2] then
  // k'th smallest can't be in the first half of the first array A
  return Selection-De&C(A[(mid1 + 1)...m], B, k - mid1)
else if mid1 + mid2 ≥ k and A[mid1] > B[mid2] then
  // k'th smallest can't be in the second half of the first array A
  return Selection-De&C(A[1...mid1], B, k)
else if mid1 + mid2 ≥ k and A[mid1] ≤ B[mid2] then
  // k'th smallest can't be in the second half of the second array B
  return Selection-De&C(A, B[1...mid2], k)

⟨Time, Space⟩ = ⟨O(log(m + n)), O(log(m + n))⟩
1. Select index $i$ in the first array $A$, where $i = \min(m, \lfloor k/2 \rfloor)$
2. Select index $j$ in the second array $B$, where $j = k - i$
3. Compare $A[i]$ and $B[j]$
4. If $A[i] > B[j]$, discard the first $j$ elements of $B$
   Find the $(k - j)$th smallest element recursively
5. If $A[i] \leq B[j]$, discard the first $i$ elements of $A$
   Find the $(k - i)$th smallest element recursively

Does index $j$ always exist in array $B$? No!
Problem can be solved by considering the shorter array as $A$. 
Solutions → Binary search (recursive)

- Case 1: $i + j \leq k$ and $A[i] > B[j]$

$k$th smallest can’t be in the first $j$ elements of the second array $B$

Eliminate the first $j$ elements of the second array $B$

Find and return $(k - j)$th smallest in the remaining elements

$\text{Selection-BinarySearch}(A, B[j + 1 \ldots n], k - j)$
• Case 2: $i + j \leq k$ and $A[i] \leq B[j]$

\[ A[i] = 30 \]

\[ B[j] = 50 \]

$k$th smallest can't be in the first $i$ elements of the first array $A$

Eliminate the first $i$ elements of the first array $A$

Find and return $(k - i)$th smallest in the remaining elements

\[ \text{Selection-BinarySearch}(A[i + 1 \ldots n], B, k - i) \]
## Selection-BinarySearch \( (A[1 \ldots m], B[1 \ldots n], k) \)

// First array should be the shorter of the two arrays
if \( m > n \) then
    return Selection-BinarySearch \( (B[1 \ldots n], A[1 \ldots m], k) \)

// If first array is empty, return the \( k \)th element of the second array
if \( m = 0 \) then return \( B[k] \)

// If \( k = 1 \), return the minimum of the first elements of the two arrays
if \( k = 1 \) then return \( \min(A[1], B[1]) \)

// Pick the number of elements that will be discarded in the two arrays
\( i \leftarrow \min(m, \lfloor k/2 \rfloor); j \leftarrow k - i \)

// If \( A[i] > B[j] \), then discard the first \( j \) elements of \( B \)
if \( A[i] > B[j] \) then
    return Selection-BinarySearch \( (A, B[j + 1 \ldots n], k - j) \)

// If \( A[i] \leq B[j] \), then discard the first \( i \) elements of \( A \)
else if \( A[i] \leq B[j] \) then
    return Selection-BinarySearch \( (A[i + 1 \ldots n], B, k - i) \)

\[ \langle \text{Time, Space} \rangle = \langle \Theta (\log k), \Theta (\log k) \rangle \]
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Extra Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concatenate-sort</td>
<td>$\Theta ((m + n) \log (m + n))$</td>
<td>$\Theta (m + n)$</td>
</tr>
<tr>
<td>Concatenate-heapify</td>
<td>$\Theta ((m + n) + k \log (m + n))$</td>
<td>$\Theta (m + n)$</td>
</tr>
<tr>
<td>Merge</td>
<td>$\Theta (m + n)$</td>
<td>$\Theta (m + n)$</td>
</tr>
<tr>
<td>Merge optimized</td>
<td>$\Theta (k)$</td>
<td>$\Theta (1)$</td>
</tr>
<tr>
<td>De&amp;C (recursive)</td>
<td>$\Theta (\log (m + n))$</td>
<td>$\Theta (\log (m + n))$</td>
</tr>
<tr>
<td>Binary search (recursive)</td>
<td>$\Theta (\log k)$</td>
<td>$\Theta (\log k)$</td>
</tr>
</tbody>
</table>
Problem

- Given an array of reals, find the subarray with the largest sum, and return its sum.
- Input: \([-2, 1, -3, 4, -1, 2, 1, -5, 4]\)
  Output: 6
- Input: \([5, 4, -1, 7, 8]\)
  Output: 23
Solutions $\rightarrow$ Brute force

\begin{verbatim}
BruteForce(A[1...n])

    max ← −∞
    for i ← 1 to n do
        for j ← i to n do
            sum ← Sum(A[i...j])
            if sum > max then max ← sum
    return max

⟨Time, Space⟩ = Θ(n^3), Θ(1)
\end{verbatim}
Solutions → Optimized brute force

- For all subarrays, calculate the sum of the subarray as you travel the array.

```
OptimizerBruteForce(A[1...n])

max ← −∞
for i ← 1 to n do
    sum ← 0
    for j ← i to n do
        sum ← sum + A[j]
        if sum > max then max ← sum
    return max
```

⟨Time, Space⟩ = ⟨Θ(n^2), Θ(1)⟩
### Optimized brute force

<table>
<thead>
<tr>
<th>$i$</th>
<th>4</th>
<th>-1</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Initial:**
  - $i = -5, j = 4; \quad sum = 0; \quad max = -\infty$

- **Iteration:**
  - $i = -5; \quad j = 4; \quad sum = -5; \quad max = -5$
  - $i = -5; \quad j = 1; \quad sum = -1; \quad max = -1$
  - $i = -5; \quad j = 1; \quad sum = -2; \quad max$ is not updated
  - $i = -5; \quad j = 1; \quad sum = -2; \quad max$ is not updated
  - $i = -5; \quad j = 1; \quad sum = -1; \quad max$ is not updated
  - $i = -5; \quad j = 1; \quad sum = 5; \quad max = 5$
  - $i = -5; \quad j = 1; \quad sum = 10; \quad max = 10$
  - $i = -5; \quad j = 1; \quad sum = 4; \quad max = 10$

- **Final:**
  - $i = -5; \quad j = 1; \quad sum = 7; \quad max$ is not updated
  - $i = -5; \quad j = 1; \quad sum = 0$

Largest Subarray Sum
1. Divide the given array in two halves
2. Return the maximum of the following three:
   (a) Max subarray sum in left half (recurse)
   (b) Max subarray sum in right half (recurse)
   (c) Max subarray sum so that the subarray crosses the midpoint

\[
\text{DivideAndConquer}(A[1 \ldots n])
\]

\[
\begin{align*}
\text{if } n &= 1 \text{ then return } A[1] \\
\text{mid} &\leftarrow \left\lfloor \frac{n}{2} \right\rfloor \\
S_{\text{left}} &\leftarrow \text{DivideAndConquer}(A[1 \ldots \text{mid}]) \\
S_{\text{right}} &\leftarrow \text{DivideAndConquer}(A[\text{mid} + 1 \ldots n]) \\
S_{\text{merge}} &\leftarrow \text{Merge}(A[1 \ldots n], \text{mid})
\end{align*}
\]

\[
\text{return Max}(S_{\text{left}}, S_{\text{right}}, S_{\text{merge}})
\]

\[
\langle \text{Time, Space} \rangle = \langle \Theta(n \log n), \Theta(\log n) \rangle
\]
### Merge($A[1...n], mid$)

// Find the maximum suffix in the first half
suffixmax ← −∞; sum ← 0
for i ← mid to 1 do
    sum ← sum + $A[i]$
    if sum > suffixmax then suffixmax ← sum

// Find the maximum prefix in the second half
prefixmax ← −∞; sum ← 0
for i ← mid + 1 to n do
    sum ← sum + $A[i]$
    if sum > prefixmax then prefixmax ← sum

// Max subarray sum so that subarray crosses the midpoint
return (suffixmax + prefixmax)
Divide-and-conquer

\[ A = \begin{bmatrix} 3 & -2 & 1 & 1 & -3 & 5 & -1 & 2 & 1 & -2 & 0 & 1 & 2 & 3 & -5 & 4 \end{bmatrix} \]

\[ \text{suffixmax} = \begin{bmatrix} 6 \\ 3 \\ 4 \\ 5 \end{bmatrix} \]

\[ \text{prefixmax} = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 2 \end{bmatrix} \]

\[ \text{suffixmax} + \text{prefixmax} = \begin{bmatrix} 11 \end{bmatrix} \]
1. Create a class called Node for any subproblem (subarray)
2. For any subproblem/subarray, we define a node with four values:
   \( \text{sum} = \) largest subarray sum, \( \text{total} = \) total subarray sum
   \( \text{prefixmax} = \) max prefix sum, \( \text{suffixmax} = \) max suffix sum
3. Compute and return the sum corresponding to the node of \( A[1 \ldots n] \) using D&C

\begin{verbatim}
IMPROVED_DIVIDE_AND_CONQUER(A[1...n])
Node answer ← GET_MAX_SUM_SUBARRAY(A[1...n])
return answer.sum

GET_MAX_SUM_SUBARRAY(A[low...high])
if low = high then return NODE(A[low])
mid ← ⌊(low + high)/2⌋
node_ℓ ← GET_MAX_SUM_SUBARRAY(A[low...mid])
node_r ← GET_MAX_SUM_SUBARRAY(A[mid + 1...high])
return MERGE(node_ℓ, node_r)
\end{verbatim}
Solutions → Improved divide-and-conquer

\[
\text{Merge}(\ell, r)
\]

\[
x \leftarrow \text{Node}(0)
\]

// Max prefix subarray sum
\[
x.\text{prefixmax} \leftarrow \text{MAX}(\ell.\text{prefixmax}, \ell.\text{total} + r.\text{prefixmax}, \ell.\text{total} + r.\text{total})
\]

// Max suffix subarray sum
\[
x.\text{suffixmax} \leftarrow \text{MAX}(r.\text{suffixmax}, r.\text{total} + \ell.\text{suffixmax}, \ell.\text{total} + r.\text{total})
\]

// Total subarray sum
\[
x.\text{total} \leftarrow \ell.\text{total} + r.\text{total}
\]

// Max subarray sum
\[
x.\text{sum} \leftarrow \text{MAX}(x.\text{prefixmax}, x.\text{suffixmax}, x.\text{total}, \ell.\text{sum}, r.\text{sum}
\]
\[
\ell.\text{suffixmax} + r.\text{prefixmax}
\]

\text{return } x

\[
\langle \text{Time, Space} \rangle = \langle \Theta (n), \Theta (\log n) \rangle
\]
Node $x$ for array $A$

$x$.prefixmax

$x$.suffixmax

$x$.total

$x$.sum
Solutions → Improved divide-and-conquer

Node $x$ for the entire array

- Node $\ell$ for left half
  - $\ell.\text{prefixmax}$
  - $\ell.\text{total}$

- Node $r$ for right half
  - $r.\text{prefixmax}$
  - $r.\text{total}$

$x.\text{prefixmax} = \max$

$x.\text{suffixmax} = \max$

Node $x$ for the entire array

- Node $\ell$ for left half
  - $\ell.\text{suffixmax}$
  - $\ell.\text{total}$

- Node $r$ for right half
  - $r.\text{suffixmax}$
  - $r.\text{total}$

Solutions $\rightarrow$ Improved divide-and-conquer

Node $x$ for the entire array

Node $\ell$ for left half  |  Node $r$ for right half

- $\ell.\text{prefixmax}$
- $\ell.\text{total}$  |  $r.\text{prefixmax}$
- $\ell.\text{suffixmax}$  |  $r.\text{suffixmax}$
- $\ell.\text{total}$  |  $r.\text{total}$
- $\ell.\text{sum}$
- $r.\text{sum}$

$x.\text{sum} = \text{max}$
1. Iterate through the array. For each number, add it to the sum we are building.
2. If sum is smaller than the element value, we know it isn’t worth keeping, so throw it away.
3. Update max (max subarray sum) every time we find a new maximum.

KadaneAlgorithm($A[1 \ldots n]$)

```plaintext
max ← A[1]; sum ← A[1]
for i ← 2 to n do
    // If sum is negative, throw it away. Otherwise, keep adding to it.
    sum ← Max(A[i], sum + A[i])
    max ← Max(max, sum)
return max
```

$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$
Solutions → Kadane’s algorithm

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\text{arr}[i]$</th>
<th>$\text{sum}$</th>
<th>$\text{max}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>1</td>
<td>-3</td>
<td>-3</td>
<td>-2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>-2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

Initialize $\text{sum}$ & $\text{max}$ to the first element

As $\text{arr}[i] > \text{sum} + \text{arr}[i]$, $\text{sum} = \text{arr}[i]$; As $\text{sum} < \text{max}$, $\text{max}$ isn’t updated

As $\text{arr}[i] < \text{sum} + \text{arr}[i]$, $\text{sum} += \text{arr}[i]$; As $\text{sum} < \text{max}$, $\text{max}$ isn’t updated

As $\text{arr}[i] < \text{sum} + \text{arr}[i]$, $\text{sum} += \text{arr}[i]$; As $\text{sum} < \text{max}$, $\text{max}$ isn’t updated

As $\text{arr}[i] > \text{sum} + \text{arr}[i]$, $\text{sum} = \text{arr}[i]$; As $\text{sum} > \text{max}$, $\text{max} = \text{sum}$

As $\text{arr}[i] < \text{sum} + \text{arr}[i]$, $\text{sum} += \text{arr}[i]$; As $\text{sum} < \text{max}$, $\text{max}$ isn’t updated

As $\text{arr}[i] < \text{sum} + \text{arr}[i]$, $\text{sum} += \text{arr}[i]$; As $\text{sum} < \text{max}$, $\text{max}$ isn’t updated

As $\text{arr}[i] < \text{sum} + \text{arr}[i]$, $\text{sum} += \text{arr}[i]$; As $\text{sum} < \text{max}$, $\text{max}$ isn’t updated

Largest Subarray Sum
### Complexity

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>$\Theta(n^3)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Optimized brute force</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Divide-and-conquer</td>
<td>$\Theta(n \log n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>Improved divide-and-conquer</td>
<td>$\Theta(n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>Kadane’s algorithm</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>
Loop in a Linked List
You are given a singly linked list that may contain a loop. Your task is to detect the presence of a loop in the linked list, and if there exists a loop, then find the node where the loop begins (intersection node), and calculate the length of the cycle.

**Input:** A singly linked list

**Output:** The following information should be returned:
- Detection of loop: True if a loop is detected, false otherwise.
- Intersection node: If a loop is detected, return the node where the loop begins. If there is no loop, return null.
- Length of the cycle: If a loop is detected, return the length of the cycle. If there is no loop, return 0.
Problem

Input:

1 → 2 → 3 → 4 → 5

Output: (true, node 5, 5)
Step 1. Create and initialize variables:

- curr: pointer to current node
- prev: pointer to the previous node
- currd: distance of the curr pointer from head
  (distance is measured in the number of nodes)
- prevd: distance of the prev pointer from head

Step 2. Repeat until the end of SLL is reached or currd < prevd

Move curr and prev pointers one node at a time

Update currd and prevd distances

Step 3. If curr reaches null, there is no loop

Step 4. If currd < prevd, there is a loop with:
- Intersection node = curr
- Loop length = prevd − currd + 1
Solutions → Storing length

**LOOPINALINKEDLIST**(head)

// Step 1. Create and initialize variables
curr ← head; prev ← head; currd ← 1; prevd ← 0
// Step 2. Move curr and prev pointers by one unit until end of SLL is reached
// or curr pointer is at a smaller distance than that of prev pointer from head
while curr ≠ null and currd > prevd do
    prev ← curr // previous pointer
    prevd ← currd // distance to previous pointer
    curr ← curr.next // current pointer
    currd ← Distance(head, curr) // distance to current pointer
// Step 3. If curr = null, there is no loop
if curr = null then return (false, null, null)
// Step 4. If curr ≠ null, there is a loop starting at node curr with a length
// of prevd − currd + 1
else return (true, curr, prevd − currd + 1)

**Distance**(first, last)

// Find the number of nodes between first and last pointers
count ← 1; current ← first
while current ≠ last do
    count ← count + 1; current ← current.next
return count

⟨Time, Space⟩ = ⟨O(n^2), O(1)⟩
Step 1
prevd = 0, currd = 1
Since prevd < currd, continue

Step 2
prevd = 1, currd = 2
Since prevd < currd, continue

Step 3
prevd = 2, currd = 3
Since prevd < currd, continue

Step 4
prevd = 3, currd = 4
Since prevd < currd, continue

Step 5
prevd = 4, currd = 5
Since prevd < currd, continue

Step 6
prevd = 5, currd = 6
Since prevd < currd, continue
Step 7  
prevd = 6, currd = 7  
Since prevd < currd, continue

Step 8  
prevd = 7, currd = 8  
Since prevd < currd, continue

Step 9  
prevd = 8, currd = 9  
Since prevd < currd, continue

Step 10  
prevd = 9, currd = 10  
Since prevd < currd, continue

Step 11  
prevd = 10, currd = 6  
Since prevd > currd, loop detected
Use a hash map to keep track nodes visited. Iterate through the linked list to check if node is present in the hash map. If found, there exists a loop else add node and its position to the hash map. The intersection node is identified as the first node already present in the hash map. As we’re already storing node positions \{node number : position\} in the hash map, the length of the loop equals the difference between the current node count and the value of the first repeated node in the hash map.
LoopInALinkedList\( (head) \)

Create a hash map \( H \)

\(\text{count} \leftarrow 0; \text{curr} \leftarrow \text{head}\)

// Traverse the singly linked list using the \textit{curr} pointer

\textbf{while} \( \text{curr} \neq \text{null} \) \textbf{do}

  // If \textit{curr} node exists in the hash map, there is a loop

  \textbf{if} \( \text{H.Contains}(\text{curr}) \) \textbf{then}

    \text{loopleft} \leftarrow (\text{count} - \text{H}[\text{curr}])

    \textbf{return} (\text{true}, \text{curr}, \text{loopleft})

  // If \textit{curr} node is visited for the first time, add it to the map

  \text{H.Add}(\text{curr}, \text{count})

  \text{count} \leftarrow \text{count} + 1; \text{curr} \leftarrow \text{curr}.\text{next}

// If the end of the linked list is reached, there is no loop

\textbf{return} (\text{false}, \text{null}, \text{null})

\langle \text{Time, Space} \rangle = \langle \mathcal{O} (n), \mathcal{O} (n) \rangle
**Solutions → Hashing**

1. **Step 1**
   Since 1 is not in $H$, insert 1 in $H$ and continue.

2. **Step 2**
   Since 2 is not in $H$, insert 2 in $H$ and continue.

3. **Step 3**
   Since 3 is not in $H$, insert 3 in $H$ and continue.

4. **Step 4**
   Since 4 is not in $H$, insert 4 in $H$ and continue.

5. **Step 5**
   Since 5 is not in $H$, insert 5 in $H$ and continue.

6. **Step 6**
   Since 6 is not in $H$, insert 6 in $H$ and continue.
Since 7 is not in $H$, Insert 7 in $H$ and continue

Since 8 is not in $H$, Insert 8 in $H$ and continue

Since 9 is not in $H$, Insert 9 in $H$ and continue

Since 10 is not in $H$, Insert 10 in $H$ and continue

Step 11
Since 6 is in $H$, Loop detected break
Step 1. Use two pointers to scan the linked list:

- tortoise: slow pointer moves one node at a time
- hare: fast pointer moves two nodes at a time

Step 2. Compare tortoise and hare each time

Step 3. If the end of the list is reached, then there is no loop

Step 4. If tortoise = hare, then there is a loop.

Now compute:

- looptlength: count the list length starting from and ending at tortoise
- intersection: start tortoise from head and hare from looptlength distance from head; advance both pointers one node at a time until they meet; this node is the intersection node
LoopInALinkedList(head)

// Step 1. tortoise and hare are slow and fast pointers, respectively
tortoise ← head; hare ← head

// Step 2. Scan the linked list and compare tortoise and hare
while hare ≠ null and hare.next ≠ null do
    tortoise ← tortoise.next
    hare ← hare.next.next
    if tortoise = hare then break

// Step 3. If end of list is reached, there is no loop
if hare = null or hare.next = null then
    return (false, null, null)

// Step 4. If tortoise = hare, there is a loop
looplanlength ← LoopLength(tortoise)
intersection ← Intersection(head, looplanlength)
return (true, intersection, looplanlength)

⟨Time, Space⟩ = ⟨O(n), O(1)⟩
Solutions → Slow and fast pointers

**Looplength** *(curr)*

\[
\text{loopleft} \leftarrow 1; \text{loop} \leftarrow \text{curr}.next \\
\textbf{while} \ \text{loop} \neq \text{curr} \ \textbf{do} \\
\quad \text{loop} \leftarrow \text{loop}.next \\
\quad \text{loopleft} \leftarrow \text{loopleft} + 1 \\
\textbf{return} \ \text{loopleft}
\]

**Intersection** *(head, loopleft)*

\[
\text{tortoise} \leftarrow \text{head}; \text{hare} \leftarrow \text{head} \\
\text{while} \ \text{loopleft} > 0 \ \textbf{do} \\
\quad \text{hare} \leftarrow \text{hare}.next \\
\quad \text{loopleft} \leftarrow \text{loopleft} − 1 \\
\quad \textbf{while} \ \text{tortoise} \neq \text{hare} \ \textbf{do} \\
\quad \quad \text{hare} \leftarrow \text{hare}.next \\
\quad \quad \text{tortoise} \leftarrow \text{tortoise}.next \\
\quad \quad \text{tortoise} \leftarrow \text{tortoise}.next \\
\quad \textbf{return} \ \text{tortoise}
\]
Solutions → Slow and fast pointers

Step 1
Begin traversal,
tortoise increment by 1
hare increment by 2

Step 2
Since hare ≠ tortoise,
tortoise increment by 1
hare increment by 2

Step 3
Since hare ≠ tortoise,
tortoise increment by 1
hare increment by 2

Step 4
Since hare ≠ tortoise,
tortoise increment by 1
hare increment by 2

Step 5
Since hare ≠ tortoise,
tortoise increment by 1
hare increment by 2

Step 6
Since hare ≠ tortoise,
tortoise increment by 1
hare increment by 2
Step 7
Since \( hare \neq tortoise \),
\( tortoise \) increment by 1
\( hare \) increment by 2

Step 8
Since \( hare \neq tortoise \),
\( tortoise \) increment by 1
\( hare \) increment by 2

Step 9
Since \( hare = tortoise \),
loop detected
break
This algorithm is identical to the previous algorithm, except that
• We check in each of the two steps of *hare* if it meets *tortoise*, to avoid *hare* jumping the *tortoise*
LoopInALinkedList(head)

// Step 1. tortoise and hare are slow and fast pointers, respectively
tortoise ← head; hare ← head

// Step 2. Scan the linked list and compare tortoise and hare
while hare ≠ null and hare.next ≠ null do
  tortoise ← tortoise.next
  hare ← hare.next
  if tortoise = hare then break
  hare ← hare.next
  if tortoise = hare then break

// Step 3. If the end of the list is reached, there is no loop
if hare = null or hare.next = null then
  return (false, null, null)

// Step 4. If tortoise = hare, there is a loop
looplength ← LoopLength(tortoise)
intersection ← Intersection(head, looplength)
return (true, intersection, looplength)

⟨Time, Space⟩ = ⟨\mathcal{O}(n), \mathcal{O}(1)⟩
Solutions → Brent’s algorithm

- Step 1. Use two pointers to scan the linked list:
  - tortoise: slow pointer transports directly to fast pointer position
  - hare: fast pointer moves steps nodes at a time
  where, steps is initially 2
- Step 2. Compare tortoise and hare for each step of hare until steps
  Increase steps to \( F(steps) \) (Brent used \( F(x) = 2x \))
  tortoise transports directly to hare position
- Step 3. If the end of the list is reached, then there is no loop
- Step 4. If tortoise = hare, then there is a loop.
  Now compute loopleft and intersection as before
### Brent's algorithm

**Algorithm**

```plaintext
LOOP_IN_A_LINKED_LIST(head)

// Step 1. *tortoise* and *hare* are slow and fast pointers, respectively
tortoise ← head; hare ← head; steps ← 1

// Step 2. Scan the linked list and compare *tortoise* and *hare*. Advance *hare* by \( F(\text{steps}) \) at a time and *tortoise* to *hare* position.

while true do
  for i ← 1 to steps do
    if hare = null then break
    hare ← hare.next
    if hare = tortoise then break
    if hare = null or hare = tortoise then break
    tortoise ← hare
    steps ← \( F(\text{steps}) \)
  // transport *tortoise* to *hare*
  // update *steps*

// Step 3. If the end of the list is reached, there is no loop
if hare = null then return (false, null, null)

// Step 4. If *hare* = *tortoise*, there is a loop
looppLength ← LOOP_LENGTH(tortoise)
intersection ← INTERSECTION(head, looppLength)
return (true, intersection, looppLength)
```

\( \langle \text{Time, Space} \rangle = \langle \mathcal{O}(n), \mathcal{O}(1) \rangle \)
Step 1
Begin traversal,
\( \text{steps } \leftarrow 2 \)
\( \text{hare } \leftarrow \text{hare.next} \)

Step 2
\( \text{steps traversed } = 1, \) Since \( \text{hare } \neq \text{tortoise}, \)
\( \text{hare } \leftarrow \text{hare.next} \)

Step 3
Since \( \text{steps} \) are done,
\( \text{tortoise } \leftarrow \text{hare} \)
\( \text{step } = 2 \times \text{step } = 4 \)

Step 4
Restart traversal,
\( \text{steps } \leftarrow 4 \)
\( \text{hare } \leftarrow \text{hare.next} \)

Step 5
\( \text{steps traversed } = 1, \) Since \( \text{hare } \neq \text{tortoise}, \)
\( \text{hare } \leftarrow \text{hare.next} \)

Step 6
\( \text{steps traversed } = 2, \) Since \( \text{hare } \neq \text{tortoise}, \)
\( \text{hare } \leftarrow \text{hare.next} \)
Step 7
steps traversed = 3,
Since hare \neq tortoise,
hare \leftarrow hare.next

Step 8
Since steps are done,
tortoise \leftarrow hare
step = 2 \times step = 8

Step 9
Restart traversal,
steps \leftarrow 4
hare \leftarrow hare.next

Step 10
steps traversed = 1,
Since hare \neq tortoise,
hare \leftarrow hare.next

Step 11
steps traversed = 2,
Since hare \neq tortoise,
hare \leftarrow hare.next

Step 12
steps traversed = 3,
Since hare \neq tortoise,
hare \leftarrow hare.next
Solutions → Brent’s algorithm

Step 13
steps traversed = 4,
Since hare ≠ tortoise,
hare ← hare.next

Step 14
Since hare = tortoise,
loop detected
break
### Complexity

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Extra space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storing length</td>
<td>$\mathcal{O}(n^2)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>Hashing</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(n)$</td>
</tr>
<tr>
<td>Slow-fast pointers</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>Slow-fast pointers alternative</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
<tr>
<td>Brent's algorithm</td>
<td>$\mathcal{O}(n)$</td>
<td>$\mathcal{O}(1)$</td>
</tr>
</tbody>
</table>
Y-shaped Linked List
There are two singly linked lists of sizes $m$ and $n$, respectively. Due to some error, the two linked lists are connected in Y-shape. Design an efficient algorithm to determine the point of intersection of the two lists given their head nodes i.e., $head_1$ and $head_2$. 
Solutions → Brute force

1. Run two nested loops. One loop for list 1 and another for list 2.
2. If the two pointers match then that is the intersection node. Else return null.

```
YShapedLinkedList-BruteForce(head1, head2)

pointer1 ← head1
  // Outer loop for nodes in list 1
while pointer1 ≠ null do
  pointer2 ← head2
  // Inner loop for nodes in list 2
  while pointer2 ≠ null do
    // First time the two pointers are the same is the intersection
    if pointer1 = pointer2 then
      return pointer1
    pointer2 ← pointer2.next()
  pointer1 ← pointer1.next()
return null
```

\[ \langle \text{Time, Space} \rangle = \langle \mathcal{O}(mn), \Theta(1) \rangle \]
Solutions → Two stacks

1. Store all nodes of list 1 in stack 1 and list 2 in stack 2
2. Pop the same node pointers from both stacks until the pointers are different
3. The last same node reference is the intersecting node

YShapedLinkedList-TwoStacks(head1, head2)

Create two stacks $S_1$ and $S_2$

$ptr_1 ← head_1; ptr_2 ← head_2; result ← null$

// Store all nodes of list 1 in stack 1 and list 2 in stack 2
while $ptr_1 ≠ null$ do { $S_1$.Push($ptr_1$); $ptr_1 ← ptr_1$.next }
while $ptr_2 ≠ null$ do { $S_2$.Push($ptr_2$); $ptr_2 ← ptr_2$.next }

// If the two stack tops are different then there is no intersection
if $S_1$.Top() ≠ $S_2$.Top() then return null

// Keep popping the same node references from the two stacks, the last node reference that is same is the intersection node
while $S_1$ is not empty and $S_2$ is not empty and $S_1$.Top() = $S_2$.Top() do
    { result ← $S_1$.Pop(); $S_2$.Pop(); }

// Return the intersecting node
return result

⟨Time, Space⟩ = ⟨Θ (m + n), Θ (m + n)⟩
Solutions → Hashset

1. Store every node reference of list 1 in a hashset
2. Scan each node reference of list 2 and check if it exists in the hashset

```javascript
YShapedLinkedList-Hashset(head1, head2)

pointer1 ← head1; pointer2 ← head2
Create a hashset $H$ to store node references
  // Store every node reference of list 1 in a hashset
  while pointer1 ≠ null do
    H.Add(pointer1)
    pointer1 ← pointer1.next()
  // Scan each node pointer of list 2 and check if it exists in the hashset
  while pointer2 ≠ null do
    if H.ContainsKey(pointer2) then
      return pointer2
    pointer2 ← pointer2.next()
  return null
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(m + n), \Theta(m) \rangle$$
1. Find the difference $diff$ in the lengths of the lists. This is the length of the bottom portion of $Y$.
2. Advance the pointer of the longer list by $diff$.
3. Now move pointers of both longer and shorter one node at a time until there the intersection node is found. Else return $null$

```
YShapedLinkedList-DifferenceCount(head1, head2)

// Find the difference in the lengths of the lists. Determine which list is longer and set the longer and shorter lists accordingly
m ← ComputeLength(head1); n ← ComputeLength(head2)
if m > n then { diff ← m − n; longer ← head1; shorter ← head2 }
else { diff ← n − m; longer ← head2; shorter ← head1 }

// Advance the pointer of the longer list by the difference in lengths
for i ← 1 to diff do longer ← longer.next()

// Iterate through both lists until the pointers meet at the merge point
while longer ≠ shorter do
| longer ← longer.next(); shorter ← shorter.next()
return longer

⟨Time, Space⟩ = ⟨Θ (m + n), Θ (1)⟩
```
1. Traverse the first linked list, count the elements, and make a circular linked list. Remember the last node so that we can break the circle later on.

2. Transformed problem: Finding the loop in the second linked list.

3. Since we already know the length of the loop (size of the first linked list), we can traverse those many nodes in the second list. Then, start another pointer from the beginning of the second list and traverse until they are equal, which is the required intersection point.

4. Remove the circular structure from the linked list.
Length of the first list $=\text{length of the loop} = 8$

$pointer1$ (at $C_3$) is 8 steps ahead of $pointer2$ (at $B_1$)
Solutions → Loop in a linked list

Step 5

Step 6

Intersection found!

Step 7
YShapedLinkedList-LoopInALinkedList(\textit{head1}, \textit{head2})

\begin{verbatim}
pointer1 \leftarrow \textit{head1}; pointer2 \leftarrow \textit{head2}; lastnode \leftarrow \text{null}

// Traverse the first linked list and make it circular
length1 \leftarrow 0
\textbf{while} pointer1.next \neq \text{null} \textbf{do}
| \hspace{0.5cm} pointer1 \leftarrow pointer1.next; length1 \leftarrow length1 + 1
lastnode \leftarrow pointer1; pointer1.next \leftarrow head1

// Set one of the pointers ahead
pointer1 \leftarrow head2
\textbf{while} length1 > 0 \textbf{do}
| \hspace{0.5cm} pointer1 \leftarrow pointer1.next; length1 \leftarrow length1 - 1

// Traverse until they are equal, which is the intersection point
\textbf{while} pointer1 \neq pointer2 \textbf{do}
| \hspace{0.5cm} pointer1 \leftarrow pointer1.next; pointer2 \leftarrow pointer2.next

// Remove the circular structure from the linked list
lastnode.next \leftarrow \text{null}
\textbf{return} pointer1
\end{verbatim}

\[\langle\text{Time, Space}\rangle = \langle\Theta (m + n), \Theta (1)\rangle\]
Let

\( X = \) length of the first list until the intersection point
\( Y = \) length of the second list until the intersection point
\( Z = \) length from intersection node (inclusive) to the last node

1. Traverse the second list and find length \( L_2 \)
2. Traverse the first list and reverse it and find length \( L_1 \)
3. Traverse the new second list and find length \( L_3 \)
4. We have system of 3 equations and 3 unknowns. Solve:

\[
X + Z = L_1; \quad Y + Z = L_2; \quad X + Y = L_3
\]

5. We get:

\[
X = \frac{1}{2} \cdot (L_1 + L_3 - L_2); \quad Y = \frac{1}{2} \cdot (L_2 + L_3 - L_1); \quad Z = \frac{1}{2} \cdot (L_1 + L_2 - L_3)
\]

6. Find the intersection node by traversing from the new second list by \( Y \) steps
7. Reverse the first linked list (if required)
Solutions → System of linear equations
Based on $L_1$, $L_2$ and $L_3$, compute $X$, $Y$ and $Z$.
YShapedLinkedList-LinearEquations(head1, head2)

// Compute the length of the second and first linked lists
L1 ← GetLinkedListLength(head1)
L2 ← GetLinkedListLength(head2)

// Reverse first linked list and compute L3
reversedhead1 ← ReverseLinkedList(head1)
L3 ← GetLinkedListLength(head2)

// Solve the equations for X, Y, and Z
X = \frac{1}{2} \cdot (L_1 + L_3 - L_2);
Y = \frac{1}{2} \cdot (L_2 + L_3 - L_1);
Z = \frac{1}{2} \cdot (L_1 + L_2 - L_3)

// Traverse the second linked list to the intersection point and return
answer ← head2
for i ← 1 to Y do  answer ← answer.next

// Restore first linked list by reversing it again
head1 ← ReverseLinkedList(reversedhead1)

return answer

⟨Time, Space⟩ = ⟨Θ (m + n), Θ (1)⟩
Solutions → System of linear equations

**GetLinkedListLength**(head)

\[
L \leftarrow 0; \text{ pointer } \leftarrow \text{ head} \\
\textbf{while} \text{ pointer } \neq \text{ null } \textbf{ do} \\
| \quad L \leftarrow L + 1; \text{ pointer } \leftarrow \text{ pointer}.\text{next} \\
\textbf{return} \ L
\]

**ReverseLinkedList**(head)

\[
\text{current } \leftarrow \text{ head}; \text{ previous } \leftarrow \text{ null}; \text{ nextcurrent } \leftarrow \text{ null} \\
\quad \text{// Iterate through the list and reverse pointers} \\
\textbf{while} \text{ current } \neq \text{ null } \textbf{ do} \\
\quad \text{nextcurrent } \leftarrow \text{ current}.\text{next} \\
\quad \text{current}.\text{next } \leftarrow \text{ previous} \\
\quad \text{previous } \leftarrow \text{ current} \\
\quad \text{current } \leftarrow \text{ nextcurrent} \\
\textbf{return} \ \text{previous}
\]
1. Scan list 1 using \(\text{pointer1}\). Scan list 2 using \(\text{pointer2}\).
2. If \(\text{pointer1}\) reaches list 1 end, then start from list 2. If \(\text{pointer2}\) reaches list 2 end, then start from list 1.
3. At any moment, when the two node references are same, it is the intersection node. Else, return \(\text{null}\).

\[
\text{YShapedLinkedList-TwoPointers}(\text{head1}, \text{head2})
\]

\[
\text{pointer1} \leftarrow \text{head1}; \text{pointer2} \leftarrow \text{head2}
\]

\[
\text{// If one of the lists is empty, then there is no intersection node}
\text{if pointer1 = null or pointer2 = null then return null}
\]

\[
\text{// Traverse the lists until the intersection node is found}
\text{while pointer1 \neq pointer2 do}
\]

\[
\text{pointer1} \leftarrow \text{pointer1.next()}; \text{pointer2} \leftarrow \text{pointer2.next()}
\]

\[
\text{if pointer1 = pointer2 then return pointer1}
\]

\[
\text{// If a pointer reaches its list end, then start from other list}
\text{if pointer1 = null then pointer1 \leftarrow \text{head2}}
\]

\[
\text{if pointer2 = null then pointer2 \leftarrow \text{head1}}
\]

\[
\text{return pointer1}
\]

\[
\langle \text{Time, Space} \rangle = \langle \mathcal{O}(m + n), \Theta(1) \rangle
\]
Solutions → Two pointers

Step 1

Step 2

Step 3

Step 4

Step 5

Step 6
Solutions → Two pointers

Step 7

Step 8

Step 9

Step 10

Step 11

Step 12
Solutions → Two pointers

Step 13
Step 14
Intersection found!
Step − 15
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Extra Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>$\mathcal{O}(mn)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Two stacks</td>
<td>$\Theta(m + n)$</td>
<td>$\Theta(m + n)$</td>
</tr>
<tr>
<td>HashSet</td>
<td>$\mathcal{O}(m + n)$</td>
<td>$\Theta(m)$</td>
</tr>
<tr>
<td>Difference count</td>
<td>$\Theta(m + n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Loop in a linked list</td>
<td>$\Theta(m + n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Linear equations</td>
<td>$\Theta(m + n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Two pointers</td>
<td>$\mathcal{O}(m + n)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>
Problem

- Search if an element $k$ exists in a $m \times n$ sorted matrix $A[1 \ldots m, 1 \ldots n]$.
- If the element $k$ exists, then return the location (i.e., row and column) of one cell whose value is $k$.
- Input: Sorted matrix $A$ of size $m \times n$
  Output: Location where $k$ exists, $-1$ otherwise.
Solutions $\rightarrow$ Linear search

\[
\text{SearchInSortedMatrix}(A[1\ldots m, 1\ldots n], k)
\]

\[
\begin{align*}
\text{for } i & \leftarrow 1 \text{ to } m \text{ do} \\
& \quad \text{for } j \leftarrow 1 \text{ to } n \text{ do} \\
& \quad \quad \text{if } A[i, j] = k \text{ then} \\
& \quad \quad \quad \text{return } (i, j) \\
\text{return } -1
\end{align*}
\]

\[
\langle \text{Time}, \text{Space} \rangle = \langle O(mn), \Theta(1) \rangle
\]
Core idea

- Step 1. Select the mid element
- Step 2. We get 3 cases:
  - Case 0: mid element \(= k\)
  - Case 1: mid element \(> k\)
  - Case 2: mid element \(< k\)
- Step 3. Eliminate a quadrant in Cases 1 and 2 and recurse
• Case 1: mid element > \( k \)
  Suppose mid element = 50 and \( k = 30 \)

\[ \begin{array}{ccc}
  & I & II \\
  III & 50 & IV \\
\end{array} \]

\( k \) can’t be in quadrant IV

Search for \( k \) in quadrants I, II, and III
Case 2: mid element $< k$
Suppose mid element $= 50$ and $k = 70$

$k$ can’t be in quadrant I
Search for $k$ in quadrants II, III, and IV
**SearchInSortedMatrix**($A[1 \ldots m, 1 \ldots n], k$)

\[
\text{return } \text{D&C}(A[1 \ldots m, 1 \ldots n], k)
\]

\[
\text{D&C}(A[r_\ell \ldots r_h, c_\ell \ldots c_h], k)
\]

\[\text{if } r_\ell > r_h \text{ or } c_\ell > c_h \text{ then return } -1\]

\[r_m \leftarrow (r_\ell + r_h)/2; \quad c_m \leftarrow (c_\ell + c_h)/2\]

\[\text{if } A[r_m, c_m] = k \text{ then return } (r_m, c_m)\]

// In this case, element is definitely not in the fourth quadrant

\[\text{else if } A[r_m, c_m] > k \text{ then}\]

\[\text{return } \text{D&C}(A[r_\ell \ldots r_m - 1, c_\ell \ldots c_m - 1], k) \text{ or } \]

\[\text{D&C}(A[r_\ell \ldots r_m-1, c_m \ldots c_h], k) \text{ or } \]

\[\text{D&C}(A[r_m \ldots r_h, c_\ell \ldots c_m - 1], k) \text{ or } \]

// In this case, element is definitely not in the first quadrant

\[\text{else if } A[r_m, c_m] < k \text{ then}\]

\[\text{return } \text{D&C}(A[r_\ell \ldots r_m, c_m + 1 \ldots c_h], k) \text{ or } \]

\[\text{D&C}(A[r_m+1 \ldots r_h, c_\ell \ldots c_m], k) \text{ or } \]

\[\text{D&C}(A[r_m + 1 \ldots r_h, c_m + 1 \ldots c_h], k) \text{ or } \]

// fourth quadrant

\[
\langle \text{Time, Space} \rangle = \langle \mathcal{O} \left( \min(m, n)^{\log_2 3} \right), \mathcal{O} (\log \max(m, n)) \rangle
\]
Core idea

- Step 1. Select the mid-row
- Step 2. Do a binary search in the mid-row for the largest index $index$ such that the array element at that index is not greater than $k$
- Step 3. We get 3 cases:
  - Case 1: there is no such index
  - Case 2: element (at index) $= k$
  - Case 3: element (at index) $< k$
- Step 4. Do the following:
  - Case 1: search the upper rectangle recursively
  - Case 2: return the location
  - Case 3: search the second and third regions recursively

In Cases 1 and 3, we eliminate at least half of the elements
• Case 1: there is no index
  i.e., suppose the first element in the mid row is 70, which is greater than $k = 50$

$k$ can’t be in the lower half
Search for $k$ in the upper half
Area of lower half is $> 50\%$
Case 3: element $< k$

Suppose element $= 30$, next element $= 70$, and $k = 50$

$k$ can’t be in regions I and IV

Search for $k$ in regions II and III

Combined area of regions I and IV is $> 50\%$
$k = 10$

<table>
<thead>
<tr>
<th>Binary search row</th>
<th>$k$ not found, 9 is max element $&lt; k$, 16 is min element $&gt; k$</th>
<th>Eliminating red zones, Recursively searching white top-right and bottom-left sub-matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4 7 11 15</td>
<td>1 4 7 11 15</td>
<td>1 4 7 11 15</td>
</tr>
<tr>
<td>2 5 8 12 19</td>
<td>2 5 8 12 19</td>
<td>2 5 8 12 19</td>
</tr>
<tr>
<td>3 6 9 16 22</td>
<td>3 6 9 16 22</td>
<td>3 6 9 16 22</td>
</tr>
<tr>
<td>10 13 14 17 24</td>
<td>10 13 14 17 24</td>
<td>10 13 14 17 24</td>
</tr>
<tr>
<td>11 21 23 26 30</td>
<td>11 21 23 26 30</td>
<td>11 21 23 26 30</td>
</tr>
</tbody>
</table>
Solutions → D&C improved

\[
\text{SearchInSortedMatrix}(A[1 \ldots m, 1 \ldots n], k)
\]

\[
\text{return } \text{D&C-Improved}(A[1 \ldots m, 1 \ldots n], k)
\]

\[
\text{D&C-Improved}(A[r_\ell \ldots r_h, c_\ell \ldots c_h], k)
\]

\[
\begin{align*}
\text{if } c_\ell & > c_h \text{ or } r_\ell > r_h \text{ then return } -1 \\
& \quad \text{// Binary search returns the largest index } j \text{ in } [c_\ell \ldots c_h] \text{ for which } A[r_m, j] \leq k. \text{ If no such index exists, it returns } -1 \\
r_m & \leftarrow (r_\ell + r_h)/2 \\
j & \leftarrow \text{BinarySearch}(A[r_m, c_\ell \ldots c_h]) \\
\text{if } j = -1 \text{ then} \\
& \quad \text{return } \text{D&C-Improved}(A[r_\ell \ldots r_m - 1, c_\ell \ldots c_h], k) \quad \text{// upper half} \\
\text{else if } A[r_m, j] = k \text{ then} \\
& \quad \text{return } (r_m, j) \\
\text{else if } A[r_m, j] < k \text{ then} \\
& \quad \text{return } \text{D&C-Improved}(A[r_\ell \ldots r_m - 1, j + 1 \ldots c_h], k) \text{ or } \text{D&C-Improved}(A[r_m + 1 \ldots r_h, c_\ell \ldots j], k) \quad \text{// region II} \\
\end{align*}
\]

\[
\langle \text{Time, Space} \rangle = \langle \mathcal{O}(m \log n), \mathcal{O}(\log m) \rangle
\]
\( k = 8 \)

# columns < # rows, so binary search on each column

\[
\begin{array}{ccccccc}
 j & -10 & -7 & -4 & -2 & 1 \\
-8 & -5 & -3 & -1 & 3 \\
-5 & -3 & -2 & 0 & 5 \\
-2 & -1 & 1 & 3 & 6 \\
0 & 2 & 4 & 5 & 7 \\
3 & 4 & 7 & 8 & 9 \\
5 & 6 & 9 & 10 & 11 \\
10 & 12 & 15 & 17 & 20
\end{array}
\]

\[
\begin{array}{ccccccc}
 j & -10 & -7 & -4 & -2 & 1 \\
-8 & -5 & -3 & -1 & 3 \\
-5 & -3 & -2 & 0 & 5 \\
-2 & -1 & 1 & 3 & 6 \\
0 & 2 & 4 & 5 & 7 \\
3 & 4 & 7 & 8 & 9 \\
5 & 6 & 9 & 10 & 11 \\
10 & 12 & 15 & 17 & 20
\end{array}
\]

\[
\begin{array}{ccccccc}
 j & -10 & -7 & -4 & -2 & 1 \\
-8 & -5 & -3 & -1 & 3 \\
-5 & -3 & -2 & 0 & 5 \\
-2 & -1 & 1 & 3 & 6 \\
0 & 2 & 4 & 5 & 7 \\
3 & 4 & 7 & 8 & 9 \\
5 & 6 & 9 & 10 & 11 \\
10 & 12 & 15 & 17 & 20
\end{array}
\]

Element found!
Perform a binary search for $k$ in each row or column

\[
\text{SearchInSortedMatrix}(A[1 \ldots m, 1 \ldots n], k)
\]

\[
\begin{aligned}
&\text{// } \#\text{rows} < \#\text{columns} \\
&\text{if } m < n \text{ then} \\
&\quad \text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
&\quad \quad j \leftarrow \text{BinarySearch}(A[i, 1 \ldots n], k) \\
&\quad \quad \text{if } j \neq -1 \text{ then return } (i, j) \\
&\text{// } \#\text{columns} \leq \#\text{rows} \\
&\text{else} \\
&\quad \text{for } j \leftarrow 1 \text{ to } n \text{ do} \\
&\quad \quad j \leftarrow \text{BinarySearch}(A[1 \ldots m, j], k) \\
&\quad \quad \text{if } j \neq -1 \text{ then return } (i, j) \\
&\text{return } -1
\end{aligned}
\]

\[
\langle \text{Time, Space} \rangle = \langle O(\min(m, n) \log \max(m, n)), \Theta(1) \rangle
\]
Core idea

- **Step 1.** Select the top-right element
- **Step 2.** We get 2 cases:
  - Case 0: element \( = k \)
  - Case 1: element \( > k \)
  - Case 2: element \( < k \)
- **Step 3.**
  - If Case 1, select the left element, and repeat Step 2
  - If Case 2, select the down element, and repeat Step 2
Solutions → Decrease-and-conquer

\[
\text{SearchInSortedMatrix}(A[1 \ldots m, 1 \ldots n], k)
\]

// Start from the top right element
\[\text{row} \leftarrow 1; \text{col} \leftarrow n\]

\[\text{while row} \leq m \text{ and col} \geq 1 \text{ do}\]
\[\text{if } A[\text{row}, \text{col}] = k \text{ then}\]
\[\text{return } (\text{row}, \text{col})\]

// In this case, go left as column \text{col} (down elements) can’t have \(k\)
\[\text{else if } k < A[\text{row}, \text{col}] \text{ then}\]
\[\text{col} \leftarrow \text{col} - 1\]

// In this case, go down as row \text{row} \text{ (left elements) can’t have } k\)
\[\text{else if } k > A[\text{row}, \text{col}] \text{ then}\]
\[\text{row} \leftarrow \text{row} + 1\]

\[\text{return } -1\]

\[\langle \text{Time, Space} \rangle = \langle \mathcal{O}(m + n), \Theta(1) \rangle\]
\[ k = 10 \]

Starting from top right

\[
\begin{array}{cccccc}
1 & 4 & 7 & 11 & 15 \\
2 & 5 & 8 & 12 & 19 \\
3 & 6 & 9 & 16 & 22 \\
10 & 13 & 14 & 17 & 24 \\
11 & 21 & 23 & 26 & 30 \\
\end{array}
\]

Element found!
Can we start from any corner?
- No!
- We can start from
  - top-right (and go left or down)
  - bottom-left (and go up or right)
- We cannot start from
  - top-left (and go right or down)
  - bottom-right (and go left or up)
- Why can’t we start from top-left or bottom-right?
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear search</td>
<td>$O(mn)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>D&amp;C</td>
<td>$O\left(\min(m, n)^{\log 3}\right)$</td>
<td>$O(\log \max(m, n))$</td>
</tr>
<tr>
<td>Improved D&amp;C</td>
<td>$O(m \log n)$</td>
<td>$O(\log m)$</td>
</tr>
<tr>
<td>Binary search</td>
<td>$O\left(\min(m, n) \log \max(m, n)\right)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Decrease-and-conquer</td>
<td>$O(m + n)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>
• Given an array $A[1 \ldots n]$ of unique integers, design an efficient approach to find the smallest missing natural number.

• Input: $[2, -9, 5, 11, 1, -10, 7]$
  Output: 3

• Extension: What if we allow duplicates or repetitions?
Solutions → Brute force 1

1. Check if $i$ is missing in the array $A[1 \ldots n]$ for $i \in [1 \ldots n]$
2. Stop and return the smallest such $i$, otherwise return $n + 1$

```
FUNCTION FirstMissingPositive-BruteForce1(A[1...n])

    // Check if the any natural number from 1 to n is missing
    FOR i ← 1 to n do
        imissing ← true
        // Iterate over the array to check if the natural number exists
        FOR j ← 1 to n do
            // If i is found then break
            IF i = A[j] then
                imissing ← false
                break
        // Missing value found
        IF imissing = true then
            return i
    return n + 1
```

$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n^2), \Theta(1) \rangle$
1. Create an empty sorted set $S$ to add all natural numbers from array
2. Check if $i$ is missing in the array $A[1\ldots n]$ for $i \in [1\ldots n]$
3. Stop and return the smallest such $i$, otherwise return $n + 1$

<table>
<thead>
<tr>
<th>FirstMissingPositive-BruteForce2($A[1\ldots n]$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>// Create a sorted set to store the natural numbers</td>
</tr>
<tr>
<td>Create an empty sorted set $S$ using a balanced search tree</td>
</tr>
<tr>
<td>for $i \leftarrow 1$ to $n$ do</td>
</tr>
<tr>
<td>if $A[i] &gt; 0$ then</td>
</tr>
<tr>
<td>$S$.Add($A[i]$)</td>
</tr>
<tr>
<td>// Find the first missing natural number from $A[1\ldots n]$ using $S$</td>
</tr>
<tr>
<td>for $i \leftarrow 1$ to $n$ do</td>
</tr>
<tr>
<td>if $S$ does not contain $i$ then</td>
</tr>
<tr>
<td>return $i$</td>
</tr>
<tr>
<td>return $n + 1$</td>
</tr>
</tbody>
</table>

$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n \log n), \mathcal{O}(n) \rangle$
1. Sort the input array in-place to skip non-natural numbers
2. Check if \(i\) is missing in the array \(A[1 \ldots n]\) for \(i \in [1 \ldots n]\)
3. Stop and return the smallest such \(i\), otherwise return \(n + 1\)

\[
\text{\textbf{FIRSTMISSINGPOSITIVE-SCAN}}(A[1 \ldots n])
\]

\[
\begin{align*}
&\quad // \text{ Sort the array in-place} \\
&\quad \text{\textbf{SORT}}(A[1 \ldots n]) \\
&\quad // \text{ Skip negative numbers and zero from the array} \\
&\quad \text{index} \leftarrow 1 \\
&\quad \text{\textbf{while}} \ A[\text{index}] < 1 \text{ \textbf{do} \ index} \leftarrow \text{index} + 1 \\
&\quad \text{i} \leftarrow 1 \\
&\quad // \text{ Find the missing natural number from the sorted input array} \\
&\quad \text{\textbf{for}} \ j \leftarrow \text{index} \ \text{\textbf{to}} \ n \ \text{\textbf{do}} \\
&\quad \quad \text{if} \ A[j] = i \ \text{\textbf{then} \ i} \leftarrow i + 1 \\
&\quad \quad \text{\textbf{else}} \ \text{\textbf{if}} \ A[j] > i \ \text{\textbf{then}} \\
&\quad \quad \quad \text{return} \ i \\
&\quad \text{return} \ i
\end{align*}
\]

\[
\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n \log n), \Theta(1) \rangle
\]
1. Use $i \in [1 \ldots n]$ of the same array to mark the presence of the numbers
2. If $A[i]$ is a natural number and $i \leq n$, swap & place it in $A[A[i]]$
3. Stop and return smallest $i$ where $A[i] \neq i$, otherwise return $n + 1$

```
\textsc{FirstMissingPositive-InPlaceHashing}(A[1 \ldots n])

\begin{verbatim}
// Swap natural number $A[i]$ to $A[i]$th index if $A[i] \in [1 \ldots n]$
for $i \leftarrow 1 \to n$ do
    while $A[i] \geq 1$ and $A[i] \leq n$ and $A[i] \neq A[A[i]]$ do
        Swap$(A[i], A[A[i]])$
    // Find the first natural number that is not $A[i] \neq i$ in $A[1 \ldots n]$
for $i \leftarrow 1 \to n$ do
    if $A[i] \neq i$ then
        return $i$
return $n + 1$
\end{verbatim}
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$
For each $i \in [1, n]$, do this in a loop.


Return the first $i$ which is not equal to $A[i]$

That is, return 5
Solutions → In-place hashing

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
A & -4 & 3 & 1 & 8 & 2 & 6 & -1 & 4 \\
i = 1; \ A[i] \not\in [1, n], \text{ so do nothing} \\
\hline
A & -4 & 3 & 1 & 8 & 2 & 6 & -1 & 4 \\
i = 2; \ A[i] \in [1, n] \text{ and } A[i] \neq A[A[i]], \text{ so Swap}(A[i], A[A[i]]) \\
\hline
A & -4 & 1 & 3 & 8 & 2 & 6 & -1 & 4 \\
i = 2; \ A[i] \in [1, n] \text{ and } A[i] \neq A[A[i]], \text{ so Swap}(A[i], A[A[i]]) \\
\hline
A & 1 & -4 & 3 & 8 & 2 & 6 & -1 & 4 \\
i = 2; \ A[i] \not\in [1, n], \text{ so do nothing} \\
\hline
A & 1 & -4 & 3 & 8 & 2 & 6 & -1 & 4 \\
i = 3; \ A[i] \in [1, n] \text{ but } A[i] = A[A[i]], \text{ so do nothing} \\
\hline
A & 1 & -4 & 3 & 8 & 2 & 6 & -1 & 4 \\
i = 4; \ A[i] \in [1, n] \text{ and } A[i] \neq A[A[i]], \text{ so Swap}(A[i], A[A[i]]) \\
\hline
A & 1 & -4 & 3 & 8 & 2 & 6 & -1 & 8 \\
i = 4; \ A[i] \in [1, n] \text{ but } A[i] = A[A[i]], \text{ so do nothing} \\
\hline
A & 1 & -4 & 3 & 4 & 2 & 6 & -1 & 8 \\
i = 5; \ A[i] \in [1, n] \text{ and } A[i] \neq A[A[i]], \text{ so Swap}(A[i], A[A[i]]) \\
\hline
A & 1 & -4 & 3 & 4 & 2 & 6 & -1 & 8 \\
i = 5; \ A[i] \not\in [1, n], \text{ so do nothing} \\
\hline
A & 1 & 2 & 3 & 4 & -4 & 6 & -1 & 8 \\
i = 6; \ A[i] \in [1, n] \text{ but } A[i] = A[A[i]], \text{ so do nothing} \\
\hline
A & 1 & 2 & 3 & 4 & -4 & 6 & -1 & 8 \\
i = 7; \ A[i] \not\in [1, n], \text{ so do nothing} \\
\hline
A & 1 & 2 & 3 & 4 & -4 & 6 & -1 & 8 \\
i = 8; \ A[i] \in [1, n] \text{ but } A[i] = A[A[i]], \text{ so do nothing} \\
\end{array}
\]
1. Insert all the array numbers in a hashtable $H$
2. Find the first natural number $i$ that is not present in $H$

```
FIRSTMISSINGPOSITIVE-HashTable(A[1...n])

    // Create a HashTable to store the natural numbers
    Create a HashTable $H$
    for $i$ ← 1 to $n$ do $H[A[i]]$ ← true
    // Find the first missing natural number from $A[1...n]$ using $H$
    for $i$ ← 1 to $n + 1$ do
        if $H$ does not contain $i$ then
            return $i$
```

$\langle$Time, Space$\rangle = \langle\mathcal{O}(n)$, $\mathcal{O}(n)\rangle$

**This Solution Might Not Always Work. Why?**
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute Force 1</td>
<td>$\mathcal{O}(n^2)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Brute Force 2</td>
<td>$\mathcal{O}(n \log n)$</td>
<td>$\mathcal{O}(n)$</td>
</tr>
<tr>
<td>Scan</td>
<td>$\mathcal{O}(n \log n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>In-Place Hashing</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>In-Place Hashing &amp; Partition</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>
The knowledge of \( n \) guests in a party is represented by a binary matrix \( M[1 \ldots n, 1 \ldots n] \), where \( M[i, j] = 1 \) means that person \( i \) knows person \( j \). Given this binary matrix, find a celebrity if there exists a celebrity, where, a celebrity is a person who is known by everyone and doesn’t know anyone.

If there are multiple celebrities, return any one celebrity. (Prove that there cannot be more than one celebrity)

If there are no celebrities, return \(-1\).

**Input:**

<table>
<thead>
<tr>
<th>( i : j )</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Output:** 2
Core idea (take-home lesson)

\[ i \text{ knows } j, \text{ i.e., } M[i, j] = 1 \]

\[ i \text{ is not a celebrity} \]
\[ j \text{ is a potential celebrity} \]

\[ i \text{ doesn’t know } j, \text{ i.e., } M[i, j] = 0 \]

\[ j \text{ is not a celebrity} \]
\[ i \text{ is a potential celebrity} \]
Solutions $\rightarrow$ Brute force

1. We iterate over each person and check if they are a celebrity.
2. The inner for loop determines whether a person is a celebrity by verifying if they are known by everyone and don’t know anyone.

```
FindCelebrity-BruteForce(M[1...n, 1...n])

for i ← 1 to n do
    iscelebrity ← true // assume person i is a celebrity
    for j ← 1 to n do
        // Skip self
        if i = j then continue
        // Check if person i knows j or if person j doesn't know i
        if M[i, j] = 1 or M[j, i] = 0 then
            iscelebrity ← false
            break
    if iscelebrity then
        return i
return -1
```

$\langle$Time, Space$\rangle = \langle \mathcal{O}(n^2), \Theta(1) \rangle$
• *Indegree* of person $i$ is the number of people who know $i$
• *Outdegree* of person $i$ is the number of people $i$ knows
• We calculate the *indegree* and *outdegree* for each person based on their relationships in $M$.
• Next, we iterate through the guests to find the *celebrity*, i.e., a person who has an *indegree* of $n - 1$ (knows everyone except self) and an *outdegree* of 0 (is not known by anyone).
### FindCelebrity($M[1...n, 1...n]$)

#### Step 1. Calculate outdegree and indegree of each node

<table>
<thead>
<tr>
<th>outdegree$[1...n]$ $\leftarrow [0...0]$; indegree$[1...n]$ $\leftarrow [0...0]$</th>
</tr>
</thead>
</table>

```plaintext
for i $\leftarrow 1$ to $n$ do
  for j $\leftarrow 1$ to $n$ do
    // As i knows j, increment outdegree of i and indegree of j
    if $M[i, j] = 1$ then
      outdegree$[i]$ $\leftarrow$ outdegree$[i]$ + 1
      indegree$[j]$ $\leftarrow$ indegree$[j]$ + 1
```

#### Step 2. Finding the celebrity

```plaintext
for i $\leftarrow 1$ to $n$ do
  if outdegree$[i] = 0$ and indegree$[i] = n - 1$ then
    return i // celebrity found
```

return $-1$

\[\langle \text{Time, Space} \rangle = \langle \Theta \left(n^2\right), \Theta \left(n\right) \rangle\]
### Solutions → Graph

#### Table: Graph Parameters

<table>
<thead>
<tr>
<th>M</th>
<th>outdegree</th>
<th>indegree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>3</td>
<td>1 1 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M</th>
<th>outdegree</th>
<th>indegree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>3</td>
<td>1 1 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M</th>
<th>outdegree</th>
<th>indegree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>3</td>
<td>1 1 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M</th>
<th>outdegree</th>
<th>indegree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 1 1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>3</td>
<td>1 1 0</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

#### Graph Diagram

- **Outdegree**: The number of edges leaving each vertex.
- **Indegree**: The number of edges entering each vertex.
- **Celebrity**: A vertex with a specific role or status.
The algorithm recursively finds a potential celebrity in the first $n - 1$ elements of the matrix $M$.

It checks the base case to return 1 when $n$ is 1, indicating that the only person is the celebrity.

If no celebrity is found in the first $n - 1$ person, it considers $n$ as the potential celebrity.

It checks if the potential celebrity knows person $n - 1$. If yes, $n - 1$ is the celebrity.

If the potential celebrity doesn’t know person $n - 1$, the previously found celebrity(id) is the celebrity.

The wrapper function ensures that the potential celebrity is a real celebrity based on the matrix $M$. 
\begin{align*}
\text{\textbf{FindCelebrity}}(M[1\ldots n, 1 \ldots n])

// Step 1. Find the celebrity candidate
\text{candidate} & \leftarrow \text{FindPotentialCelebrity}(M, n) \\
\text{if } \text{candidate} = -1 & \text{ then}
| \quad \text{return } -1
\quad // \text{ no celebrity found}

// Step 2. Check if the candidate is a celebrity
\text{outdegree} & \leftarrow 0; \text{indegree} \leftarrow 0
\text{for } i \leftarrow 1 \text{ to } n \text{ do}
| \quad \text{if } i \neq \text{candidate} \text{ then}
| | \quad \text{outdegree} \leftarrow \text{outdegree} + M[\text{candidate}, i]
| | \quad \text{indegree} \leftarrow \text{indegree} + M[i, \text{candidate}]
\text{if } \text{outdegree} = 0 \text{ and } \text{indegree} = n - 1 & \text{ then}
| | \quad \text{return } \text{candidate}
\text{return } -1 \quad // \text{ no celebrity found}

\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle
\end{align*}
**Solutions → Recursion**

\[
\text{FindPotentialCelebrity}(M[1 \ldots n, 1 \ldots n], m)
\]

\[
\begin{align*}
\text{if } m = 0 & \text{ then return } -1 \\
& \quad // \text{ Recursively find celebrity in the first } m - 1 \text{ persons} \\
\text{candidate} & \leftarrow \text{FindPotentialCelebrity}(M, m - 1) \\
& \quad // \text{ If there is no candidate in the first } m - 1 \text{ people, } m \text{ is the new candidate} \\
\text{if } \text{candidate} = -1 & \text{ then return } m \\
& \quad // \text{ If candidate knows } m \text{ person, then } m \text{ is the new candidate} \\
\text{if } M[\text{candidate}, m] = 1 & \text{ then return } m \\
& \quad // \text{ If } m \text{ knows candidate, then candidate is the new candidate} \\
\text{if } M[m, \text{candidate}] = 1 & \text{ then return candidate} \\
\text{return } -1
\end{align*}
\]
Solutions → Recursion

\[
M = \begin{bmatrix}
0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0
\end{bmatrix}
\]

\[N = 4\]

\[\text{FindPotentialCelebrity}(M, 4)\]

\[\text{FindPotentialCelebrity}(M, 3)\]

\[\text{FindPotentialCelebrity}(M, 2)\]

\[\text{FindPotentialCelebrity}(M, 1)\]

\[\text{FindPotentialCelebrity}(M, 0)\]

\[\text{Base case}\]

\[\text{Exiting with } -1\]

\[\text{candidate} = -1, \text{ so exiting with } m = 1\]

\[\text{candidate} \neq -1\]

\[M[\text{candidate}, m] = M[1, 2] = 1, \text{ so exiting with } m = 2\]

\[\text{candidate} \neq -1\]

\[M[\text{candidate}, m] = M[2, 3] \neq 1, \text{ so checking ahead}\]

\[M[m, \text{candidate}] = M[3, 2] = 1, \text{ so exiting with } m = 2\]

\[\text{candidate} \neq -1\]

\[M[\text{candidate}, m] = M[2, 4] \neq 1, \text{ so checking ahead}\]

\[M[m, \text{candidate}] = M[4, 2] = 1, \text{ so exiting with } m = 2\]

Potential celebrity is 2
• We use a stack to eliminate potential non-celebrities.
• We compare pairs of individuals to determine which one cannot be a celebrity and pushes the other back into the stack.
• After processing all pairs, one person remains in the stack.
• Check if this person is known to everyone and doesn’t know anyone to identify the celebrity.
### FindCelebrity($M[1 \ldots n, 1 \ldots n]$)

// Step 1. Find a potential celebrity
Create a stack $S \leftarrow \[]$ to store all potential celebrities
for $i \leftarrow 1$ to $n$ do $S$.Push($i$)
while Stack $S$ has greater than 1 element do
    $i \leftarrow S$.Pop(); $j \leftarrow S$.Pop(); // pop 2 elements
    // Check if $i$ knows $j$, and push the potential celebrity to stack
    if $M[i,j] = 1$ then $S$.Push($j$)
    else $S$.Push($i$)
candidate $\leftarrow S$.Pop()

// Step 2. Check if the candidate is a celebrity
for $i \leftarrow 1$ to $n$ do
    if $i \neq$ candidate then
        if $M[i, candidate] = 0$ then return $-1$
        if $M[candidate, i] = 1$ then return $-1$

return candidate

$\langle$Time, Space$\rangle = \langle \Theta \left( n \right), \Theta \left( n \right) \rangle$
Solutions → Elimination technique

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

|   |   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 1 | 1 | 0 | 0 |

|   |   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 1 | 1 | 0 | 0 |

|   |   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 1 | 1 | 0 | 0 |

|   |   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 1 | 1 | 0 | 0 |

|   |   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 1 | 1 | 0 | 0 |

1. **Potential celebrity is 2**
We iterate through the people, starting with the first person $r$.

- Check if $r$ knows the $i^{th}$ person and updates the diagonal elements accordingly.
- After processing, it checks if any person can be a celebrity by verifying if they are known by everyone and don’t know anyone.
- If a potential celebrity is found, return it; otherwise, $-1$ is returned if no celebrity is found.
**FindCelebrity**\((M[1 \ldots n, 1 \ldots n])\)

// Step 1. Find the celebrity candidate
\(r \leftarrow 1\)
for \(i \leftarrow 2\) to \(n\) do 
  if \(M[r, i] = 1\) then
    \(M[r, r] \leftarrow \ast\)  // \(r\) can’t be a celebrity
    \(r \leftarrow i\)  // update \(r\) to \(i\)
  else
    \(M[i, i] \leftarrow \ast\)  // \(i\) can’t be a celebrity

// The single candidate will have its diagonal cell as 0
\(candidate \leftarrow 1\)
while \(candidate \leq n\) do
  if \(M[candidate, candidate] = 0\) then break

// Step 2. Check if the candidate is really the celebrity
for \(i \leftarrow 1\) to \(n\) do 
  if \(i \neq candidate\) then
    if \(M[i, candidate] = 0\) then return \(-1\)
    if \(M[candidate, i] = 1\) then return \(-1\)

return \(candidate\)

\(\langle\text{Time, Space}\rangle = \langle \Theta (n), \Theta (1) \rangle\)

Assumption: The input matrix \(M\) can be updated
### Efficient elimination technique

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$r = 1$ Since $M[r,i] = M[1,2] = 1,$

$i = 2$ Set: $M[r,r] = M[1,1] \leftarrow \star$ and $r \leftarrow i = 2$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$r = 2$ Since $M[r,i]$ is 0,

$i = 3$ Set: $M[i,i] \leftarrow 1$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$r = 2$ Since $M[r,i]$ is 0,

$i = 4$ Set: $M[i,i] \leftarrow 1$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>*</td>
</tr>
</tbody>
</table>

Check Diagonals

Potential celebrity is 2
• Initialize two pointers, \( i \) at 1 (person 1) and \( j \) at \( n \) (person \( n \)).
• Iteratively check if person \( j \) knows person \( i \). If so, decrement \( j \); otherwise, increment \( i \).
• Person pointed to by \( i \) is considered the celebrity candidate.
• Verify if the celebrity candidate is known by everyone and knows no one for every \( i \).
• If the candidate satisfies these conditions, they are considered the celebrity, and their index is returned; otherwise, \(-1\) is returned if no celebrity is found.
// Step 1. Find the celebrity candidate
\[
i \leftarrow 1, \ j \leftarrow n
\]
while \( i < j \) do
    \[
    \text{if } M[j, i] = 1 \text{ then}
    \]
    \[
    j \leftarrow j - 1 \quad \text{// person } j \text{ knows person } i, \text{ so } j \text{ can’t be a celebrity}
    \]
    \[
    \text{else}
    \]
    \[
    i \leftarrow i + 1 \quad \text{// person } j \text{ doesn’t know } i, \text{ so } i \text{ can’t be a celebrity}
    \]
candidate \leftarrow i \quad \text{// person } i \text{ is the celebrity candidate}

// Step 2. Check if the candidate is really the celebrity
for \( j \leftarrow 1 \) to \( n \) do
    \[
    \text{if } j \neq \text{candidate then}
    \]
    \[
    \text{if } M[j, \text{candidate}] = 0 \text{ or } M[\text{candidate}, j] = 1 \text{ then}
    \]
    \[
    \text{return } -1 \quad \text{// candidate is not a celebrity}
    \]
return candidate \quad \text{// candidate is the celebrity}

\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle
### Solutions → Two pointers (example 1)

<table>
<thead>
<tr>
<th>M</th>
<th>People</th>
<th>Pointers</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="matrix.png" alt="Matrix" /></td>
<td><img src="people.png" alt="People" /></td>
<td><img src="pointers.png" alt="Pointers" /></td>
<td><img src="action.png" alt="Action" /></td>
</tr>
</tbody>
</table>

- **First Iteration:**
  - **i = 1:** Since \( M[j, i] = M[5, 1] = 1 \), \( j \) cannot be a celebrity. Hence, \( j \leftarrow j - 1 \).

- **Second Iteration:**
  - **i = 1:** Since \( M[j, i] = M[4, 1] = 1 \), \( j \) cannot be a celebrity. Hence, \( j \leftarrow j - 1 \).

- **Third Iteration:**
  - **i = 1:** Since \( M[j, i] = M[3, 1] = 0 \), \( i \) cannot be a celebrity. Hence, \( i \leftarrow i + 1 \).

- **Fourth Iteration:**
  - **i = 2:** Since \( M[j, i] = M[3, 2] = 1 \), \( j \) cannot be a celebrity. Hence, \( j \leftarrow j - 1 \).

- **Fifth Iteration:**
  - **i = 2:** Since \( i = j = 2 \) i.e. \( i \geq j \), Potential celebrity is 2.

- **Final Iteration:**
  - **i = 2:** Since \( i = j = 2 \) i.e. \( i \geq j \), Potential celebrity is 2.

- **Return:**
  - \( i = 2 \) turns out to be a celebrity, return 2.
### Solutions → Two pointers (example 2)

<table>
<thead>
<tr>
<th>M</th>
<th>People</th>
<th>Pointers</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="M" /></td>
<td><img src="image" alt="People" /></td>
<td><img src="image" alt="Pointers" /></td>
<td><img src="image" alt="Action" /></td>
</tr>
<tr>
<td><img src="image" alt="M" /></td>
<td><img src="image" alt="People" /></td>
<td><img src="image" alt="Pointers" /></td>
<td><img src="image" alt="Action" /></td>
</tr>
<tr>
<td><img src="image" alt="M" /></td>
<td><img src="image" alt="People" /></td>
<td><img src="image" alt="Pointers" /></td>
<td><img src="image" alt="Action" /></td>
</tr>
</tbody>
</table>

- **M**:
  - 1 2 3 4 5
  - 1 0 1 1 0 1
  - 2 0 0 0 0 0
  - 3 0 1 0 0 0
  - 4 1 1 0 0 0
  - 5 0 0 0 1 1

- **People**:
  - 1 2 3 4 5
  - i = 1
  - j = 5

- **Pointers**:
  - 1 2 3 4 5
  - i = 1
  - j = 5

- **Action**:
  - Since \( M[j, i] = M[5, 1] = 0 \),
    - i cannot be a celebrity.
    - Hence, \( i \leftarrow i + 1 \).

  - Since \( M[j, i] = M[5, 2] = 0 \),
    - i cannot be a celebrity.
    - Hence, \( i \leftarrow i + 1 \).

  - Since \( M[j, i] = M[5, 3] = 0 \),
    - i cannot be a celebrity.
    - Hence, \( i \leftarrow i + 1 \).

  - Since \( M[j, i] = M[5, 4] = 1 \),
    - j cannot be a celebrity.
    - Hence, \( j \leftarrow j - 1 \).

  - Since \( i = j = 4 \) i.e. \( i \geq j \),
    - Potential celebrity is 4

  - i = 4 turns out to not be a celebrity,
    - return −1
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>$O(n^2)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Graph</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Recursion</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Elimination</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Efficient elimination</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Two pointers</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>
Random Permutation
Generate a random permutation of $A[1, 2, \ldots, n]$. A random permutation of $A[1, 2, \ldots, n]$ is $A[p_1, p_2, \ldots, p_n]$, where $[p_1, p_2, \ldots, p_n]$ is a random permutation of $[1, 2, \ldots, n]$ with probability of occurring $1/n!$. 
RandomPermutation(A[1...n])

for i ← 1 to n – 1 do
    x ← Random([1...n])
    y ← Random([1...n])
    Swap(A[x], A[y])

The algorithm is incorrect. Counterexample:
Suppose k = #iterations for which actual swap takes place
Total #permutations = n!
Total #outcomes = ⌈n^2⌉

If #permutations does not divide #outcomes, then the output permutations are not equally likely

For n = 3 and arbitrary natural number k, #permutations (3!) does not divide #outcomes (⌈3^2⌉)
Algorithm 1

`RandomPermutation(A[1...n])`

```plaintext
for i ← 1 to n − 1 do
    x ← Random([1...n])
    y ← Random([1...n])
    Swap(A[x], A[y])
```

The algorithm is incorrect.

Counterexample:
- Suppose $k = \#$iterations for which actual swap takes place
  total $\#$permutations $= n!$
  total $\#$outcomes $= (n^2)^k$
- If $\#$permutations does not divide $\#$outcomes,
  then the output permutations are not equally likely
- For $n = 3$ and arbitrary natural number $k$,
  $\#$permutations ($3!$) does not divide $\#$outcomes ($(3^2)^k$)
Solutions → Algorithm 2

```plaintext
RandomPermutation(A[1...n])

\[
\text{for } i \leftarrow 1 \text{ to } n - 1 \text{ do} \\
\quad x \leftarrow i \\
\quad y \leftarrow \text{Random}([1...n]) \\
\quad \text{Swap}(A[x], A[y])
\]
```

The algorithm is incorrect.

Counterexample: Suppose \(k = \#\text{iterations for which actual swap takes place}\) total \(\#\text{permutations} = n!\) total \(\#\text{outcomes} = n^k\) If \(\#\text{permutations}\) does not divide \(\#\text{outcomes}\), then the output permutations are not equally likely

For \(n = 3\) and arbitrary natural number \(k\), \(\#\text{permutations} (3!)\) does not divide \(\#\text{outcomes} (n^k)\)
The algorithm is incorrect.

Counterexample:

- Suppose $k = \#\text{iterations for which actual swap takes place}$
  - total $\#\text{permutations} = n!$
  - total $\#\text{outcomes} = n^k$
- If $\#\text{permutations}$ does not divide $\#\text{outcomes}$, then the output permutations are not equally likely
- For $n = 3$ and arbitrary natural number $k$, $\#\text{permutations}$ ($3!$) does not divide $\#\text{outcomes}$ ($n^k$)
RandomPermutation($A[1 \ldots n]$)

\[
\text{for } i \leftarrow 1 \text{ to } n - 1 \text{ do}
\]
\[
\begin{align*}
    &x \leftarrow i \\
    &y \leftarrow \text{Random}([i + 1 \ldots n]) \\
    &\text{Swap}(A[x], A[y])
\end{align*}
\]
RandomPermutation($A[1 \ldots n]$)

\[
\text{for } i \leftarrow 1 \text{ to } n - 1 \text{ do} \\
\quad x \leftarrow i \\
\quad y \leftarrow \text{Random}([i + 1 \ldots n]) \\
\quad \text{Swap}(A[x], A[y])
\]

The algorithm is incorrect.

Counterexample:
- total #permutations = $n!$
  - total #outcomes = $(n - 1)!$
- If #permutations does not divide #outcomes, then the output permutations are not equally likely
- For any natural number $n > 1$,
  - #permutations ($n!$) does not divide #outcomes ($(n - 1)!$)
**Algorithm 4**

**RandomPermutation**($A[1 \ldots n]$)

1. Choose a random hash function for the hash table $H$
2.\[\text{for } i \leftarrow 1 \text{ to } n \text{ do}\]
   \[H.\text{Add}(A[i]) \quad // \quad \text{add the element even if the key exists}\]
3.\[\text{for } i \leftarrow 1 \text{ to } n \text{ do}\]
   \[A[i] \leftarrow \text{ith element in hash table } H\]
The algorithm is correct.

Proof:

- We want to prove that probability of producing random permutation is $1/n!$

\[
\text{Probability} = \frac{1}{n} \times \frac{1}{n-1} \times \cdots \times \frac{1}{2} = \frac{1}{n!}
\]
The algorithm is correct.
RandomPermutation(A[1...n])

for i ← n downto 2 do
    x ← i
    y ← Random([1...i])
    Swap(A[x], A[y])

Proof:
• Let X = [1, 2, ..., n] and Y = [p_1, p_2, ..., p_n]
• We want to prove that probability of producing Y from X is 1/n!

\[ = P(Y[n] = p_n) \]
\[ \times P(Y[n-1] = p_{n-1} | Y[n] = p_n) \]
\[ \times P(Y[n-2] = p_{n-2} | Y[n] = p_n, Y[n-1] = p_{n-1}) \]
\[ \times \cdots \]
\[ = \frac{1}{n} \times \frac{1}{n-1} \times \cdots \times \frac{1}{1} = \frac{1}{n!} \]
Problem

- Given an array of unique integers \( A[1 \ldots n] \) and a positive integer \( k \), count all distinct pairs with differences equal to \( k \).
- Input: \([8, 5, 1, 4, 2], k = 3\]
  Output: 3  \((4 - 1 = 5 - 2 = 8 - 5)\)
- Input: \([8, 12, 16, 4, 0, 20], k = 4\]
  Output: 5  \((20 - 16 = 16 - 12 = 12 - 8 = 8 - 4 = 4 - 0)\)
- Input: \([1, 1, 1, 2, 2, 2, 2], k = 1\]
  Input has duplicates, so this type of input is not allowed
• Consider every pair of elements and increment count if the difference equals $k$

\[
\text{COUNTPAIRS-BruteForce}(A[1 \ldots n], k)
\]
\[
\begin{array}{l}
count \leftarrow 0 \\
\text{for } i \leftarrow 1 \text{ to } n - 1 \text{ do} \\
\quad \text{for } j \leftarrow i + 1 \text{ to } n \text{ do} \\
\quad \\
\quad \quad \text{if } \text{absolute}(A[i] - A[j]) = k \text{ then} \\
\quad \\
\quad \quad \quad count \leftarrow count + 1 \\
\text{return } count
\end{array}
\]

$$\langle \text{Time, Space} \rangle = \langle \Theta(n^2), \Theta(1) \rangle$$
1. Sort the array, initialize $count$ to 0
2. For each element $A[i]$ for $i \in [1 \ldots n - 1]$, search for $A[i] + k$ in the remaining array $A[i + 1 \ldots n]$ using binary search
3. Each time $A[i] + k$ is found, increment $count$ by 1

```
CountingPairs-BinarySearch(A[1 \ldots n], k)

Sort(A[1 \ldots n])

count ← 0
for i ← 1 to n − 1 do
    // Check if $A[i] + k$ in $A[i + 1 \ldots n]$ using binary search
    if BinarySearch(A[i + 1 \ldots n], A[i] + k) ≠ −1 then
        count ← count + 1
return count
```

$\langle \text{Time, Space} \rangle = \langle \Theta(n \log n), \Theta(1) \rangle$
1. Add all elements of the array to the HashTable $H$
2. For each element $A[i]$ for $i \in [1 \ldots n]$, search for $A[i] + k$ and $A[i] - k$ in $H$ and increment count on finding a match
3. Return $\text{count}/2$

\[
\text{COUNTINGPAIRS-Hashing}(A[1 \ldots n], k)
\]

Create an empty HashTable $H$
for $i \leftarrow 1$ to $n$ do $H$.Add($A[i]$)

$count \leftarrow 0$
for $i \leftarrow 1$ to $n$ do
  if $H$.Contains($A[i] + k$) then $count \leftarrow count + 1$
  if $H$.Contains($A[i] - k$) then $count \leftarrow count + 1$
return $count/2$

\[
\langle \text{Time, Space} \rangle = \langle \Theta(n)^*, \mathcal{O}(n) \rangle
\]
1. Sort $A[1...n]$, initialize two pointers low and high to 1


   While $high \leq n$
   - If $diff = k$, increment $low$, $high$, and $count$
   - If $diff > k$, increment $low$
   - If $diff < k$, increment $high$

\[
\text{CountingPairs-TwoPointers}(A[1...n], k)\\
\]
\[
\begin{align*}
\text{Sort}(A[1...n]) \\
\text{count} & \leftarrow 0; \text{low} \leftarrow 1; \text{high} \leftarrow 1 \\
\text{while} \ high \leq n \ \text{do} \\
& \quad \text{diff} = A[high] - A[low] \\
& \quad \text{if} \ diff = k \ \text{then} \\
& \quad \quad \text{count} \leftarrow \text{count} + 1; \ \text{low} \leftarrow \text{low} + 1; \ \text{high} \leftarrow \text{high} + 1 \\
& \quad \text{else if} \ diff > k \ \text{then} \ \text{low} \leftarrow \text{low} + 1 \\
& \quad \text{else if} \ diff > k \ \text{then} \ \text{high} \leftarrow \text{high} + 1 \\
\text{return} \ \text{count}
\end{align*}
\]

\[
\langle \text{Time, Space} \rangle = \langle \Theta(n \log n), \Theta(1) \rangle
\]
This logic applies only for sorted arrays

$k = 3$

\[
count = 0; \ A[h] - A[\ell] < k, \text{ so increment } h
\]

\[
count = 0; \ A[h] - A[\ell] < k, \text{ so increment } h
\]

\[
count = 0; \ A[h] - A[\ell] < k, \text{ so increment } h
\]

\[
count = 0; \ A[h] - A[\ell] > k, \text{ so increment } \ell
\]

\[
count = 0; \ A[h] - A[\ell] = k, \text{ so increment } \ell, \ h, \ count
\]

\[
count = 1; \ A[h] - A[\ell] = k, \text{ so increment } \ell, \ h, \ count
\]

\[
count = 2; \ A[h] - A[\ell] < k, \text{ so increment } h
\]
### Counting pairs summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Sorting + Binary Search</td>
<td>$\Theta(n \log n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Hashing</td>
<td>$\Theta(n^*)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Sorting + Two Pointers</td>
<td>$\Theta(n \log n)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>

Solve the problem when there are duplicates in the array.
Problem

- Given an array, find the maximum and minimum elements in the array.
- We consider the number of array element comparisons for measuring time.
- Input: $A = [4, 2, 0, -2, 20, 9, 2]$
  
Output: $[20, -2]$
• Traverse the array and compare each element with $max$ and $min$.

\[
\text{BruteForce}(A[1\ldots n])
\]

\[
max \leftarrow A[1]; \ min \leftarrow A[1]
\]

\[
\text{for } i \leftarrow 2 \ \text{to} \ n \ \text{do}
\]

\[
\quad \text{if } A[i] > max \ \text{then} \quad max \leftarrow A[i]
\]

\[
\quad \text{else if } A[i] < min \ \text{then} \quad min \leftarrow A[i]
\]

\[
\text{return } (max, \ min)
\]

\[
\langle \text{Time, Space} \rangle = \langle 2n - 2, \Theta(1) \rangle
\]
Solutions → Increment by two

- Pick elements in pairs, the smaller element amongst the two becomes a candidate for min and the larger element for max.

\[
\text{IncrementByTwo}(A[1 \ldots n])
\]

\[
\begin{align*}
\text{if } n \text{ is odd then } & \quad \{ \; \text{max} \leftarrow A[1], \text{min} \leftarrow A[1], i \leftarrow 2 \; \} \\
\text{else} & \\
\quad \text{if } A[1] < A[2] \text{ then } & \quad \{ \; \text{max} \leftarrow A[2]; \text{min} \leftarrow A[1] \; \} \\
\quad \text{else} & \quad \{ \; \text{max} \leftarrow A[1]; \text{min} \leftarrow A[2] \; \} \\
\quad i \leftarrow 3 \\
\text{while } i < n \text{ do} & \\
\quad \text{if } A[i] < A[i + 1] \text{ then} & \\
\quad \quad \text{if } A[i] < \text{min} \text{ then } & \quad \text{min} \leftarrow A[i] \\
\quad \quad \text{if } A[i + 1] > \text{max} \text{ then } & \quad \text{max} \leftarrow A[i + 1] \\
\quad \text{else} & \\
\quad \quad \text{if } A[i] > \text{max} \text{ then } & \quad \text{max} \leftarrow A[i] \\
\quad \quad \text{if } A[i + 1] < \text{min} \text{ then } & \quad \text{min} \leftarrow A[i + 1] \\
\quad \quad i \leftarrow i + 2 \\
\text{return } (\text{max}, \text{min})
\end{align*}
\]

\[
\langle \text{Time, Space} \rangle = \left\langle \frac{3}{2} \left( n - 1 - \text{if } n \text{ is even} \right), \Theta(1) \right\rangle
\]
Solutions → Divide-and-conquer

1. Divide the problem into two equal size sub-problems.
2. Recursively find the max and min of left and right parts.
3. Compare the max of both halves to get the overall max, and the min of both halves to get the overall min.
**DivideAndConquer**($A[low...high]$)

```plaintext
def DivideAndConquer(A[low...high])

size ← high − low + 1

if size = 1 then  { max ← A[low]; min ← A[low] }
else if size = 2 then
    else  { max ← A[low]; min ← A[high] }
else
    mid ← ⌊(low + high)/2⌋
    (ℓmax, ℓmin) ← DivideAndConquer(A[low...mid])
    (rmax, rmin) ← DivideAndConquer(A[mid + 1...high])
    if ℓmax > rmax then  max ← ℓmax
    else  max ← rmax
    if ℓmin < rmin then  min ← ℓmin
    else  min ← rmin
return (max, min)
```

$$T(n) = \begin{cases} n - 1 & \text{if } n = 1 \text{ or } 2, \\ 2T(n/2) + 2 & \text{if } n > 2. \end{cases}$$

$$\langle \text{Time, Space} \rangle = \left\langle \frac{3n}{2} - 2, \Theta(\log n) \right\rangle$$
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute force</td>
<td>$2n - 2$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Increment by two</td>
<td>$\frac{3}{2} \left( n - 1 - \frac{n}{2} \right)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Divide-and-conquer</td>
<td>$\frac{3n}{2} - 2$</td>
<td>$\Theta(\log n)$</td>
</tr>
</tbody>
</table>
Sorting Algorithms
Problem

- Design an efficient algorithm to sort a given array $A[1 \ldots n]$.
- Input: [80, 30, 90, 50, 40, 20, 100]
  Output: [20, 30, 40, 50, 80, 90, 100]
- Input: [23, 15, 40, 15, 10]
  Output: [10, 15, 15, 23, 40]
\textbf{PermutationSort}(A[1 \ldots n])

while \textit{true} do
    \textbf{RandomPermute}(A[1 \ldots n])
    if \textbf{IsSorted}(A[1 \ldots n]) then
        break
    \textbf{RandomPermute}(A[1 \ldots n])
for \( i \leftarrow 1 \) to \( n - 1 \) do
    \( A[i] \leftarrow \text{Swap}(A[i], A[\text{Random}(i \ldots n)]) \)

\langle \text{Time, Space} \rangle = \langle \mathcal{O}(\infty), \Theta(1) \rangle
Solutions → Slow sort

1. Divide the subarray into two halves.
2. Sort the first half recursively.
3. Sort the second half recursively.
4. Swap the last elements of the two halves if they are out of order.
5. Sort the subarray except the last element recursively.

\[
\text{SLOWSORT}(A[\text{low} \ldots \text{high}])
\]

\[
\begin{array}{l}
\quad \text{if } i \geq j \text{ then return} \\
\quad \text{// Sort the two halves recursively} \\
\quad \text{mid} \leftarrow (\text{low} + \text{high})/2 \\
\quad \text{SLOWSORT}(A[\text{low} \ldots \text{mid}]) \\
\quad \text{SLOWSORT}(A[\text{mid} + 1 \ldots \text{high}]) \\
\quad \text{// The largest element of the subarray should go to its correct position} \\
\quad \text{if } A[\text{high}] < A[\text{mid}] \text{ then} \\
\quad \quad \text{Swap}(A[\text{high}], A[\text{mid}]) \\
\quad \text{// Sort the remaining subarray} \\
\quad \text{SLOWSORT}(A[\text{low} \ldots \text{high} - 1])
\end{array}
\]

\[
\langle \text{Time, Space} \rangle = \langle \mathcal{O} \left( n \log_2 n \right), \Theta (n) \rangle
\]
1. Suppose the index of the maximum element in $A[1 \ldots n]$ is $\text{maxindex}$.
2. Reverse $A[1 \ldots \text{maxindex}]$ to move the largest element in the array to index 1.

\begin{verbatim}
PancakeSort(A[1...n])

// The $i$th iteration finds the $i$th largest element
for $i \leftarrow n$ downto 2 do
    // Step 1. Find the index of the $\max(A[1\ldots i])$
    $\text{maxindex} \leftarrow 1$
    for $j \leftarrow 2$ to $i$ do
        if $A[j] > A[\text{maxindex}]$ then
            $\text{maxindex} \leftarrow j$
    // Step 2. Move $\max(A[1\ldots \text{maxindex}])$ to index 1
    Reverse($A[1\ldots \text{maxindex}]$)
    Reverse($A[1\ldots i]$)
\end{verbatim}

$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n^2), \Theta(1) \rangle$
## Pancake sort

### i = 5; maxindex = 2
Reverse $A[1\ldots maxindex]$
Reverse $A[1\ldots i]$

### i = 4; maxindex = 2
Reverse $A[1\ldots maxindex]$
Reverse $A[1\ldots i]$

### i = 3; maxindex = 1
Reverse $A[1\ldots maxindex]$
Reverse $A[1\ldots i]$

### i = 2; maxindex = 2
Reverse $A[1\ldots maxindex]$
Reverse $A[1\ldots i]$
Solutions → Stooge sort

1. If the start element is greater than the end element, swap them.
2. If there are three or more elements in the array:
   1. Recursively sort the first 2/3rd of the array
   2. Recursively sort the last 2/3rd of the array
   3. Recursively sort the first 2/3rd of the array

\[
\text{StoogeSort}(A[\ell \ldots h])
\]

\[
\begin{align*}
\text{size} & \leftarrow h - \ell + 1 \\
\text{if } (A[\ell] > A[h]) & \text{ then} \\
& \quad \text{SWAP}(A[\ell], A[h]) \\
\text{if } (\text{size} > 2) & \text{ then} \\
& \quad \text{third} \leftarrow \text{size}/3 \\
& \quad \text{StoogeSort}(A[\ell \ldots h - \text{third}]) \\
& \quad \text{StoogeSort}(A[\ell + \text{third} \ldots h]) \\
& \quad \text{StoogeSort}(A[\ell \ldots h - \text{third}])
\end{align*}
\]

\[
\langle \text{Time, Space} \rangle = \langle \Theta \left( n^{\log_{1.5} 3} \right), \Theta (\log n) \rangle
\]
# Stooge sort

<table>
<thead>
<tr>
<th>Original array</th>
<th>The original array</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 3 8 6 7 1 5 2 4</td>
<td>First (2/3)rd of the array</td>
</tr>
<tr>
<td>9 3 8 6 7 1 5 2 4</td>
<td>Sort the first (2/3)rd of the array</td>
</tr>
<tr>
<td>1 3 6 7 8 9 5 2 4</td>
<td>Last (2/3)rd of the array</td>
</tr>
<tr>
<td>1 3 6 2 4 5 7 8 9</td>
<td>Sort the last (2/3)rd of the array</td>
</tr>
<tr>
<td>1 3 6 2 4 5 7 8 9</td>
<td>First (2/3)rd of the array</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8 9</td>
<td>Sort the first (2/3)rd of the array</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8 9</td>
<td>The original array is sorted</td>
</tr>
</tbody>
</table>
Assumption

- Items are natural numbers with maximum value $k$.

1. Create an array for indices in the range $[0, k]$
2. Distribute items to these indices to compute item frequencies
3. Compute the cumulative frequencies of items for indices
   in the range $[0, k]$
4. Find the sorted array

<table>
<thead>
<tr>
<th>Unsorted array $A[1..n]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 5 3 0 2 3 0 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frequencies array $C[0..k]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 0 2 3 0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cumulative frequencies array $C[0..k]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 2 4 7 7 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sorted array $B[1..n]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 2 2 3 3 3 5</td>
</tr>
</tbody>
</table>
**CountingSort**

\[ \text{CountingSort}(A[1\ldots n]) \]

- \( k \leftarrow \max(A[1\ldots n]) \)
- Create new array \( B[1\ldots n] \)
- Create new array \( C[0\ldots k] \) and initialize it to 0

  // Find the frequencies of items
  // After this step, \( C[i] \) will contain #elements equal to \( i \)

  for \( j \leftarrow 1 \) to \( n \) do
    \( C[A[j]] \leftarrow C[A[j]] + 1 \)

  // Find the cumulative frequencies of items
  // After this step, \( C[i] \) will contain #elements less than or equal to \( i \)

  for \( i \leftarrow 1 \) to \( k \) do
    \( C[i] \leftarrow C[i] + C[i - 1] \)

  // Get the sorted array in \( B \)

  for \( j \leftarrow n \) downto 1 do
    \( B[C[A[j]]] \leftarrow A[j] \)
    \( C[A[j]] \leftarrow C[A[j]] - 1 \)

  // Copy the sorted array to \( A \)

  for \( j \leftarrow 1 \) to \( n \) do
    \( A[j] \leftarrow B[j] \)
### Solutions → Counting sort

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Algorithm Steps
1. Initialize arrays A, B, and C with initial values.
2. Iterate through the arrays, comparing elements and swapping.
3. Continue until the arrays are sorted.

#### Example
- **Initial state:**
  - A: [2 5 3 0 2 3 0 3]
  - B: [2 2 4 7 7 8]
  - C: [2 2 4 6 7 8]
- **Final state:**
  - A: [2 5 3 0 2 3 0 3]
  - B: [0 0 2 3 3 3]
  - C: [0 2 3 5 7 8]
This algorithm counts the number of occurrences of each element in the input sequence and then uses that information to construct the sorted output. It is often used when the range of input elements is known in advance.

The algorithm works by distributing the input elements into a number of bins/buckets based on their values and then collecting from the bins in order, resulting in a sorted output.

This algorithm is typically used for sorting a large number of elements with a small range of possible values.
CountingSortVariant($A[1\ldots n]$)

$(max, min) \leftarrow \text{MAXMIN}(A[1\ldots n])$

$size \leftarrow max - min + 1$ \hspace{1cm} // size of range $[min, max]$

Create an array $B[1\ldots size] \leftarrow [0\ldots 0]$

// Distribute array $A$ elements to buckets in $B$

for $j \leftarrow 1$ to $n$ do

\hspace{1cm} $i \leftarrow A[j] - min + 1$; $B[i] \leftarrow B[i] + 1$

// Construct the sorted array $A$ based on the bucket array

index $\leftarrow 1$

for $i \leftarrow 1$ to $size$ do

\hspace{1cm} while $B[i] > 0$ do

\hspace{2cm} $A[index] \leftarrow i + min - 1$

\hspace{2cm} index $\leftarrow index + 1$

\hspace{2cm} $B[i] \leftarrow B[i] - 1$

Let $\#\text{buckets} = \max(A[1\ldots n]) - \min(A[1\ldots n])$

$\langle \text{Time, Space} \rangle = \langle \Theta(n + \#\text{buckets}), \Theta(\#\text{buckets}) \rangle$
### Solutions \rightarrow Counting sort variant

<table>
<thead>
<tr>
<th>Index</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 7 9 3 5 3 4 5</td>
<td>2 1 3 0 1 0 1</td>
</tr>
<tr>
<td>2</td>
<td>3 3</td>
<td>1 1 3 0 1 0 1</td>
</tr>
<tr>
<td>3</td>
<td>3 3 4</td>
<td>0 1 3 0 1 0 1</td>
</tr>
<tr>
<td>4</td>
<td>0 1 3 0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0 0 3 0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0 0 3 0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0 0 2 0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0 0 2 0 1 0 1</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{Index} = 1, \ i = 1 \]
\[ A[\text{index}] = i + \text{min} - 1 \]
\[ B[i] -- \]

\[ \text{Index} = 6, \ i = 3 \]
\[ A[\text{index}] = i + \text{min} - 1 \]
\[ B[i] -- \]

\[ \text{Index} = 7, \ i = 5 \]
\[ A[\text{index}] = i + \text{min} - 1 \]
\[ B[i] -- \]

\[ \text{Index} = 8, \ i = 6 \]
\[ A[\text{index}] = i + \text{min} - 1 \]
\[ B[i] -- \]

\[ \text{Index} = 8, \ i = 7 \]
\[ A[\text{index}] = i + \text{min} - 1 \]
\[ B[i] -- \]
1. Sort the numbers based on digits at unit’s place
2. Sort the numbers based on digits at ten’s place
3. Sort the numbers based on digits at hundred’s place
4. Continue the process until you cover all decimal digits
5. By the end, the entire array will be sorted
**RadixSort**

Let's break down the code step by step:

1. **Initialization**
   - Compute `max ← Max(A[1...n])` and set `exp ← 1`.
   - This initializes the maximum value in the array and sets the exponent to 1, which corresponds to the number of digits in the smallest number.

2. **Sorting**
   - **While Loop**: `while exp ≤ max do`.
     - **Array Setup**:
       - Set `C[0...9] ← [0...0]` and `B[1...n] ← [0...0]`.
       - This initializes the count array to 0 for each digit and prepares the output array.
   - **Cumulative Count**:
     - **For Loop**:
       - For each element `i` from 1 to `n` do:
         - Increment `C[index]`.
       - This finds the cumulative frequency of each digit in the array.
     - For `i ← 1 to 9` do:
       - `C[i] ← C[i] + C[i - 1]`.
       - This updates the count array to reflect the cumulative frequency.

3. **Output Population**
   - **For Loop**:
     - For each element `i` from `n` downto `1` do:
       - This rearranges the elements into the `B` array according to their index in the count array, effectively sorting the array.

4. **Completion**
   - After completing the loops, `A[1...n] ← B[1...n]` and `exp ← exp × 10`.

**Time and Space Complexity**

The time complexity is \( \Theta(n \log n) \) due to the loop involving \( \log n \) steps, and the space complexity is \( \Theta(n) \) due to the count array.

\[
\langle \text{Time}, \text{Space} \rangle = \langle \Theta(n \log n), \Theta(n) \rangle
\]
• Sort numbers based on the digits at unit’s place

<table>
<thead>
<tr>
<th></th>
<th>329</th>
<th>457</th>
<th>839</th>
<th>436</th>
<th>720</th>
<th>355</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>355</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>329</th>
<th>457</th>
<th>839</th>
<th>436</th>
<th>720</th>
<th>355</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>720</td>
<td>355</td>
<td>436</td>
<td></td>
<td></td>
<td>839</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>329</th>
<th>457</th>
<th>839</th>
<th>436</th>
<th>720</th>
<th>355</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>720</td>
<td>355</td>
<td>436</td>
<td>329</td>
<td>839</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>329</th>
<th>457</th>
<th>839</th>
<th>436</th>
<th>720</th>
<th>355</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>720</td>
<td>355</td>
<td>436</td>
<td></td>
<td>839</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>329</th>
<th>457</th>
<th>839</th>
<th>436</th>
<th>720</th>
<th>355</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>720</td>
<td>355</td>
<td>436</td>
<td>457</td>
<td></td>
<td>839</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>329</th>
<th>457</th>
<th>839</th>
<th>436</th>
<th>720</th>
<th>355</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>720</td>
<td>355</td>
<td>436</td>
<td></td>
<td>839</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>329</th>
<th>457</th>
<th>839</th>
<th>436</th>
<th>720</th>
<th>355</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>720</td>
<td>355</td>
<td>436</td>
<td>457</td>
<td>839</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>329</th>
<th>457</th>
<th>839</th>
<th>436</th>
<th>720</th>
<th>355</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>720</td>
<td>355</td>
<td>436</td>
<td>457</td>
<td>329</td>
<td>839</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
• Sort numbers based on the digits at ten’s place
• $B$ from the previous iteration will be the $A$ for this iteration.
• Sort numbers based on the digits at **hundred’s place**
• *B* from the previous iteration will be the *A* for this iteration.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>720</td>
<td>329</td>
<td>436</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>457</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>720</td>
<td>329</td>
<td>436</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>457</td>
<td>839</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>720</td>
<td>329</td>
<td>436</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>457</td>
<td>839</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>720</td>
<td>329</td>
<td>436</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>457</td>
<td>839</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>720</td>
<td>329</td>
<td>436</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>457</td>
<td>720</td>
<td>839</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>720</td>
<td>329</td>
<td>436</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>457</td>
<td>839</td>
<td>720</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
Solutions → Bitonic sort

BitonicSort($n$)

[3 7 4 8 6 2 1 5]

Split

[3 7 4 8] [6 2 1 5]

[3 4 7 8] [6 5 2 1]

BitonicMerge

1 2 3 4 5 6 7 8
**BitonicSort**($n$)

```
[3 7 4 8 6 2 1 5]
[3 7 4 8]  [6 2 1 5]
[3 7]  [4 8]  [6 2]
[3 7]  [8 4]  [2 6]
[3 4 7 8]  [6 5 2 1]
[1 2 3 4 5 6 7 8]
```
• Invoke \texttt{BitonicSort}(A[1 \ldots n], ascending)

<table>
<thead>
<tr>
<th>BitonicSort(A[\ell \ldots h], order)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{size} \leftarrow h - \ell + 1</td>
</tr>
<tr>
<td>\textbf{if} \texttt{size} \texttt{&gt; 1} \textbf{then}</td>
</tr>
<tr>
<td>\hspace{1em} \texttt{m} \leftarrow (\ell + h)/2</td>
</tr>
<tr>
<td>\hspace{1em} \texttt{BitonicSort}(A[\ell \ldots m], ascending)</td>
</tr>
<tr>
<td>\hspace{1em} \texttt{BitonicSort}(A[m + 1 \ldots h], descending)</td>
</tr>
<tr>
<td>\hspace{1em} \texttt{BitonicMerge}(A[\ell \ldots h], order)</td>
</tr>
</tbody>
</table>
**BitonicMerge**($A[\ell \ldots h], order$)

**Input:** Array $A[\ell \ldots h]$, ascending/descending order  
**Output:** Bitonic merge the array  

\[ size \leftarrow h - \ell + 1 \]

- **if** $size > 1$ **then**
  \[ m \leftarrow (\ell + h)/2 \]
  \[ \text{COMPARE\&SWAP}(A[\ell \ldots h], order) \]
  \[ \text{BitonicMerge}(A[\ell \ldots m], order) \]
  \[ \text{BitonicMerge}(A[m + 1 \ldots h], order) \]

**Compare\&Swap**($A[\ell \ldots h], order$)

**Input:** Array $A[\ell \ldots h]$, ascending/descending order  
**Output:** Compare items in left & right halves of $A[\ell \ldots h]$ and order them  

\[ size \leftarrow h - \ell + 1 \]

- **for** $i \leftarrow \ell$ **to** $\ell + size/2 - 1$ **do**
  \[ j \leftarrow i + size/2 \]
  **if** (order is ascending and $A[i] > A[j]$) or (order is descending and $A[i] < A[j]$) **then**
  \[ \text{SWAP}(A[i], A[j]) \]
Solutions → Bitonic sort

\[ \langle \text{Time, Space} \rangle = \langle \Theta \left( n \log^2 n \right), \Theta (n) \rangle \]

\[ T(n) = \begin{cases} 
\Theta (1) & \text{if } n = 1, \\
2T(n/2) + T^{\text{merge}}(n/2) & \text{if } n > 1. 
\end{cases} \]

\[ T^{\text{merge}}(n) = \begin{cases} 
\Theta (1) & \text{if } n = 1, \\
2T^{\text{merge}}(n/2) + \Theta (n) & \text{if } n > 1. 
\end{cases} \]
Contributors

Tejas Bhatia, Usha Vudatha, Abiyaz Chowdhury, Taha Kothawala, Ajay Hegde, Sai Sujith Bezawada