Algorithms

(Algorithmic Problem Solving)

Pramod Ganapathi
Department of Computer Science
State University of New York at Stony Brook

December 5, 2023

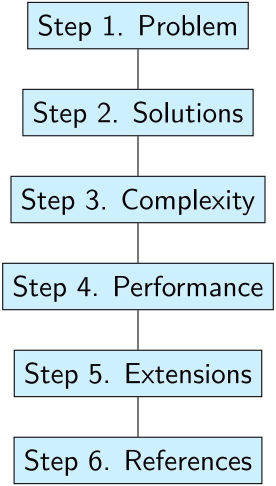
Contents

Contents

Contents

Sorting Algorithms

Algorithmic-problem-solving template



Practical Algorithms

HOME

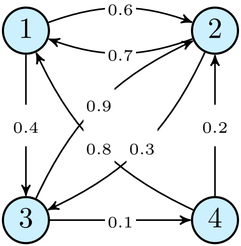
Webpage Ranking

HOME

Problem

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).

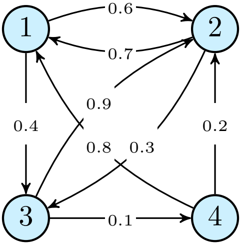
- If a user visits pages of the Internet for a long period of time as per transition probabilities, the distribution stabilizes and this is called stationary/stable distribution (SD)

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

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



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

- 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

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

Modeling page rank using linear algebra

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

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

Modeling page rank using linear algebra

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

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

$$\begin{aligned} P_t &= \left[P_t[1] \; P_t[2] \; \cdots \; P_t[n] \right] \\ &= \left[\sum_{j=1}^{n} \left(P_{t-1}[j] \times T[j,1] \right) \;\; \cdots \;\; \sum_{j=1}^{n} \left(P_{t-1}[j] \times T[j,n] \right) \right] \\ &= P_{t-1} \times T \end{aligned}$$

Modeling page rank using linear algebra

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

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

$$\begin{aligned} P_t &= [P_t[1]\ P_t[2]\ \cdots\ P_t[n]] \\ &= \left[\sum_{j=1}^{n} \left(P_{t-1}[j] \times T[j,1]\right)\ \cdots\ \sum_{j=1}^{n} \left(P_{t-1}[j] \times T[j,n]\right) \right] \\ &= P_{t-1} \times T \end{aligned}$$

$$\boxed{P_t = \begin{cases} \left[\frac{1}{n}\ \frac{1}{n}\ \cdots\ \frac{1}{n}\right] & \text{if } t = 0, \\ P_{t-1} \times T & \text{if } t > 0. \end{cases}}$$

Modeling page rank using linear algebra

Page ranks or stable distribution P is computed as:

$$\boxed{P = P \times T}$$

Questions...

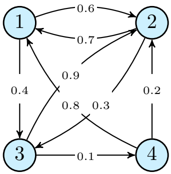
- Do we always get a stable distribution (convergence)?
- If there is a stable distribution, is it always unique?
- Does every initial distribution converge to a stable distribution?

Modeling page rank using linear algebra

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

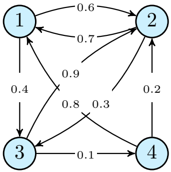


$$T = \begin{bmatrix} 0 & 0.6 & 0.4 & 0 \\ 0.7 & 0 & 0.3 & 0 \\ 0 & 0.9 & 0 & 0.1 \\ 0.8 & 0.2 & 0 & 0 \end{bmatrix}$$

	P_t			
t	$P_t[1]$	$P_t[2]$	$P_t[3]$	$P_t[4]$
0	0.25	0.25	0.25	0.25
1	0.29166667	0.30833333	0.225	0.175
2	0.31305556	0.34305556	0.21972222	0.12416667
3	0.32186111	0.36550926	0.22252778	0.09010185
4	0.32388673	0.38081019	0.22781759	0.06748549
.	...	...	...	...
19	0.310945	0.415423	0.248756	0.0248756

Page ranks $P = [0.310945, 0.415423, 0.248756, 0.0248756]$

Solutions $\rightarrow$ System of linear equations

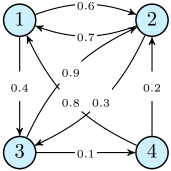


$$T = \begin{bmatrix} 0 & 0.6 & 0.4 & 0 \\ 0.7 & 0 & 0.3 & 0 \\ 0 & 0.9 & 0 & 0.1 \\ 0.8 & 0.2 & 0 & 0 \end{bmatrix}$$

$$\begin{aligned} P_t[1] &= 0.7 \cdot P_t[2] + 0.8 \cdot P_t[4] \\ P_t[2] &= 0.6 \cdot P_t[1] + 0.9 \cdot P_t[3] + 0.2 \cdot P_t[4] \\ P_t[3] &= 0.4 \cdot P_t[1] + 0.3 \cdot P_t[2] \\ P_t[4] &= 0.1 \cdot P_t[3] \\ 1 &= P_t[1] + P_t[2] + P_t[3] + P_t[4] \end{aligned}$$

Page ranks $P = [0.310945, 0.415423, 0.248756, 0.0248756]$

Solutions → Eigenvector

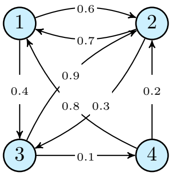


$$T = \begin{bmatrix} 0 & 0.6 & 0.4 & 0 \\ 0.7 & 0 & 0.3 & 0 \\ 0 & 0.9 & 0 & 0.1 \\ 0.8 & 0.2 & 0 & 0 \end{bmatrix}$$

$$[P_t[1]\ P_t[2]\ P_t[3]\ P_t[4]] = [P_t[1]\ P_t[2]\ P_t[3]\ P_t[4]] \times \begin{bmatrix} 0 & 0.6 & 0.4 & 0 \\ 0.7 & 0 & 0.3 & 0 \\ 0 & 0.9 & 0 & 0.1 \\ 0.8 & 0.2 & 0 & 0 \end{bmatrix}$$

$$\implies \begin{bmatrix} P_t[1] \\ P_t[2] \\ P_t[3] \\ P_t[4] \end{bmatrix} = \begin{bmatrix} 0 & 0.7 & 0 & 0.8 \\ 0.6 & 0 & 0.9 & 0.2 \\ 0.4 & 0.3 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \end{bmatrix} \times \begin{bmatrix} P_t[1] \\ P_t[2] \\ P_t[3] \\ P_t[4] \end{bmatrix}$$

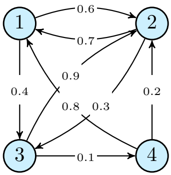
Solutions $\rightarrow$ Eigenvector



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

$$\left(\ P = P \cdot T\ \right) \implies \left(\ \mathsf{col}(P) = T^{\mathsf{transpose}} \cdot \mathsf{col}(P)\ \right)$$

Solutions $\rightarrow$ Eigenvector



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

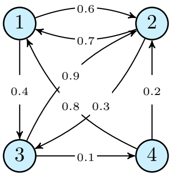
$$(\ P = P \cdot T\ ) \implies (\ \mathsf{col}(P) = T^{\mathsf{transpose}} \cdot \mathsf{col}(P)\ )$$

which is equivalent to the formula

$$\lambda \vec{v} = A \cdot \vec{v}$$

where $\lambda = 1$, $A = T^{\mathsf{transpose}}$, and $\vec{v} = \mathsf{col}(P)$
and $\vec{v}$ is the eigenvector of A corresponding to eigenvalue 1. So,

Solutions $\rightarrow$ Eigenvector



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

$$(\ P = P \cdot T\ ) \implies (\ \mathsf{col}(P) = T^{\mathsf{transpose}} \cdot \mathsf{col}(P)\ )$$

which is equivalent to the formula

$$\lambda \vec{v} = A \cdot \vec{v}$$

where $\lambda = 1$, $A = T^{\mathsf{transpose}}$, and $\vec{v} = \mathsf{col}(P)$
and $\vec{v}$ is the eigenvector of A corresponding to eigenvalue 1. So,

$\boxed{\mathsf{col}(P) \text{ is the eigenvector of } T^{\mathsf{transpose}} \text{ corresponding to eigenvalue } 1}$

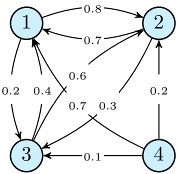
$\boxed{\text{Page ranks } P = [0.310945, 0.415423, 0.248756, 0.0248756]}$

Problems

There can be two types of problems.

- Page ranks can be zero
- Page ranks can be unstable

Problem → Page rank being zero

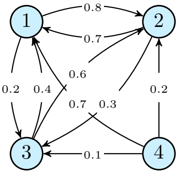


$$T = \begin{bmatrix} 0 & 0.8 & 0.2 & 0 \\ 0.7 & 0 & 0.3 & 0 \\ 0.4 & 0.6 & 0 & 0 \\ 0.7 & 0.2 & 0.1 & 0 \end{bmatrix}$$

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

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

Problem $\rightarrow$ Page rank being zero



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

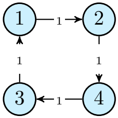
$$P = [0.376147, 0.422018, 0.201835, 0]$$

- 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 $\rightarrow$ Unstable page ranks

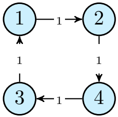


$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

	P_t			
t	$P_t[1]$	$P_t[2]$	$P_t[3]$	$P_t[4]$
0	1	0	0	0
1	0	1	0	0
2	0	0	0	1
3	0	0	1	0
4	1	0	0	0

- How can the ranks of pages be unstable?

Problem → Unstable page ranks



$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

	P_t			
t	$P_t[1]$	$P_t[2]$	$P_t[3]$	$P_t[4]$
0	1	0	0	0
1	0	1	0	0
2	0	0	0	1
3	0	0	1	0
4	1	0	0	0

- 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

Theorem for Markov chains

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

Ideas

- 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

Teleportation factor

- 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:

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

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

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

Teleportation factor

- 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:

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

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

  where S is an $n \times n$ matrix with all 1s
- What does this transformation mean?
  This transformation means that

  $$T[i,j] \text{ will be } \left\{ \begin{array}{ll} \text{incremented} & \text{if } T[i,j] \in [0, 1/n) \\ \text{the same} & \text{if } T[i,j] = 1/n \\ \text{decremented} & \text{if } T[i,j] \in (1/n, 1] \end{array} \right\}$$

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

Solutions

Algorithms

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

Solutions $\rightarrow$ Brute force

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

$\textsc{TransformTransitionMatrix}(T)$
Create the pagerank arrays $P_{\mathsf{old}}[1 \ldots n]$ and $P_{\mathsf{new}}[1 \ldots n]$
$P_{\mathsf{old}}[1 \ldots n] \leftarrow [1/n, 1/n, \ldots, 1/n]$
$P_{\mathsf{new}}[1 \ldots n] \leftarrow P_{\mathsf{old}}[1 \ldots n]$
$maxerror \leftarrow 10^{-6}$; $error \leftarrow \infty$
while $error > maxerror$ **do**
  $P_{\mathsf{new}} \leftarrow P_{\mathsf{old}} \cdot T$
  $error \leftarrow |P_{\mathsf{new}} - P_{\mathsf{old}}|$
  $P_{\mathsf{old}} \leftarrow P_{\mathsf{new}}$
return $P_{\mathsf{new}}[1 \ldots n]$

Solutions → System of equations

$\textsc{SystemOfEquations}(T[1 \ldots n, 1 \ldots n])$

$\textsc{TransformTransitionMatrix}(T)$
Create the pagerank array $P[1 \ldots n]$
$P \leftarrow \textsc{SolveSystemOfEquations}(P = P \cdot T)$

return $P[1 \ldots n]$

Solutions $\rightarrow$ Eigenvector method

$\textsc{EigenvectorMethod}(T[1 \ldots n, 1 \ldots n])$

$\textsc{TransformTransitionMatrix}(T)$
Create the pagerank array $P[1 \ldots n]$
$P \leftarrow \textsc{SolveForFirstEigenvector}(\mathsf{col}(P) = T^{\mathsf{transpose}} \cdot \mathsf{col}(P))$
return $P[1 \ldots n]$

Which is the best algorithm?

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

Theoretically fast algos might not always be the best in practice

References

- GO Page rank video by Reducible
- GO AMS description of page rank
- GO Cornell tutorial for page rank
- GO Brin and Page's paper
- GO Linear algebra behind Google

Stable Marriage HOME

Problem

Problem

- 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.

Problem

- 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).

	Preference		
Guy	1st	2nd	3rd
A	p	q	r
B	q	r	p
C	r	p	q

	Preference		
Gal	1st	2nd	3rd
p	B	C	A
q	C	A	B
r	A	B	C

- 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

Guy	Preference 1st	2nd	3rd
A	p	q	r
B	q	r	p
C	r	p	q

Gal	Preference 1st	2nd	3rd
p	B	C	A
q	C	A	B
r	A	B	C

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

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

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.

Gale-Shapley's algorithm

- 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.

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

A proposes to p and p accepts the proposal
A is p's first offer/proposal

Gale-Shapley's algorithm

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

B proposes to p and p rejects the proposal
p's current partner A is better than B

Gale-Shapley's algorithm

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

B proposes to p and p rejects the proposal
p's current partner A is better than B

Gale-Shapley's algorithm

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

C proposes to q and q accepts the proposal

C is q's first offer/proposal

Gale-Shapley's algorithm

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

D proposes to s and s accepts the proposal
D is s's first offer/proposal

Gale-Shapley's algorithm

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

B proposes to s and s accepts the proposal
B is better than s's current partner D

Gale-Shapley's algorithm

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

D proposes to q and q accepts the proposal
D is better than q's current partner C

Gale-Shapley's algorithm

Guy	Preference			
	1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference			
	1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

C proposes to p and p accepts the proposal
C is better than p's current partner A

Gale-Shapley's algorithm

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

A proposes to q and q rejects the proposal
q's current partner D is better than A

Gale-Shapley's algorithm

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

A proposes to r and r accepts the proposal
A is r's first offer/proposal

Gale-Shapley's algorithm

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

There is no man who is not engaged
Algorithm terminates
Stable matching is achieved
Matching best for men and worst for women

Gale-Shapley's algorithm

Gal	Preference 1st	2nd	3rd	4th
p	D	C	A	B
q	B	D	A	C
r	D	A	B	C
s	C	B	A	D

Guy	Preference 1st	2nd	3rd	4th
A	p	q	r	s
B	p	s	r	q
C	q	p	r	s
D	s	q	r	p

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

```
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 $= \mathcal{O}\left(n^2\right)$
- Space $= \mathcal{O}\left(n\right)$ for notengaged dequeue and engaged map

Gale-Shapley's algorithm

- Stable matching even when #men $>$ #women or #women $>$ #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

- GO Gale-Shapley paper
- GO Gale-Shapley simulation

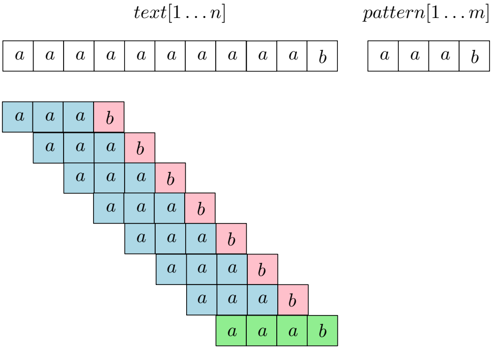
String Matching

HOME

Problem

- Given a text $text[1 \dots n]$ and a pattern $pattern[1 \dots m]$, design an algorithm to find the location of the first occurrence of the pattern in the text.

Solutions → Brute force



Solutions → Brute force

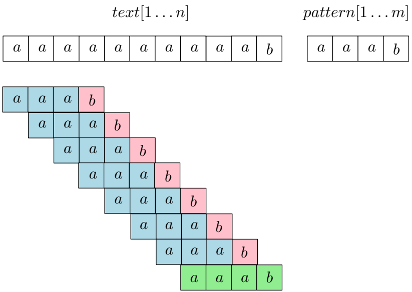
1. Check if the pattern matches the text starting from the 1st index of text.
2. If not, check if the pattern matches with the text starting from the 2nd index of the text.
3. Repeat this process until either the pattern is found or the end of the text is reached (without finding any pattern).

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

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

$$\langle \text{PreprocessTime, MatchTime, Space} \rangle = \langle 0, \mathcal{O}(mn), \Theta(1) \rangle$$

Solutions → Hashing



$\textsc{Hash}(string[1 \dots m], b, p)$

// 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

$hash \leftarrow 0$

for $i \leftarrow 1$ **to** m **do**

| $hash \leftarrow (hash \times b + string[i]) \bmod p$

return $hash$

Time $= \Theta\left(m\right)$

Solutions → Hashing

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).

```
StringMatching-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
```

$$\langle \text{PreprocessTime, MatchTime, Space} \rangle = \langle \Theta(m), \mathcal{O}(mn), \Theta(1) \rangle$$

Solutions → RabinKarp (rolling hash)

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

```
p ← a good prime                                   // e.g.: 101
b ← size of ASCII set                              // i.e., 256
h ← b^(m−1) mod p          // highest term in the polynomial hash
patternhash ← Hash(pattern, b, p)
texthash ← Hash(text[1 ... m], b, p)

for i ← 1 to n − m + 1 do
    if texthash = patternhash and text[i ... (i + m − 1)] = pattern then
        return i

    // Rolling hash: Compute hash value of the next text window using
       the current text window in Θ(1) time
    if i ≠ n − m + 1 then
        texthash ← RollingHash(texthash, text[i ... i + m])
return −1
```

Solutions $\to$ RabinKarp (rolling hash)

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

$texthash \leftarrow ((texthash - string[1] \times h) \times b + string[m']) \bmod p$
return $texthash$

$$\text{Time} = \Theta\left(1\right)$$

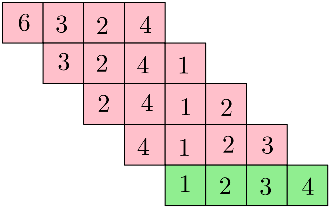
Solutions → RabinKarp (rolling hash)

$text[1 \dots n]$

6	3	2	4	1	2	3	4	2	6	8	9	7

$pattern[1 \dots m]$

1	2	3	4



$$1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0 = 1234$$

$$6 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0 = 6324$$

$$(6324 - 6 \cdot 10^3) \cdot 10 + 1 = 3241$$

$$(3241 - 3 \cdot 10^3) \cdot 10 + 2 = 2412$$

$$(2412 - 2 \cdot 10^3) \cdot 10 + 3 = 4123$$

$$(4123 - 4 \cdot 10^3) \cdot 10 + 4 = 1234$$

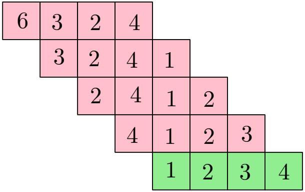
Solutions → RabinKarp (rolling hash)

$text[1 \ldots n]$

6	3	2	4	1	2	3	4	2	6	8	9	7

$pattern[1 \ldots m]$

1	2	3	4



$$(1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0) \bmod 31 = 2$$

$$(6 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0) \bmod 31 = 8$$

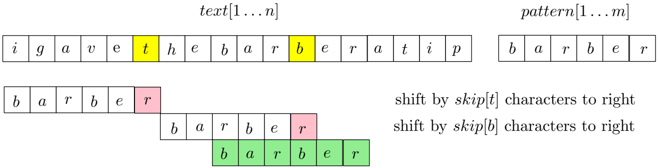
$$((8 - 6 \cdot 10^3 \bmod 31) \cdot 10 + 1) \bmod 31 = 25$$

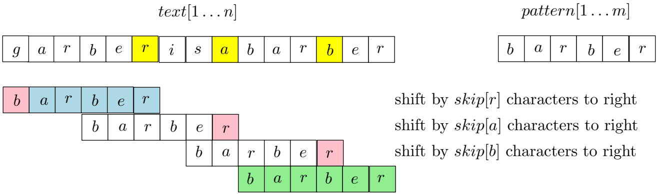
$$((25 - 3 \cdot 10^3 \bmod 31) \cdot 10 + 2) \bmod 31 = 2$$

$$((2 - 2 \cdot 10^3 \bmod 31) \cdot 10 + 3) \bmod 31 = 8$$

$$((8 - 4 \cdot 10^3 \bmod 31) \cdot 10 + 4) \bmod 31 = 2$$

Solutions → Boyer-Moore-Horspool





character	b	a	r	e	other
$skip$[character]	2	4	3	1	6

$$skip[\alpha] = \begin{cases} \text{distance from the end of the pattern of } \alpha\text{'s last occurrence} & \text{if } \alpha \neq pattern[m] \\ \text{distance from the end of the pattern of } \alpha\text{'s last but one occurrence} & \text{if } \alpha = pattern[m] \end{cases}$$

Solutions → Boyer-Moore-Horspool

```
StringMatching-BMH(text[1...n], pattern[1...m])

skip[0...255] ← ConstructSkipTable(pattern)
i ← m
while i ≤ n do
|  if text[(i − m + 1)...i] = pattern comparing from right to left then
|  |  return i − m + 1
|  else
|  |  i ← i + skip[text[i]]
return −1
```

```
ConstructSkipTable(pattern[1...m])

// Initialize the skip table of ASCII characters to m
skip[0...255] ← [m...m]
for i ← 1 to m − 1 do
|  skip[pattern[i]] ← m − i
return skip[0...255]
```

$$\langle \text{PreprocessTime, MatchTime, Space} \rangle = \langle \Theta\left(m + |\Sigma|\right), \mathcal{O}\left(mn\right), \Theta\left(|\Sigma|\right) \rangle$$

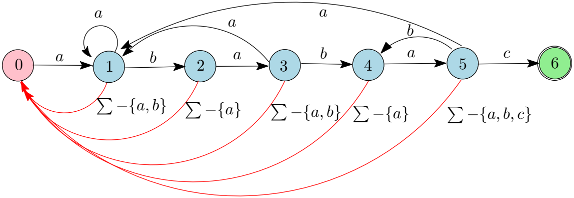
Solutions → Aho-Corasick

$text[1\dots n]$

a	a	b	a	c	a	a	b	a	b	a	c	a	a

$pattern[1\dots m]$

a	b	a	b	a	c



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	–	–	–	–

Solutions $\rightarrow$ Aho-Corasick

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

$transitiontable[0 \ldots m, 0 \ldots 255] \leftarrow \textsc{BuildTransitionTable}(pattern)$
$state \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
    $state \leftarrow transitiontable[state, text[i]]$
    if $state = m$ **then**
        return $i - m + 1$
return -1

$$\langle \text{PreprocessTime, MatchTime, Space} \rangle = \langle \Theta(m), \mathcal{O}(n), \Theta(m) \rangle$$

Solutions $\rightarrow$ 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
X[0...m] ← [0...0]; len ← 0
for i ← 1 to m do
    if i < m and pattern[i+1] = pattern[len+1] then
    |   len ← len + 1; X[i] ← len
    else if len ≠ 0 then
    |   len ← X[len − 1]; i ← i − 1
    else
    |   X[i] ← 0
// 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
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(m\Sigma), \Theta(m\Sigma) \rangle$$

Complexity

Algorithm	Preprocess time	Matching time	Space
Brute force	$-$	$\mathcal{O}(mn)$	$\Theta(1)$
Rabin Karp	$\Theta(m)$	$\mathcal{O}(mn)$	$\Theta(1)$
Horspool	$\Theta(m + \|\Sigma\|)$	$\mathcal{O}(mn)$	$\Theta(\|\Sigma\|)$
Aho-Corasick	$\Omega(m\|\Sigma\|)$	$\mathcal{O}(n)$	$\Theta(m\|\Sigma\|)$

Bit Tricks HOME

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

$$x = \boxed{+}\ x_{w-1} 2^{w-1} + x_{w-2} 2^{w-2} + \cdots + x_0 2^0$$

- Signed integer value stored in x is

$$x = \boxed{-}\ x_{w-1} 2^{w-1} + x_{w-2} 2^{w-2} + \cdots + x_0 2^0$$

- 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

$$A = 0B10110011$$
$$B = 0B01101001$$

Operator	Description	Operation
$\&$	AND	$A \ \& \ B = 0B00100001$
$\vert$	OR	$A \mid B = 0B11111011$
$\oplus$	XOR	$A \oplus B = 0B11011010$
$\sim$	NOT	$\sim A = 0B01001100$
$>>$	shift right	$A >> 1 = 0B01011001$
	shift right	$A >> 2 = 0B00101100$
$<<$	shift left	$A >> 1 = 0B01100110$
	shift left	$A >> 2 = 0B11001100$

Complementation

Problem

- Take the complement of a word x

$$\boxed{\text{1's complement} = \sim x}$$

$$\boxed{\text{2's complement} = \sim x + 1 = -x}$$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
$\sim x$	0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 0
$\sim x + 1$	0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1

Odd or even

Problem

- Check if an integer is odd or even

$$\boxed{A = x \ \& \ 1}$$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
$A = x \ \& \ 1$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Extract a bit

Problem

- Extract the kth bit in a word x.

$$\boxed{mask = 1 << k}$$

$$\boxed{A = (x \,\&\, mask) >> k}$$

Term	Bits
x	1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 1
$mask = 1 << 7$	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
$x \,\&\, mask$	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
$A = (x \,\&\, mask) >> 7$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Set a bit

Problem

- Set the kth bit in a word x.

$$\boxed{mask = 1 << k}$$
$$\boxed{A = x \mid mask}$$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
$mask = 1 << 7$	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
$A = x \mid mask$	1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 1

Clear a bit

Problem

- Clear the kth bit in a word x.

$$\boxed{mask = \sim (1 << k)}$$

$$\boxed{A = x \ \& \ mask}$$

Term	Bits
x	1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 1
$1 << 7$	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
$mask = \sim (1 << 7)$	1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
$A = x \ \& \ mask$	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1

Toggle a bit

Problem

- Toggle the kth bit in a word x.

$$\boxed{mask = 1 << k}$$
$$\boxed{A = x \oplus mask}$$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
$mask = 1 << 7$	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
$A = x \oplus mask$	1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 1

Extract a bit field

Problem

- Extract a bit field in a word x.

$$\boxed{A = (x \,\&\, mask) >> shift}$$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
$mask$	0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
$x \,\&\, mask$	0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0
$(x \,\&\, mask) >> shift$	0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

Set a bit field

Problem

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

$$A = (x \ \& \ \sim mask) \mid ((y << shift) \ \& \ mask)$$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
$mask$	0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
$\sim mask$	1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1
$A_1 = x \ \& \ \sim mask$	1 0 1 1 1 0 0 0 0 1 1 0 1 1 0 1
y	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
$A_2 = (y << 7) \ \& \ mask$	0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
$A = A_1 \mid A_2$	1 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1

Swap

Problem

- Swap two integers x and y.

Using temporary variables: $\boxed{t = x;\ x = y;\ y = t;}$

No temporary variables: $\boxed{x = x \oplus y;\ y = x \oplus y;\ x = x \oplus y;}$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
y	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
$x = x \oplus y$	1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0
$y = x \oplus y$	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
$x = x \oplus y$	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

Swap

Problem

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

Core idea:

$$\boxed{a \oplus a = 0} \implies \boxed{b \oplus a \oplus a = a \oplus b \oplus a = a \oplus a \oplus b = b}$$

- How do you apply this idea to this algorithm?
  Let's keep the x and y variables unchanged

$$a = x \oplus y$$
$$b = a \oplus y = x \oplus y \oplus y = x$$
$$c = a \oplus b = x \oplus y \oplus x = y$$

- 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.

$$\boxed{A = (x \oplus y) < 0}$$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
y	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
$(x \oplus y)$	1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0
$A = (x \oplus y) \ < \ 0$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
y	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
$(x \oplus y)$	0 1 0 0 0 0 1 0 0 1 1 0 1 1 0 1
$A = (x \oplus y) \ < \ 0$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sign of an integer

Problem

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

$$\boxed{A = (x > 0) - (x < 0)}$$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
$(x > 0)$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
$(x < 0)$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
$A = (x > 0) - (x < 0)$	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Term	Bits
x	0 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
$A = (x > 0) - (x < 0)$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Term	Bits
x	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
$A = (x > 0) - (x < 0)$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Check if number is a power of 2

Problem

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

$$\boxed{A = x \ \wedge \ !(x \ \& \ (x-1))}$$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 0
$(x-1)$	1 0 1 1 1 1 0 1 0 1 1 0 1 0 1 1
$(x \ \& \ (x-1))$	1 0 1 1 1 1 0 1 0 1 1 0 1 0 0 0
$!(x \ \& \ (x-1))$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
$A = x \ \wedge \ !(x \ \& \ (x-1))$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Term	Bits
x	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
$(x-1)$	0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
$(x \ \& \ (x-1))$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
$!(x \ \& \ (x-1))$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
$A = x \ \wedge \ \sim (x \ \& \ (x-1))$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Create the table for $x = 0$.

Minimum of two integers

Problem

- Find the minimum of two integers x and y.

$$\boxed{A = y \oplus ((x \oplus y) \;\&\; -(x < y))}$$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
y	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
$(x < y)$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
$-(x < y)$	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
$(x \oplus y)$	1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0
$(x \oplus y) \;\&\; -(x < y)$	1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0
$A = y \oplus ((x \oplus y) \;\&\; -(x < y))$	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1

- $x < y \implies -(x < y) = -1 = 1 \ldots 1 \implies A = y \oplus (x \oplus y) = x$
- $x \geq y \implies -(x < y) = 0 = 0 \ldots 0 \implies A = y \oplus 0 = y$

Maximum of two integers

Problem

- Find the maximum of two integers x and y.

$$\boxed{A = x \oplus ((x \oplus y)\ \&\ -(x < y))}$$

Term	Bits
x	1 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1
y	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
$(x < y)$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
$-(x < y)$	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
$(x \oplus y)$	1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0
$(x \oplus y)\ \&\ -(x < y)$	1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0
$A = x \oplus ((x \oplus y)\ \&\ -(x < y))$	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

- $x < y \implies -(x < y) = -1 = 1 \ldots 1 \implies A = x \oplus (x \oplus y) = y$
- $x \geq y \implies -(x < y) = 0 = 0 \ldots 0 \implies A = x \oplus 0 = x$

Count set bits in unsigned int

Problem

- Count set bits in unsigned int x.

$$\boxed{\text{for } (A = 0; x; A{+}{+})\ x \leftarrow x \ \&\ (x - 1)}$$

Count	Term	Bits
$A = 0$	x	1 0 0 0 1 1 0 1 0 1 1 0 0 1 0 1
$A = 1$	$x = x \ \&\ (x - 1)$	1 0 0 0 1 1 0 1 0 1 1 0 0 1 0 0
$A = 2$	$x = x \ \&\ (x - 1)$	1 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0
$A = 3$	$x = x \ \&\ (x - 1)$	1 0 0 0 1 1 0 1 0 1 0 0 0 0 0 0
$A = 4$	$x = x \ \&\ (x - 1)$	1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0
$A = 5$	$x = x \ \&\ (x - 1)$	1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
$A = 6$	$x = x \ \&\ (x - 1)$	1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
$A = 7$	$x = x \ \&\ (x - 1)$	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
$A = 8$	$x = x \ \&\ (x - 1)$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

$\lceil \log_2 x \rceil$ for an unsigned int x

Problem

- Compute $\lceil \log_2 x \rceil$ for an unsigned int x.

$$\text{for } (A = 0; x >>= 1; A{+}{+}) \;;$$

Log	Term	Bits
$A = 0$	x	0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1
$A = 1$	$x >>= 1$	0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0
$A = 2$	$x >>= 1$	0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1
$A = 3$	$x >>= 1$	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
$A = 4$	$x >>= 1$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
$A = 5$	$x >>= 1$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
$A = 6$	$x >>= 1$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

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.

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

Term	Bits
x	0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
$x--$	0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1
$x\| = x >> 1$	0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1
$x\| = x >> 2$	0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1
$x\| = x >> 4$	0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1
$x\| = x >> 8$	0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
$x\| = x >> 16$	0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
$x\| = x >> 32$	0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
$x++$	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

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

Polynomial Multiplication HOME

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

$$\boxed{C(x)} = \boxed{A(x)} \times \boxed{B(x)} \qquad \text{where}$$

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

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

$$C(x) = c_0 + c_1 x^1 + \cdots + c_{2n-2} x^{2n-2}$$

Step 2. Subproblem

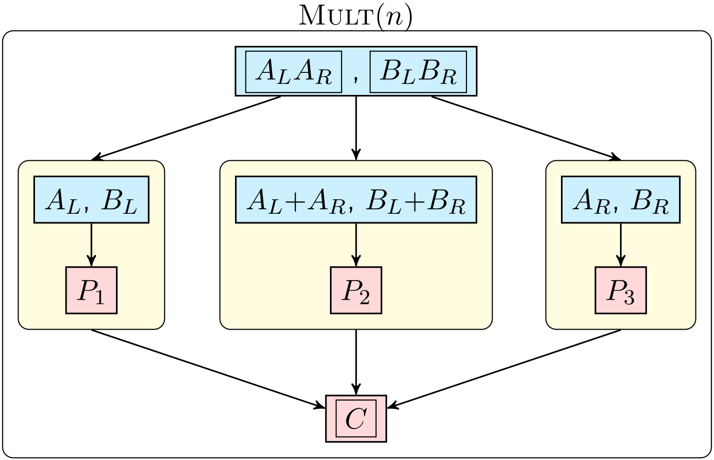
$\textsc{Mult}(A[\ell..h], B[\ell..h])$ = Multiply two $(h-\ell)$ degree polynomials $A[\ell..h]$ and $B[\ell..h]$.

Compute $\textsc{Mult}(A[0..n-1], B[0..n-1])$.

Step 3. Core idea

$$
\begin{aligned}
A \times B &= (A_L A_R) \times (B_L B_R) \\
&= (A_L + A_R \cdot x^{n/2}) \times (B_L + B_R \cdot x^{n/2}) \\
&= (A_L \times B_L) + (A_L \times B_R + A_R \times B_L) \cdot x^{n/2} \\
&\qquad + (A_R \times B_R) \cdot x^n \\
&= (A_L \times B_L) \\
&\qquad + \left(\begin{array}{c} (A_L + A_R) \times (B_L + B_R) \\ -(A_L \times B_L) - (A_R \times B_R) \end{array} \right) \cdot x^{n/2} \\
&\qquad + (A_R \times B_R) \cdot x^n
\end{aligned}
$$

Step 3. Core idea



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 $A(x) \cdot B(x)$:

$$\begin{aligned}
&[-6, 11, -6, 1] \times [-120, 74, -15, 1] \\
&= ([-6, 11] + [-6, 1]x^2) \times ([-120, 74] + [-15, 1]x^2) \\
&= [-6, 11] \times [-120, 74] \\
&\qquad + ([-6, 11] \times [-15, 1] + [-6, 1] \times [-120, 74])x^2 \\
&\qquad + ([-6, 1] \times [-15, 1])x^4 \\
&= [-6, 11] \times [-120, 74] + \\
&\qquad + \left(\begin{array}{c} ([-6, 11] + [-6, 1]) \times ([-120, 74] + [-15, 1]) \\ -([-6, 11] \times [-120, 74]) - ([-6, 1] \times [-15, 1]) \end{array} \right) \cdot x^2 \\
&\qquad + ([-6, 1] \times [-15, 1])x^4
\end{aligned}$$

Step 5. Algorithm

$\textsc{KaratsubaProduct}(A[\ell \ldots h], B[\ell \ldots h])$

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

Output: Product of polynomials A and B

if $\ell = h$ **then** **return** $A[\ell] \times B[\ell]$

$mid \leftarrow \lfloor (h + \ell)/2 \rfloor$; $n \leftarrow h - \ell + 1$

$A_L \leftarrow A[\ell \ldots mid]$, $A_R \leftarrow A[mid + 1 \ldots h]$

$B_L \leftarrow B[\ell \ldots mid]$, $B_R \leftarrow B[mid + 1 \ldots h]$

parallel: $P_1 \leftarrow \textsc{KaratsubaProduct}(A_L, B_L)$

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

$P_3 \leftarrow \textsc{KaratsubaProduct}(A_R, B_R)$

return $(P_1 + (P_2 - P_1 - P_3) \cdot x^{n/2} + P_3 \cdot x^n)$

Step 6. Complexity

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

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

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

$$\text{Cache } Q(n) = \begin{cases} \mathcal{O}(M/B) & \text{if } n \leq \gamma M, \\ 3Q(n/2) + \Theta(n/B) & \text{if } n > \gamma M. \end{cases} \in \mathcal{O}\left(\frac{n^{\log_2 3}}{MB}\right)$$

Polynomial representation

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

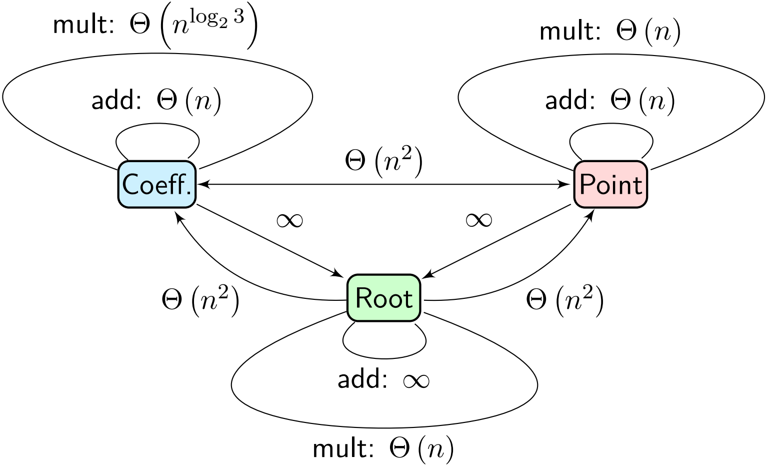
Polynomial representation

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

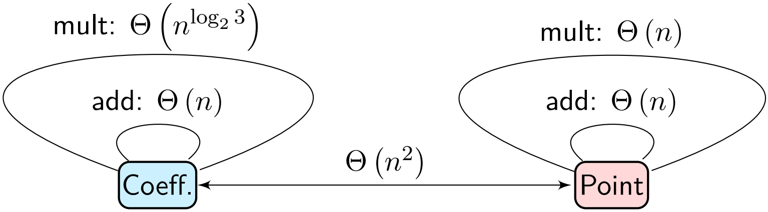
2. Root representation
- 3-degree polynomial can be represented using 3 roots
- $A(x) = 1(x-1)(x-2)(x-3)$
- $A(x) = [1, \{1, 2, 3\}]$ ▷ set of roots

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



Operations on polynomials



- 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

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

$\boxed{X = \{x_0, x_1, \ldots, x_{n-1}\}}$ and $\boxed{Y = V_X A}$

- Evaluation.
  Convert coefficient to point representation.
  A is known, X is chosen, Y is computed.
  Y can be computed in $\Theta\left(n^2\right)$ time using Horner's formula.
- Interpolation.
  Convert point to coefficient representation.
  X and Y are known, A is computed.
  A can be computed in $\Theta\left(n^2\right)$ time using Lagrange's formula.

Evaluation and interpolation

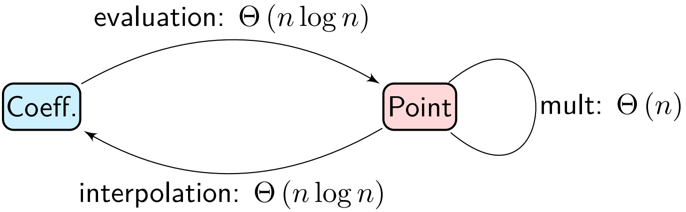
Problem

- Can we perform evaluation & interpolation better than $\Theta\left(n^2\right)$?

Great idea

- We can perform evaluation and interpolation in $\Theta\left(n \log n\right)$ using roots of 1 and divide-and-conquer.
- Evaluation of $(n-1)$-degree polynomial $A(x)$ at n roots of unity can be done in $\Theta\left(n \log n\right)$ using divide-and-conquer. Interpolation of n roots of unity to an $(n-1)$-degree polynomial can be done in $\Theta\left(n \log n\right)$ using divide-and-conquer.

Step 3. Core idea



Step 3. Core idea

- Core idea.

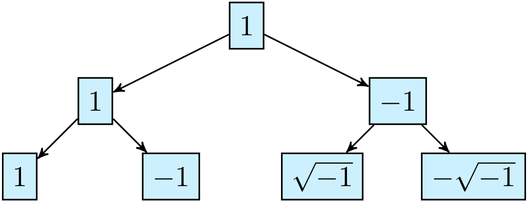
$$\boxed{\begin{aligned} A(x) &= A^{\text{even}}(x^2) + xA^{\text{odd}}(x^2) \\ A(-x) &= A^{\text{even}}(x^2) - xA^{\text{odd}}(x^2) \end{aligned}}$$

- Example.

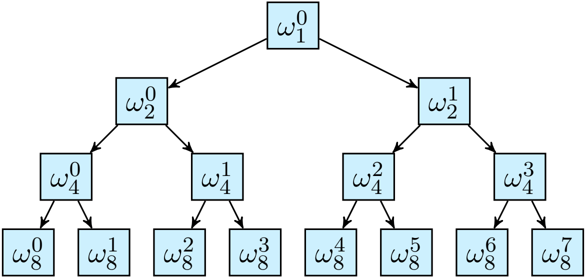
$$\begin{aligned} 7 + 3x + 2x^2 + 6x^3 &= (7 + 2x^2) + x(3 + 6x^2) \\ 7 + 3(-x) + 2(-x)^2 + 6(-x)^3 &= (7 + 2x^2) - x(3 + 6x^2) \end{aligned}$$

- 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 $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$, where $\omega_n^k = e^{k(2\pi i)/n} = \cos(k(2\pi/n)) + i\sin(k(2\pi/n))$ and $i = \sqrt{-1}$.



Step 3. Core idea

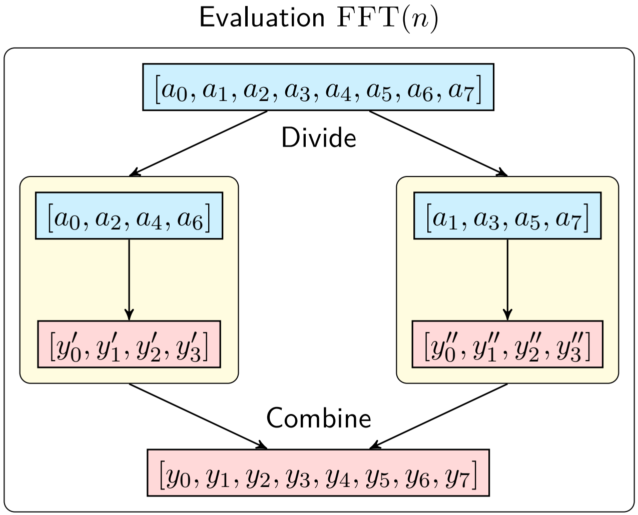
- $\Theta(n \log n)$ evaluation: Use $\boxed{X = \{\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}\}}$.

$$\begin{bmatrix} A(\omega_n^1) \\ A(\omega_n^1) \\ A(\omega_n^2) \\ \vdots \\ A(\omega_n^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & \omega_n^0 & (\omega_n^0)^2 & \cdots & (\omega_n^0)^{n-1} \\ 1 & \omega_n^1 & (\omega_n^1)^2 & \cdots & (\omega_n^1)^{n-1} \\ 1 & \omega_n^2 & (\omega_n^2)^2 & \cdots & (\omega_n^2)^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & (\omega_n^{n-1})^2 & \cdots & (\omega_n^{n-1})^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

- $\Theta(n \log n)$ interpolation: Use $\boxed{X = \frac{1}{n}\{\omega_n^{-0}, \omega_n^{-1}, \ldots, \omega_n^{-(n-1)}\}}$.

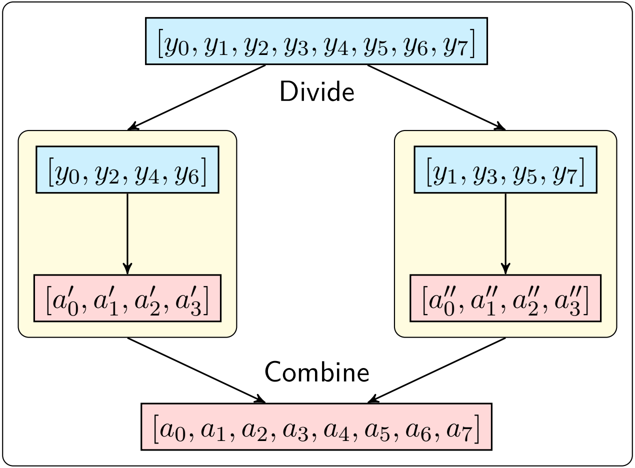
$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & \omega_n^{-0} & (\omega_n^{-0})^2 & \cdots & (\omega_n^{-0})^{n-1} \\ 1 & \omega_n^{-1} & (\omega_n^{-1})^2 & \cdots & (\omega_n^{-1})^{n-1} \\ 1 & \omega_n^{-2} & (\omega_n^{-2})^2 & \cdots & (\omega_n^{-2})^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{-(n-1)} & (\omega_n^{-(n-1)})^2 & \cdots & (\omega_n^{-(n-1)})^{n-1} \end{bmatrix} \begin{bmatrix} A(\omega_n^1) \\ A(\omega_n^1) \\ A(\omega_n^2) \\ \vdots \\ A(\omega_n^{n-1}) \end{bmatrix}$$

Step 3. Core idea



Step 3. Core idea

Interpolation $\textsc{InverseFFT}(n)$



Step 4. Example (Evaluation)

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

$$[a_0, a_1] = [7, 2] \qquad [a_0, a_1] = [3, 6]$$

$$a_0 = 7 \qquad a_0 = 2 \qquad a_0 = 3 \qquad a_0 = 6$$

$$y_0 = 7 \qquad y_0 = 2 \qquad y_0 = 3 \qquad y_0 = 6$$

$$\begin{aligned} y_0 &= 7 + 2\omega_2^0 \\ y_1 &= 7 - 2\omega_2^0 \end{aligned} \qquad \begin{aligned} y_0 &= 3 + 6\omega_2^0 \\ y_1 &= 3 - 6\omega_2^0 \end{aligned}$$

$$\begin{aligned}
y_0 &= (7 + 2\omega_2^0) + (3 + 6\omega_2^0)\omega_4^0 &&= 7 + 3\omega_4^0 + 2(\omega_4^0)^2 + 6(\omega_4^0)^3 \\
y_1 &= (7 - 2\omega_2^0) + (3 - 6\omega_2^0)\omega_4^1 &&= 7 + 3\omega_4^1 + 2(\omega_4^1)^2 + 6(\omega_4^1)^3 \\
y_2 &= (7 + 2\omega_2^0) - (3 + 6\omega_2^0)\omega_4^0 &&= 7 + 3\omega_4^2 + 2(\omega_4^2)^2 + 6(\omega_4^2)^3 \\
y_3 &= (7 - 2\omega_2^0) - (3 - 6\omega_2^0)\omega_4^1 &&= 7 + 3\omega_4^3 + 2(\omega_4^3)^2 + 6(\omega_4^3)^3
\end{aligned}$$

Step 5. Algorithm

$\mathrm{FFT}([a_0, a_1, \ldots, a_{n-1}])$ ▷ Evaluation

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: $Y^{\text{even}} \leftarrow \mathrm{FFT}(A^{\text{even}})$
$Y^{\text{odd}} \leftarrow \mathrm{FFT}(A^{\text{odd}})$
// [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

$\textsc{InverseFFT}([y_0, y_1, \ldots, y_{n-1}])$ ▷ Interpolation

Input: Point values vector Y for X values $[\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}]$
Output: Coefficients of polynomial $A(x)$: $[a_0, a_1, \ldots, a_{n-1}]$

```
if n = 1 then return y_0
ω_n ← (1/n)e^{−2πi/n}
ω ← 1
// [Stage 1. Divide] ....................................
Y^even ← [y_0, y_2, ..., y_{n−2}]
Y^odd ← [y_1, y_3, ..., y_{n−1}]
// [Stage 2. Conquer] ...................................
parallel: A^even ← InverseFFT(Y^even)
          A^odd ← InverseFFT(Y^odd)
// [Stage 3. Combine] ...................................
for k ← 0 to n/2 − 1 do
    a_k ← A_k^even + ω A_k^odd
    a_{n/2+k} ← A_k^even − ω A_k^odd
    ω ← ω ω_n
return [a_0, a_1, ..., a_{n−1}]
```

Step 5. Algorithm

$\textsc{CooleyTukeyProduct}(A(x), B(x))$

Input: Polynomials $A(x)$ and $B(x)$ of same degree
Output: Polynomial product $C(x) = A(x) \times B(x)$

$[a_0, a_1, \ldots, a_{n-1}] \leftarrow \textsc{Coefficients}(A(x))$
$[b_0, b_1, \ldots, b_{n-1}] \leftarrow \textsc{Coefficients}(B(x))$
// [Stage 1. Add high-order coefficients] . . . .
$[a_n, a_{n+1}, \ldots, a_{2n-1}] \leftarrow [0, 0, \ldots, 0]$
$[b_n, b_{n+1}, \ldots, b_{2n-1}] \leftarrow [0, 0, \ldots, 0]$
// [Stage 2. Evaluate] . . . .
parallel: $[y_0^A, y_1^A, \ldots, y_{2n-1}^A] \leftarrow \mathrm{FFT}([a_0, a_1, \ldots, a_{2n-1}])$
$[y_0^B, y_1^B, \ldots, y_{2n-1}^B] \leftarrow \mathrm{FFT}([b_0, b_1, \ldots, b_{2n-1}])$
// [Stage 3. Pointwise multiply] . . . .
parallel: **for** $k \leftarrow 0$ **to** $2n - 1$ **do**
| $y_k^C \leftarrow y_k^A \times y_k^B$
// [Stage 4. Interpolate] . . . .
$[c_0, c_1, \ldots, c_{2n-1}] \leftarrow \textsc{Inverse}\mathrm{FFT}([y_0^C, y_1^C, \ldots, y_{2n-1}^C])$
$C(x) \leftarrow [c_0, c_1, \ldots, c_{2n-1}]$

return $C(x)$

Step 6. Complexity

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

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

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

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

Probabilistic Algorithms

Primality HOME

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

Solutions $\rightarrow$ Naive algorithm

- If n is divisible by any number in the range $[2, n-1]$, then n is composite, else, n is prime

```
PRIMALITY-NAIVEALGORITHM(n)

for i ← 2 to n − 1 do
    if n mod i = 0 then
        return composite
return prime
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n), \Theta(1) \rangle$$

Solutions $\rightarrow$ School algorithm

- 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

Primality-SchoolAlgorithm(n)

for $i \leftarrow 2$ **to** $\lfloor\sqrt{n}\rfloor$ **do**
  if $n \bmod i = 0$ **then**
    return composite
return prime

$$\langle\text{Time, Space}\rangle = \langle\mathcal{O}\left(\sqrt{n}\right), \Theta\left(1\right)\rangle$$

Solutions $\rightarrow$ 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 (⌊√n⌋ − 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
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(\sqrt{n}\right), \Theta\left(1\right) \rangle$$

Solutions → Sieve of Eratosthenes

PRIMALITY-SIEVEOFERATOSTHENES(n)

```
last ← ⌊√n⌋
Create a Boolean array P[2 ... last] to indicate prime numbers
for i ← 2 to last do
|   P[i] ← true
for j ← 2 to last do
|   if P[j] = true then
|   |   for k ← 2 to ⌊last/j⌋ do
|   |   |   i ← j × k
|   |   |   P[i] ← false
|   |   if n mod j = 0 then
|   |   |   return composite
return prime
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(\sqrt{n}\log\log n\right), \Theta\left(\sqrt{n}\right)\rangle$$

Solutions → Wilson's theorem

- Wilson's theorem: A positive integer $n > 1$ is prime iff $((n-1)! + 1) \bmod n = 0$

n	$(n-1)!$	$((n-1)! + 1) \bmod n$	Is Prime?
2	1	0	✓
3	2	0	✓
4	6	2	✗
5	24	0	✓
6	120	1	✗
7	720	0	✓
8	5040	1	✗
9	40320	1	✗
10	362880	1	✗
11	3628800	0	✓
12	39916800	1	✗
13	479001600	0	✓

- Wilson's theorem: A positive integer $n > 1$ is prime iff $((n-1)! + 1) \bmod n = 0$

PRIMALITY-WILSONTHEOREM(n)

$factorial \leftarrow 1$
for $i \leftarrow 2$ **to** $n-1$ **do**
| $factorial \leftarrow (factorial \times i) \bmod n$

if $(factorial + 1) = n$ **then** **return** prime
return composite

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Solutions $\rightarrow$ Fermat's theorem

- $n \geq 4$ is prime iff for all $a \in [2, n-2]$, we have $(a^{n-1} - 1) \bmod n = 0$.

$n : a$	2	3	4	5	6	7	8	9	10	11	12	13
4	3											
5	0	0										
6	1	2	3									
7	0	0	0	0								
8	7	2	7	4	7							
9	3	8	6	6	8	3						
10	1	2	3	4	5	6	7					
11	0	0	0	0	0	0	0	0				
12	7	2	3	4	11	6	7	8	3			
13	0	0	0	0	0	0	0	0	0	0		
14	1	2	3	4	5	6	7	8	9	10	11	
15	3	8	0	9	5	3	3	5	9	0	8	3

Solutions → Fermat's theorem

```
PRIMALITY-FERMATTHEOREM(n)

if n = 2 or n = 3 then  return prime
for a ← 2 to n − 2 do
    // If (a^(n−1) − 1) mod n ≠ 0, then n is definitely composite
    if POWER(a, n − 1, n) ≠ 1 then
        return composite
return prime
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(n \log n\right), \Theta\left(1\right) \rangle$$

Power function using repeated squaring

```
POWER(a, b, c)
Output: Computes (a^b) mod c in Θ(log b) time
result ← 1
while b > 0 do
    if b mod 2 = 1 then  result ← (result × a) mod c
    b = b/2; a = (a × a) mod c
return result
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(\log b), \Theta(1) \rangle$$

Solutions $\rightarrow$ Fermat's test

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

Bad news.

- If a cell in the nth 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 $(a^{n-1} - 1) \bmod n = 0$, 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

```
PRIMALITY-FERMATTEST(n)

if n = 2 or n = 3 then  return prime

 // More trials increases the probability of success
for count ← 1 to #trials do
|   a ← RandomNumber({2, 3, 4, ..., n − 2})
|    // If (a^(n−1) − 1) mod n ≠ 0, then n is definitely composite
|   if POWER(a, n − 1, n) ≠ 1 then
|   |   return composite

return prime                                    // may or may not be prime
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(\#trials \cdot \log n\right), \Theta\left(1\right) \rangle$$

Solutions $\rightarrow$ Miller's theorem

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 $i \in [0, k-1]$.

Solutions $\rightarrow$ Miller's theorem

For odd integer $n > 1$, $n - 1 = 2^k m$, where $k \geq 1$ and m is odd

$$
\begin{aligned}
(x^{2^k m} - 1) &= ((x^{2^{k-1} m})^2 - 1) \\
&= (x^{2^{k-1} m} - 1)(x^{2^{k-1} m} + 1) \\
&= (x^{2^{k-2} m} - 1)(x^{2^{k-2} m} - 1)(x^{2^{k-1} m} + 1) \\
& \dots \\
&= (x^m - 1)(x^m + 1)(x^{2m} + 1)(x^{4m} + 1) \cdots (x^{2^{k-1} m} + 1)
\end{aligned}
$$

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

$$(a^m - 1)(a^m + 1)(a^{2m} + 1)(a^{4m} + 1) \cdots (a^{2^{k-1} m} + 1) \equiv 0 \bmod n$$

When n is odd prime, one of these factors must be $0 \bmod n$, so

$a^m \equiv 1 \bmod n$ or $a^{2^i m} \equiv -1 \bmod n$ for some $i \in [0, \dots k-1]$

Solutions → Miller's theorem

	$a \in [2, n-2]$																					
n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
5	1	1																				
7	−1	0	−1	0																		
9	–	–	–	–	–	–																
11	0	−1	−1	−1	0	0	0	−1														
13	1	−1	0	1	1	1	1	−1	0	1												
15	–	–	–	–	–	–	–	–	–	–	–	–										
17	2	3	1	3	3	3	2	2	3	3	3	1	3	2								
19	0	0	−1	−1	−1	−1	0	−1	0	−1	0	0	0	0	−1	−1						
21	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–				
23	−1	−1	−1	0	−1	0	−1	0	0	−1	−1	0	0	−1	0	0	0	0	0	0		
25	–	–	–	–	–	1	–	–	–	–	–	–	–	–	–	–	1	–	–	–	–	–

Cell	Test 1	Test 2	Comments
-1	✓	?	Test 1 passed; We don't care about Test 2
≥ 0	✗	✓	Cell = least $i \in [0, k-1]$ that passes Test 2
$-$	✗	✗	Test 1 failed; Test 2 failed

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

Solutions → Miller's theorem

PRIMALITY-MILLERTHEOREM(n)

```
if n > 2 and n is even then  return composite
 // Find exponent k and odd number m such that (n − 1) = 2^k × m
k ← 0; m ← n − 1
while m mod 2 = 0 do  { m ← m/2; k ← k + 1 }

 // Apply Miller's theorem. a = 1 & a = n − 1 are redundant.
for a ← 2 to n − 2 do
    T ← POWER(a, m, n)
     // Check test 1; If test 1 fails, check test 2 for i = 0
    if T = 1 or T = n − 1 then continue
     // Check test 2 for i ∈ [1, k − 1]
    for i ← 1 to k − 1 do
        T ← POWER(T, 2, n)
         // If T = 1, we only get 1's for future values of i
        if T = 1 then return composite
        if T = n − 1 then break
    if T ≠ n − 1 then return composite
return prime
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n \log^2 n\right), \Theta\left(1\right) \right\rangle$$

Solutions → Miller-Rabin's test

Primality-MillerRabinTest(n)

if $n > 2$ **and** n is even **then** **return** composite

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

$k \leftarrow 0$; $m \leftarrow n-1$

while $m \bmod 2 = 0$ **do** $\{ \ m \leftarrow m/2;\ k \leftarrow k+1 \ \}$

// Apply Miller's constraints in a randomized way as suggested by Rabin

for $count \leftarrow 1$ **to** $\#trials$ **do**

- $a \leftarrow$ RandomNumber($\{2, 3, 4, \ldots, n-2\}$)
- $T \leftarrow$ Power(a, m, n)
- // Check test 1; If test 1 fails, check test 2 for $i = 0$
- **if** $T = 1$ **or** $T = n-1$ **then continue**
- // Check test 2 for $i \in [1, k-1]$
- **for** $i \leftarrow 1$ **to** $k-1$ **do**
  - $T \leftarrow$ Power($T, 2, n$)
  - **if** $T = 1$ **then return** composite
  - **if** $T = n-1$ **then break**
- **if** $T \neq n-1$ **then return** composite

return prime

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(\#trials \cdot \log^2 n\right), \Theta(1) \right\rangle$$

Solutions → Naive AKS's test

- $n \geq 2$ is prime iff all coefficients, except first and last, of the nth row in the Pascal's triangle are multiples of n

	0	1	2	3	4	5	6	7	8	9
0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
5	1	5	10	10	5	1				
6	1	6	15	20	15	6	1			
7	1	7	21	35	35	21	7	1		
8	1	8	28	56	70	56	28	8	1	
9	1	9	36	84	126	126	84	36	9	1

Solutions → Naive AKS's test

- $n \geq 2$ is prime iff for all $i \in [1, n-1]$, ${}^{n}C_i$ is a multiple of n.

```
PRIMALITY-NAIVEAKSTEST(n)

// Step 1. Compute all required binomial coefficients
r ← ⌊n/2⌋                          // binomial coefficients are symmetric
Create an array C[0...r]
for i ← 0 to n do
    for j ← 0 to min(i, r) do
        if j = 0 or j = i then  C[j] ← 1
        else  C[j] ← C[j − 1] + C[j]

// Step 2. Check if the binomial coefficients are multiples of n
for j ← 1 to r do
    if C[j] mod n ≠ 0 then
        return composite
return prime
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^2\right), \Theta\left(n\right) \right\rangle$$

Complexity

Algorithm	Time	Space	Probabilistic?
Naive algorithm	$\mathcal{O}(n)$	$\Theta(1)$	✗
School algorithm	$\mathcal{O}(\sqrt{n})$	$\Theta(1)$	✗
Opt. school algo.	$\mathcal{O}(\sqrt{n})$	$\Theta(1)$	✗
SieveOfEratosthenes	$\mathcal{O}(\sqrt{n}\log\log n)$	$\Theta(\sqrt{n})$	✗
Wilson's theorem	$\Theta(n)$	$\Theta(1)$	✗
Fermat's theorem	$\mathcal{O}(n\log n)$	$\Theta(1)$	✗
Fermat's test	$\mathcal{O}(\#trials \cdot \log n)$	$\Theta(1)$	✓
Miller's theorem	$\mathcal{O}\left(n\log^2 n\right)$	$\Theta(1)$	✗
Miller-Rabin's test	$\mathcal{O}\left(\#trials \cdot \log^2 n\right)$	$\Theta(1)$	✓
Naive AKS test	$\Theta(n^2)$	$\Theta(n)$	✗

Membership

HOME

Problem

Problem

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

Solutions

	Add	Search	Comments
Balanced tree	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	works for sort-related ops.
Hash table	$\mathcal{O}(n)$ $\mathcal{O}(1)*$	$\mathcal{O}(n)$ $\mathcal{O}(1)*$	worst case is worse amortized case is awesome
Bloom filter	$\mathcal{O}(1)$	$\mathcal{O}(1)$	has false positive errors

- There are many more solutions.

Bloom filter

- 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 $A[0 \dots N-1]$
  Independent hash functions $h_1, h_2, \dots, h_k$ such that
  $h_i : \Sigma^* \rightarrow \{0, 1, 2, \dots, N-1\}$ such that the mapping is uniform

Bloom filter class

class $\textsc{BloomFilter}(n, p)$

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

$\textsc{Initialize}()$; $\textsc{Add}(x)$; $\textsc{Search}(x)$

$\textsc{Initialize}()$

$k \leftarrow \left\lfloor \frac{-\ln p}{\ln 2} \right\rfloor$; $N \leftarrow \left\lfloor n \cdot \frac{k}{\ln 2} \right\rfloor$
$A[0 \ldots N-1] \leftarrow [0 \ldots 0]$

$\textsc{Add}(x)$

for $i \leftarrow 1$ **to** k **do**
| $A[h_i(x)] \leftarrow 1$

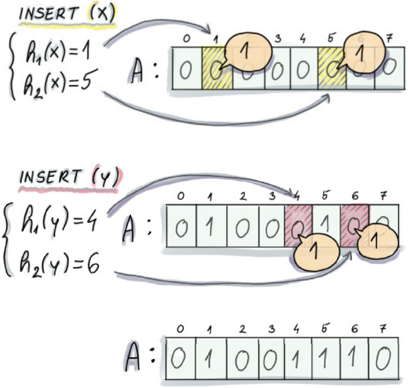
$\textsc{Search}(x)$

for $i \leftarrow 1$ **to** k **do**
| **if** $A[h_i(x)] \neq 1$ **then**
| | **return** false
return true

Add

$\textsc{Add}(x)$

for $i \leftarrow 1$ **to** k **do**
| $A[h_i(x)] \leftarrow 1$

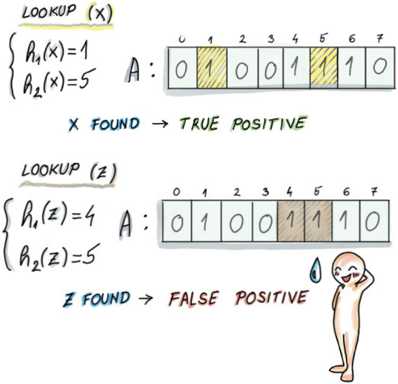


Source: Medjedovic-Tahirovic-Dedovic's Algorithms and Data Structures for Massive Datasets

Search

SEARCH(x)

for $i \leftarrow 1$ **to** k **do**
  if $A[h_i(x)] \neq 1$ **then**
    return false
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

$$\boxed{k \leftarrow \left\lfloor \frac{-\ln p}{\ln 2} \right\rfloor}$$

- Compute the Bloom filter bit array size N as

$$\boxed{N \leftarrow \left\lfloor n \cdot \frac{k}{\ln 2} \right\rfloor}$$

Feature	Complexity
Add	$\mathcal{O}(k) = \mathcal{O}(\ln(1/p))$
Search	$\mathcal{O}(k) = \mathcal{O}(\ln(1/p))$
Space	$\mathcal{O}(N) = \mathcal{O}(nk) = \mathcal{O}(n \ln(1/p))$

Error analysis

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

$$\boxed{\text{False positive prob.} = p = \left(1 - (1 - 1/N)^{nk}\right)^k}$$

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

$$\boxed{\text{False positive prob.} = p* = \left(\frac{1}{N^{k(n+1)}}\right) \cdot \sum_{i=1}^{m} i^k i! \ {}^{m}C_i \ {}^{kn}C_i}$$

- 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 $= \left(1 - (1 - 1/N)^{nk}\right)$

Prob. that k bits are 1 after n insertions $= \left(1 - (1 - 1/N)^{nk}\right)^k$

Simplify.

Prob. of false positives

$=$ Prob. that k bits are 1 after n insertions

$$= \left(1 - (1 - 1/N)^{nk}\right)^k = \left(1 - \left((1 - 1/N)^N\right)^{\frac{nk}{N}}\right)^k$$

$$\approx \left(1 - e^{\frac{nk}{N}}\right)^k \qquad (\because (1 - 1/x)^x \approx e)$$

So,

$$\boxed{\text{False positive probability} = p = \left(1 - e^{\frac{nk}{N}}\right)^k}$$

Error analysis

n	$p = 0.0001$	$p = 0.001$	$p = 0.01$	$p = 0.1$
10^1	$N = \lfloor N_{0.0001} n \rfloor$, $k = 13$	$N = \lfloor N_{0.001} n \rfloor$, $k = 10$	$N = \lfloor N_{0.01} n \rfloor$, $k = 7$	$N = \lfloor N_{0.1} n \rfloor$, $k = 3$
10^2	$N = \lfloor N_{0.0001} n \rfloor$, $k = 13$	$N = \lfloor N_{0.001} n \rfloor$, $k = 10$	$N = \lfloor N_{0.01} n \rfloor$, $k = 7$	$N = \lfloor N_{0.1} n \rfloor$, $k = 3$
10^3	$N = \lfloor N_{0.0001} n \rfloor$, $k = 13$	$N = \lfloor N_{0.001} n \rfloor$, $k = 10$	$N = \lfloor N_{0.01} n \rfloor$, $k = 7$	$N = \lfloor N_{0.1} n \rfloor$, $k = 3$
10^4	$N = \lfloor N_{0.0001} n \rfloor$, $k = 13$	$N = \lfloor N_{0.001} n \rfloor$, $k = 10$	$N = \lfloor N_{0.01} n \rfloor$, $k = 7$	$N = \lfloor N_{0.1} n \rfloor$, $k = 3$
10^5	$N = \lfloor N_{0.0001} n \rfloor$, $k = 13$	$N = \lfloor N_{0.001} n \rfloor$, $k = 10$	$N = \lfloor N_{0.01} n \rfloor$, $k = 7$	$N = \lfloor N_{0.1} n \rfloor$, $k = 3$
10^6	$N = \lfloor N_{0.0001} n \rfloor$, $k = 13$	$N = \lfloor N_{0.001} n \rfloor$, $k = 10$	$N = \lfloor N_{0.01} n \rfloor$, $k = 7$	$N = \lfloor N_{0.1} n \rfloor$, $k = 3$
10^7	$N = \lfloor N_{0.0001} n \rfloor$, $k = 13$	$N = \lfloor N_{0.001} n \rfloor$, $k = 10$	$N = \lfloor N_{0.01} n \rfloor$, $k = 7$	$N = \lfloor N_{0.1} n \rfloor$, $k = 3$
10^8	$N = \lfloor N_{0.0001} n \rfloor$, $k = 13$	$N = \lfloor N_{0.001} n \rfloor$, $k = 10$	$N = \lfloor N_{0.01} n \rfloor$, $k = 7$	$N = \lfloor N_{0.1} n \rfloor$, $k = 3$
10^9	$N = \lfloor N_{0.0001} n \rfloor$, $k = 13$	$N = \lfloor N_{0.001} n \rfloor$, $k = 10$	$N = \lfloor N_{0.01} n \rfloor$, $k = 7$	$N = \lfloor N_{0.1} n \rfloor$, $k = 3$
10^{10}	$N = \lfloor N_{0.0001} n \rfloor$, $k = 13$	$N = \lfloor N_{0.001} n \rfloor$, $k = 10$	$N = \lfloor N_{0.01} n \rfloor$, $k = 7$	$N = \lfloor N_{0.1} n \rfloor$, $k = 3$

- $N_{0.0001} = 19.170116754734877$, $N_{0.001} = 14.377587566051161$
  $N_{0.01} = 4.792529188683719$, $N_{0.1} = 0.9965784284662087$
- Note that Bloom filter bit array size N is in bits

References

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

Frequency HOME

Problem

Problem

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

Solutions

	Update	Estimate	Comments
Balanced tree	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	works for sort-related ops.
Hash table	$\mathcal{O}(n)$ $\mathcal{O}(1)*$	$\mathcal{O}(n)$ $\mathcal{O}(1)*$	worst case is worse amortized case is awesome
Count-min sketch	$\mathcal{O}(1)$	$\mathcal{O}(1)$	has false positive errors with approximations

- 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 $A[1 \dots k, 1 \dots w]$
  Independent hash functions $h_1, h_2, \dots, h_k$ such that
  $h_i : \Sigma^* \to \{1, 2, \dots, m\}$ such that the mapping is uniform

Count-min sketch class

class $\textsc{CountMinSketch}(\epsilon, \delta)$

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 $k \times w$

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

$\textsc{Initialize}()$

$k \leftarrow \left\lceil \ln \frac{1}{\delta} \right\rceil$; $w \leftarrow \left\lceil \frac{e}{\epsilon} \right\rceil$
$A[1 \ldots k, 1 \ldots w] \leftarrow [0 \ldots 0, 0 \ldots 0]$

$\textsc{Update}(x, c_x)$

for $i \leftarrow 1$ **to** k **do**
  $A[i, h_i(x)] \leftarrow A[i, h_i(x)] + c_x$

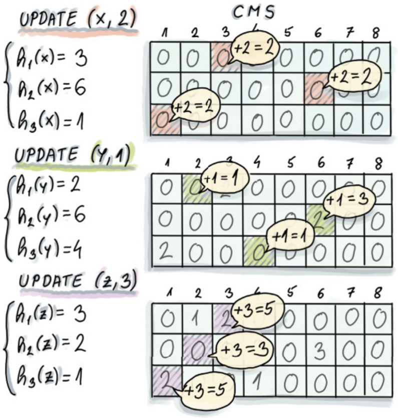
$\textsc{Estimate}(x)$

$min \leftarrow A[1, h_1(x)]$
for $i \leftarrow 2$ **to** k **do**
  if $A[i, h_i(x)] < min$ **then**
    $min \leftarrow A[i, h_i(x)]$

return min

Update

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

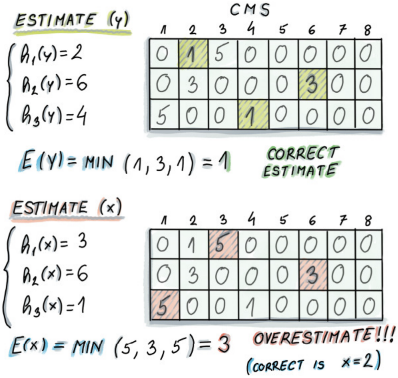


Source: Medjedovic-Tahirovic-Dedovic's Algorithms and Data Structures for Massive Datasets

Estimate

ESTIMATE(x)

$min \leftarrow A[1, h_1(x)]$
for $i \leftarrow 2$ **to** k **do**
  if $A[i, h_i(x)] < min$ **then**
    $min \leftarrow A[i, h_i(x)]$
return min



Source: Medjedovic-Tahirovic-Dedovic's Algorithms and Data Structures for Massive Datasets

Complexity analysis

- Given approximation fixed parameter ϵ
- Given error probability fixed parameter δ
- Compute the number of hash functions k as

$$\boxed{k \leftarrow \left\lceil \ln \frac{1}{\delta} \right\rceil}$$

- Compute the CMS matrix width size w as

$$\boxed{w \leftarrow \left\lceil \frac{e}{\epsilon} \right\rceil}$$

Feature	Complexity
Update	$\mathcal{O}(k) = \mathcal{O}(1)$
Estimate	$\mathcal{O}(k) = \mathcal{O}(1)$
Space	$\mathcal{O}(mk) = \mathcal{O}(1)$

Error and approximation analysis

- Let the data stream be $(a_1, c_1), (a_2, c_2), \ldots, (a_t, c_t)$
- Let $N =$ sum of all frequencies $= c_1 + c_2 + \cdots + c_t$
- Let $f_x^{\text{true}} =$ true frequency of item x in CMS

  Let $f_x^{\text{est}} =$ estimated frequency of item x in CMS

  Then

  $$\boxed{f_x^{\text{est}} \text{ is in } \begin{Bmatrix} [f_x^{\text{true}}, f_x^{\text{true}} + \epsilon \cdot N] & \text{with probability} \geq 1 - \delta \\ (f_x^{\text{true}} + \epsilon \cdot N, \infty) & \text{with probability} \leq \delta \end{Bmatrix}}$$

  where, $\epsilon, \delta \in (0, 1)$

Differences between Bloom filter and CMS

Feature	Bloom filter	CMS
Duplicates	Set	Multiset
Array	1-D	2-D
Hash function	Maps to entire array	Maps to portions of 2-D array
Query	Existential queries	Counting queries
Size	Typ. linear w.r.t set size	Typ. sublinear w.r.t total freq
Randomness	Uniformly random	Uni. random & pairwise ind.

Cardinality HOME

Problem

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...n])

distinct ← 1
for i ← 2 to n do
    j ← 1
    while j ≤ i do
        // Check current element with previous (j < i) or current (j = i)
        if A[i] = A[j] then break
        j ← j + 1
    // If there is no previous duplicate, then increment distinct
    if j = i then
        distinct ← distinct + 1
return distinct
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n^2\right), \Theta\left(1\right) \right\rangle$$

Solutions → Sort and count

1. Sort the array
2. Equal values are together in the sorted input
3. Count the first occurrence of each value

```
SortAndCount(A[1...n])
  distinct ← 1
  Sort(A[1...n])
  for i ← 2 to n do
      // Current value is different from previous value
      if A[i] ≠ A[i − 1] then
          distinct ← distinct + 1
  return distinct
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n \log n), \Theta(n) \rangle$$

Solutions $\rightarrow$ Bit vector

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

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

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

```
HashSet(A[1 ... n])
Create a hash set H to store unique values
for i ← 1 to n do
|  H.Add(A[i])
distinct ← H.Size()          // #elements in the hash set
return distinct
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$$

Solutions $\rightarrow$ Linear counting

- 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

Solutions $\rightarrow$ Linear counting

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

$$\boxed{\#\text{Distinct elements} = \#\text{zeros in the bit array}}$$

Solutions → Linear counting

```
LinearCounting(A[1...n])

m ← n                                         // assuming n is known
linearcounter[1...m] ← [0...0]

// Adding elements to the linear counter
for i ← 1 to n do
|  linearcounter[h(A[i])] ← 1

// Compute the number of zeros in the linear counter
zercount ← 0
 for i ← 1 to n do
|  if linearcounter[i] = 0 then
|  |  zerocount ← zerocount + 1

// Estimate #distinct elements using probabilistic analysis
distinct ← ⌊−m × ln(zerocount / m)⌋

return distinct
```

$$\boxed{distinct \leftarrow \left\lfloor -m \times \ln\left(\frac{zerocount}{m}\right)\right\rfloor}$$

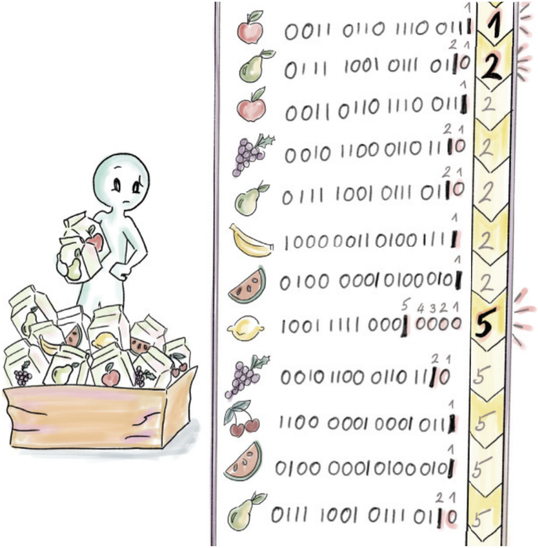
$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$$

Solutions → Probabilistic counting

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

$$\boxed{\#\text{Distinct elements} = 2^{z_{\text{max}}}}$$

Solutions → Probabilistic counting



#distinct=7, 16-bit hashes, estimated #distinct$= 2^5 = 32$

Source: Medjedovic-Tahirovic-Dedovic's Algorithms and Data Structures for Massive Datasets

Solutions → Probabilistic counting

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

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

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

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 $\underbrace{0\ldots0}_{j}$ as the last digits

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 $\underbrace{0\ldots0}_{j}$ 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 $1\underbrace{0\ldots0}_{j-1}$) is $1/2^j$

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 $\underbrace{0\ldots0}_{j}$ 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 $1\underbrace{0\ldots0}_{j-1}$) is $1/2^j$

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

An event having prob. $1/2^{z_{\mathsf{max}}}$ occurs if on avg. $2^{z_{\mathsf{max}}}$ trials are performed

Solutions $\rightarrow$ Stochastic averaging

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

Solutions $\rightarrow$ Stochastic averaging

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

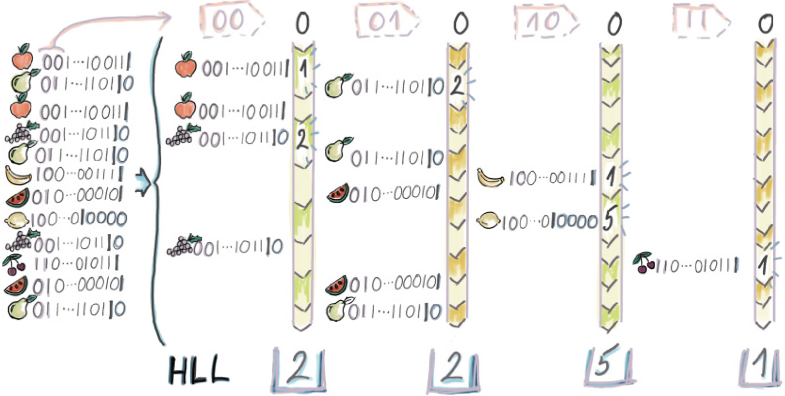
- Have $m = 2^b$ buckets
- Find the bucket using the last b bits of the hash value
- Perform probabilistic counting with the remaining bits.
- Add the distinct items in all buckets

Solutions $\rightarrow$ Stochastic averaging

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 $z_{\text{avg}} = \frac{z_1+z_2+\cdots+z_m}{m}$
   $=$ average estimator/prediction of all buckets

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

Solutions → Stochastic averaging



#distinct=7, 16-bit hashes, 4 buckets, estimated
#distinct= Round $\left(4 \cdot 2^{2.5}\right) = 23$

Source: Medjedovic-Tahirovic-Dedovic's Algorithms and Data Structures for Massive Datasets

Solutions → Stochastic averaging

```
StochasticAveraging(A[1...n])
  // Initialize estimators in m buckets
  Create an array z_max[0, 1, ..., m − 1] ← [0, 0, ..., 0]
  // Compute estimators in m buckets
  for i ← 1 to n do
      bucket ← h(A[i]) mod m                      // determines bucket
      buckethash ← ⌈h(A[i])/m⌉                    // determines hash in bucket
      z ← CountTrailingZeros(buckethash)
      if z > z_max[bucket] then z_max[bucket] ← z
  // Find the average of estimators in m buckets
  z_avg ← (1/m) · (z_max[0] + z_max[1] + ··· + z_max[m − 1])
  // Estimate the #distinct elements in all buckets
  distinct ← Round(m · 2^(z_avg))
  return distinct
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

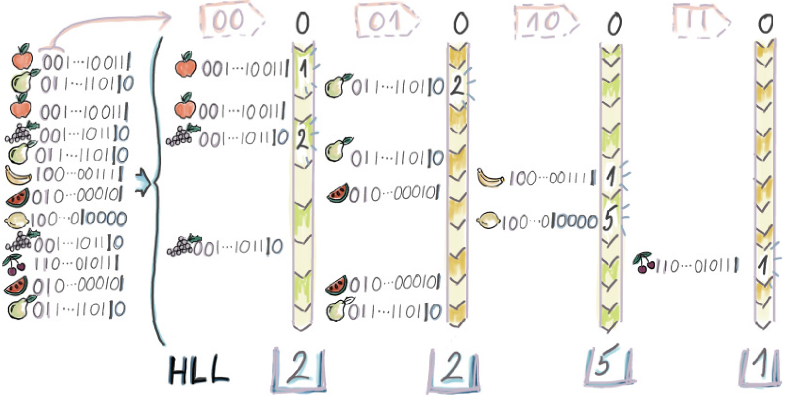
Solutions $\rightarrow$ LogLog

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 $z_{\text{avg}} = \frac{z_1+z_2+\cdots+z_m}{m}$
   = average estimator/prediction of all buckets

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

$$\boxed{\alpha_m = \left\{\begin{array}{ll} 0.39701 - \frac{2\pi^2+\ln^2 2}{48m} & \text{if } m < 64, \\ 0.39701 & \text{if } m \geq 64. \end{array}\right\}}$$

Solutions → LogLog



$\#$distinct$=7$, 16-bit hashes, 4 buckets, estimated

$\#$distinct$=$ 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

$\textsc{LogLog}(A[1 \ldots n])$

// 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**
  $bucket \leftarrow h(A[i]) \bmod m$ // determines bucket
  $buckethash \leftarrow \lceil h(A[i])/m \rceil$ // determines hash in bucket
  $z \leftarrow \text{CountTrailingZeros}(buckethash)$
  if $z > z_{\text{max}}[bucket]$ **then** $z_{\text{max}}[bucket] \leftarrow z$

// 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
$distinct \leftarrow \text{Round}\left(\alpha_m \cdot m \cdot 2^{z_{\text{avg}}}\right)$
return $distinct$

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Solutions $\rightarrow$ HyperLogLog

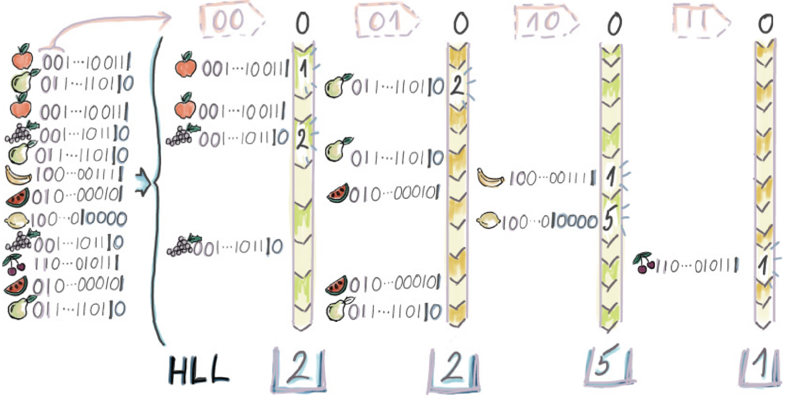
1. Let $z_1 =$ estimator/prediction for bucket 1
2. Let $z_2 =$ estimator/prediction for bucket 2
3. so on...
4. Let $z_m =$ estimator/prediction for bucket m
5. Let $B = \frac{m}{\frac{1}{2^{z_1}}+\frac{1}{2^{z_2}}+\cdots+\frac{1}{2^{z_m}}}$

   $=$ average estimator/prediction of all buckets

$$\boxed{\#\text{Distinct elements} = \text{Round}\left(\beta_m \cdot m \cdot B\right)}$$

$$\boxed{\beta_m = \begin{Bmatrix} 0.541 & \text{if } m = 4, \\ 0.627 & \text{if } m = 8, \\ 0.673 & \text{if } m = 16, \\ 0.697 & \text{if } m = 32, \\ 0.709 & \text{if } m = 64, \\ \frac{0.723}{1+1.079/m} & \text{if } m \geq 128. \end{Bmatrix}}$$

Solutions → HyperLogLog



#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

Solutions → HyperLogLog

$\textsc{HyperLogLog}(A[1 \ldots n])$

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

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

// Find the harmonic average of estimators in m buckets

$$B \leftarrow \left(\frac{m}{\frac{1}{2^{z_{\mathsf{max}}[0]}} + \frac{1}{2^{z_{\mathsf{max}}[1]}} + \cdots + \frac{1}{2^{z_{\mathsf{max}}[m-1]}}} \right)$$

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

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Complexity

Algorithm	Time	Space
Brute force	$\mathcal{O}(n^2)$	$\Theta(1)$
Sort and count	$\Theta(n \log n)$	$\Theta(n)$
Bit vector	$\Theta(n + U)$	$\Theta(U)$
Hash table	$\Theta(n)$	$\Theta(n)$
Linear counting	$\Theta(n)$	$\Theta(n)$
Probabilistic counting	$\Theta(n)$	$\Theta(1)$
Stochastic averaging	$\Theta(n)$	$\Theta(1)$
LogLog	$\Theta(n)$	$\Theta(1)$
HyperLogLog	$\Theta(n)$	$\Theta(1)$

References

- Book: Algorithms and Data Structures for Massive Datasets
- Book: Probabilistic Data Structures and Algorithms for Big Data Applications

External-Memory Algorithms

Merge k Sorted Arrays HOME

Problem

- 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
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(kn \log\left(kn\right)\right), \Theta\left(kn\right) \rangle$$

Solutions $\rightarrow$ Naive merging

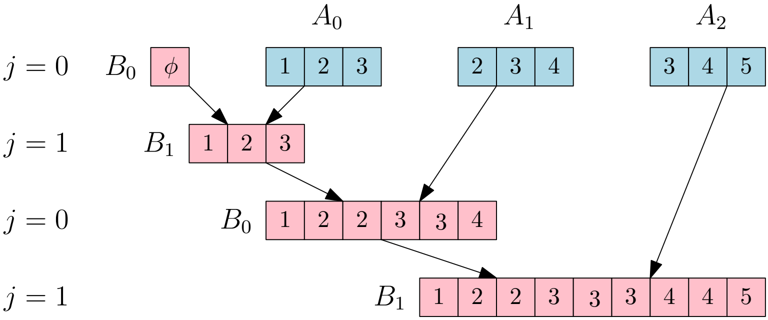
- Create an empty dynamic array
- Merge each array with the evolving array one-by-one

$\textsc{NaiveMerging}(A_1[1 \dots n], \dots, A_k[1 \dots n])$

Create two dynamic arrays $B_0 \leftarrow [\;]$ and $B_1 \leftarrow [\;]$
$j \leftarrow 0$
for $i \leftarrow 1$ **to** k **do**
  $j \leftarrow (j+1) \bmod 2$
  $B_j \leftarrow \textsc{Merge}(B_{(j+1) \bmod 2}, A_i)$
return B_j

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(k^2 n\right), \Theta\left(kn\right) \right\rangle$$

B_0 and B_1 are empty and $j = 0$



Return B_j, where $j = 1$

Solutions → Divide-and-conquer

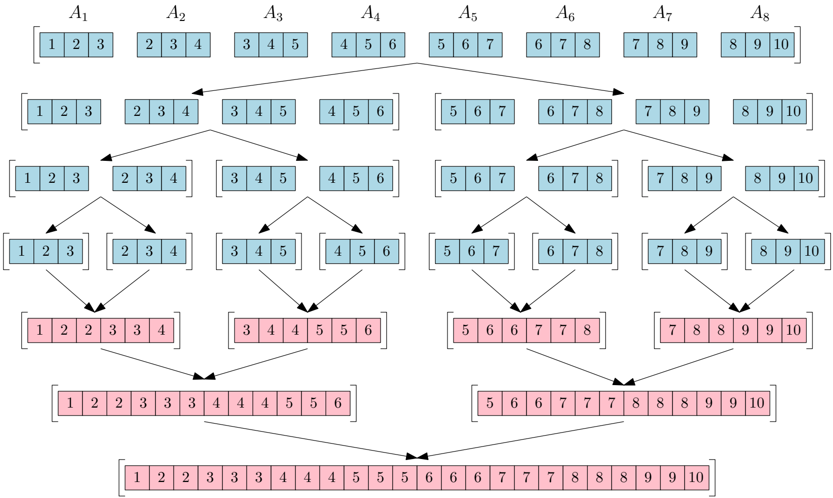
- Merge arrays in groups of two to get $k/2$ arrays
- Merge arrays in groups of two to get $k/4$ arrays
- Repeat this process until there is only one array
- That single array is the merged sorted array

```
MergeKSortedArrays(A_1[1...n], ..., A_k[1...n])

return Merge-D&C(A_1[1...n], ..., A_k[1...n])
```

```
Merge-D&C(A_low, ..., A_high)

n ← (high − low + 1)
if n = 1 then  return A_low
mid ← (low + high)/2
 // Split the arrays into left and right sets
A_left ← Merge-D&C([A_low, A_low+1, ..., A_mid])
A_right ← Merge-D&C([A_mid+1, A_mid+2, ..., A_high])
 // Merge the left and right sets
A_merged ← Merge(A_left, A_right)
return A_merged
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(kn\log k\right), \Theta\left(kn\right) \rangle$$

Solutions → Divide-and-conquer



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

Solutions → Naive k-way merge

```
NaivekwayMerge(A_1[1...n], ..., A_k[1...n])
  A_merged ← [ ]
  pointer[1...k] ← [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 ← ∞; minindex ← −1
      for i ← 1 to k do
          if pointer[i] ≤ n and A_i[pointer[i]] < minval then
              minval ← A_i[pointer[i]]
              minindex ← 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] ← pointer[minindex] + 1
  return A_merged
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(k^2 n\right), \Theta\left(kn\right) \right\rangle$$

Solutions → Naive k-way merge

				A_{merged}	$pointer$
A_1	1	2	3		1
A_2	2	3	4	ϕ	1
A_3	3	4	5		1

				A_{merged}	$pointer$
A_1	1	2	3		2
A_2	2	3	4	1	1
A_3	3	4	5		1

				A_{merged}	$pointer$
A_1	1	2	3		3
A_2	2	3	4	1 2	1
A_3	3	4	5		1

				A_{merged}	$pointer$
A_1	1	2	3		3
A_2	2	3	4	1 2 2	2
A_3	3	4	5		1

				A_{merged}	$pointer$
A_1	1	2	3		4
A_2	2	3	4	1 2 2 3	2
A_3	3	4	5		1

Solutions → Naive k-way merge (continued)

				A_{merged}	$pointer$
A_1	1	2	3		4
A_2	2	3	4	1 2 2 3 3	3
A_3	3	4	5		1

A_1	1	2	3		4
A_2	2	3	4	1 2 2 3 3 3	3
A_3	3	4	5		2

A_1	1	2	3		4
A_2	2	3	4	1 2 2 3 3 3 4	4
A_3	3	4	5		2

A_1	1	2	3		4
A_2	2	3	4	1 2 2 3 3 3 4 4	4
A_3	3	4	5		3

A_1	1	2	3		4
A_2	2	3	4	1 2 2 3 3 3 4 4 5	4
A_3	3	4	5		4

Solutions → Improved k-way merge

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

Solutions → Improved k-way merge

```
ImprovedkwayMerge(A_1[1...n], ..., A_k[1...n])

A_merged ← [ ]; Create a hash map pointer where the key is the array number
and the value is the index inside that array
for i ← 1 to k do  pointer.Add(⟨i, 1⟩)
// 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 ← ∞; minindex ← −1
    foreach key in pointer.Keys() do
        if A_key[pointer[key]] < minval then
            minval ← A_key[pointer[key]]
            minindex ← 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] ← 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
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(k^2 n\right), \Theta\left(kn\right) \right\rangle$$

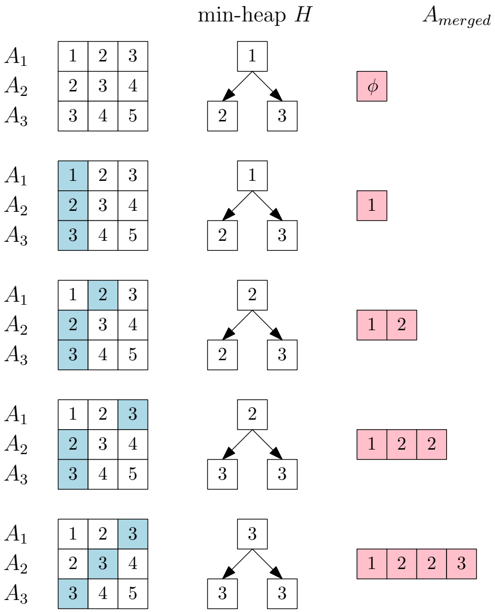
Solutions → Best k-way merge

- 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.

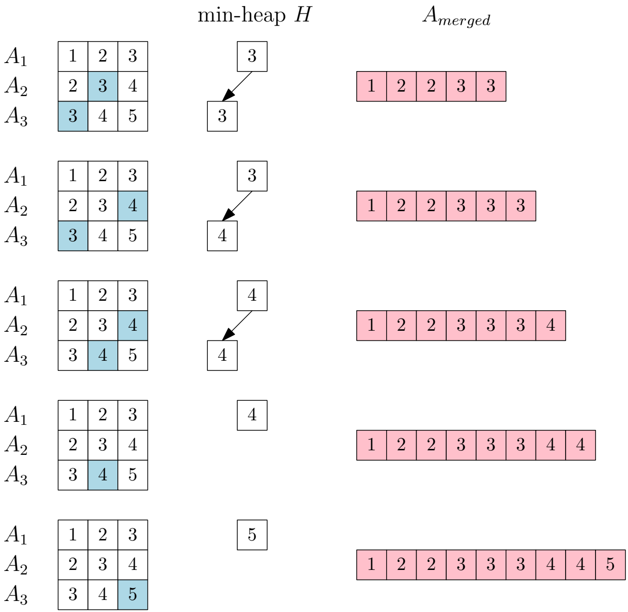
```
BestkwayMerge(A_1[1...n], ..., A_k[1...n])

A_merged ← [ ]
Create a min-heap H and initialize with the first element of each array
for i ← 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) ← H.RemoveMin()
|  A_merged.Append(minelement)
|  // Move to the next element in the array that contributed the min element
|  elementindex ← elementindex + 1
|  // If there are more elements in the current array, insert the next element
|     into the heap
|  if elementindex ≤ n then
|  |  H.Add((A_arrayindex[elementindex], arrayindex, elementindex))
return A_merged
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(nk \log k\right), \Theta\left(kn\right) \rangle$$

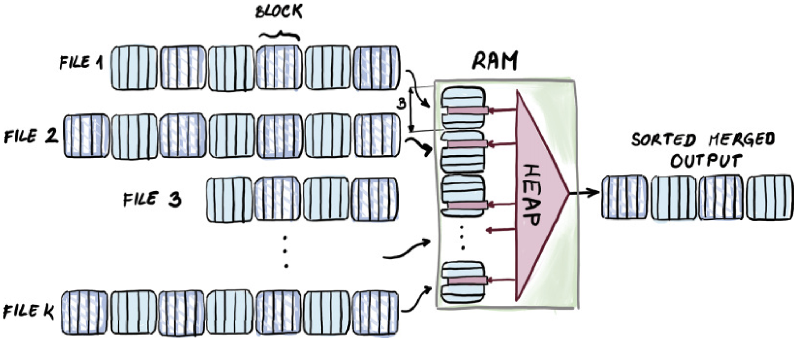
Solutions → Best k-way merge



Solutions → Best k-way merge (continued)



Solutions → External-memory k-way merge



Source: Medjedovic-Tahirovic-Dedovic's Algorithms and Data Structures for Massive Datasets

Solutions $\rightarrow$ External-memory k-way merge

kWAYMERGE$(F_1, F_2, \ldots, F_k, M, B)$

For each file F_i, create an $inputbuffer[i]$ 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 $outputbuffer$ to store the result
Initialize min-heap H with the first element of every input buffer

```
for i ← 1 to k do
    element ← ReadNextElementFromInputBuffer(inputbuffer[i])
    if element is not none then  H.Add(element, i)

while min-heap H is not empty do
    // Remove from min-heap H the min and input buffer index; and write to output buffer
    minelement, bufferindex ← H.RemoveMin()
    WriteElementToOutputBuffer(minelement, outputbuffer)
    if output buffer is full then
        WriteOutputBufferToDisk(outputbuffer, merged file F_merged)
        Clear contents of outputbuffer

    // Add to min-heap H the next element from input buffer having index bufferindex
    element ← ReadNextElementFromInputBuffer(inputbuffer[bufferindex])
    if element is none then
        inputbuffer[bufferindex] ← ReadNextDataBlockFromDisk(F_bufferindex)
        element ← ReadNextElementFromInputBufferinputbuffer[bufferindex]
    H.Add(element, bufferindex)

return F_merged
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(\frac{kn}{B}\right) \text{ I/Os}, \Theta\left(kn\right) \right\rangle$$

Complexity

Algorithm	Time	Space
Internal-memory algorithms		
Naive solution	$\Theta\left(kn\log\left(kn\right)\right)$	$\Theta\left(kn\right)$
Naive merging	$\Theta\left(k^2n\right)$	$\Theta\left(kn\right)$
Divide-and-conquer	$\Theta\left(kn\log k\right)$	$\Theta\left(kn\right)$
Naive k-way merge	$\Theta\left(k^2n\right)$	$\Theta\left(kn\right)$
Improved k-way merge	$\mathcal{O}\left(k^2n\right)$	$\Theta\left(kn\right)$
Best k-way merge	$\Theta\left(nk\log k\right)$	$\Theta\left(kn\right)$
External-memory algorithms		
External-memory k-way merge	$\Theta\left(\frac{kn}{B}\right)$ I/Os	$\Theta\left(kn\right)$

Merge Sort

HOME

Solutions → 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 $k = M/B$ ....
$sortedfile \leftarrow$ RecursiveMerge($sortedchunks, k$)
return $sortedfile$

RecursiveMerge($sortedchunks, k$)

if there is only one chunk in $sortedchunks$ **then**
  return the only sorted chunk
Create $newsortedchunks \leftarrow [\,]$ to contain pointers to merged chunks
while there $sortedchunks$ 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($newsortedchunks, k$)

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(\frac{n}{B}\log_{\frac{M}{B}}\frac{n}{B}\right)\ \text{I/Os}, \Theta(n) \right\rangle$$

Solutions → Merge sort (Non-recursive)

MergeSortNonRecursive($unsortedfile, M, B$)

// 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 hard disk

Create $sortedchunks$ to contain pointers to all sorted chunks

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

while $sortedchunks$ has more than one sorted file **do**

  Create $newsortedchunks$ to contain pointers to merged chunks

  Divide all sorted chunks in groups of k, except possibly the last group

  foreach group of k sorted chunks **do**

    Merge the k chunks into a single sorted chunk using k-way merge

    Append this merged chunk to $newsortedchunks$

  $sortedchunks \leftarrow newsortedchunks$

Let the only file in $sortedchunks$ be called $sortedfile$

return $sortedfile$

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(\frac{n}{B} \log_{\frac{M}{B}} \frac{n}{B}\right) \text{ I/Os}, \Theta\left(n\right) \right\rangle$$

Quantum Algorithms

Fundamentals HOME

What is Hilbert space?

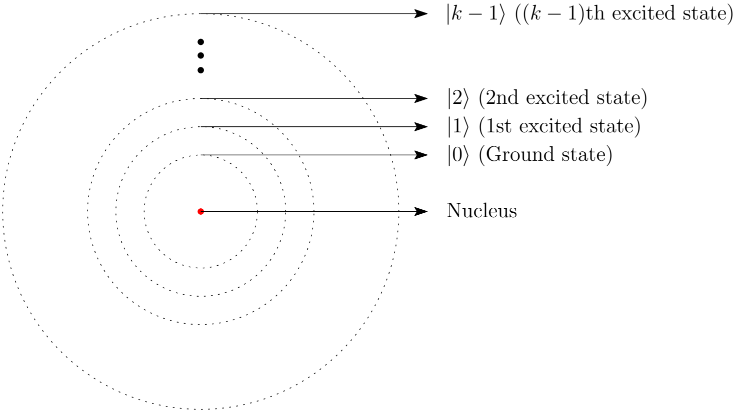
Definition

- The inner/dot/scalar product $\vec{v} \bullet \vec{w}$ 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,

  $\vec{v} \bullet \vec{w} = \overline{v_1} w_1 + \overline{v_2} w_2 + \cdots + \overline{v_n} w_n.$
- Hilbert space is a complex vector Euclidean space with well-defined inner product.

What is superposition?



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.

What is superposition?

Superposition is when a quantum particle

is in multiple states simultaneously

What is superposition?

Energy of an electron in an atom

- After measuring, the electron can be in any one of $|0\rangle$, $|1\rangle$, ..., $|k-1\rangle$ quantum energy states, where

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, |k-1\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

  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.,

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

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

  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?

$\boxed{\text{A qubit represents the superpositioned state}}$

$\boxed{\text{of a 2-state quantum system}}$

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

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

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

where $|a|^2 + |b|^2 = 1$

Qubit $|\psi\rangle$ is invalid if $|a|^2 + |b|^2 \neq 1$

Measuring a qubit

$\boxed{\text{A qubit when measured collapses to one of the two basis states}}$

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

$\boxed{\text{Probability of measuring qubit } |\psi\rangle \text{ as } |0\rangle \text{ is } |a|^2}$

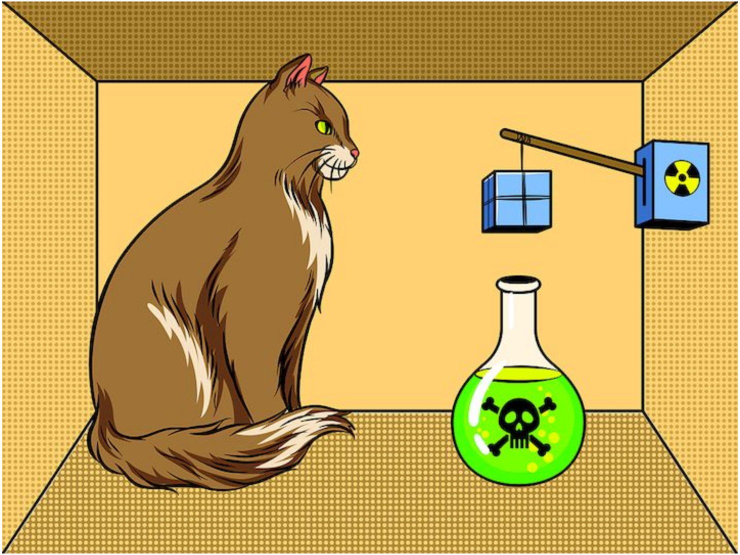
$\boxed{\text{Probability of measuring qubit } |\psi\rangle \text{ as } |1\rangle \text{ is } |b|^2}$

Observe that the sum of all collapsing probabilities must be 1

Comparison between classical and quantum bit

Feature	Bit	Qubit
Implementation	Transistor	Quantum system
Exclusive states	0 and 1	$\|0\rangle$ and $\|1\rangle$
State after measuring	0 or 1	$\|0\rangle$ or $\|1\rangle$
State before measuring	0 or 1	Superposition of $\|0\rangle$ and $\|1\rangle$
Representation	bit $\in \{0, 1\}$	qubit $= a\|0\rangle + b\|1\rangle$

Schrödinger's cat



Source: https://www.rms.com/blog/2018/09/04/schrodingers-cat-model

Schrödinger's cat

a |  ⟩ $+\ b$ |  ⟩

Problem

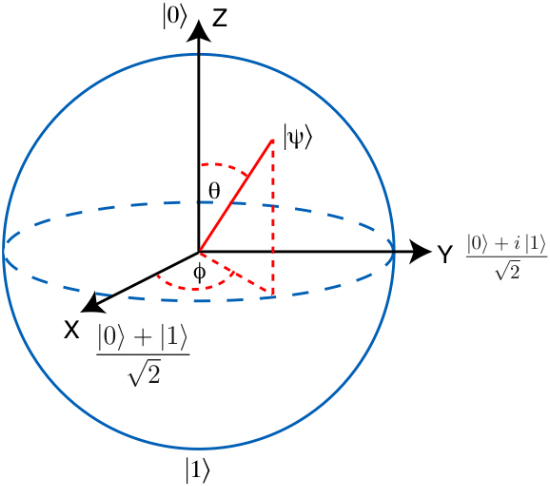
- Suppose the probabilities of cat being alive and the cat being dead are the same. Then, what are the values of a and b?

How to visualize a qubit? Block sphere

- 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

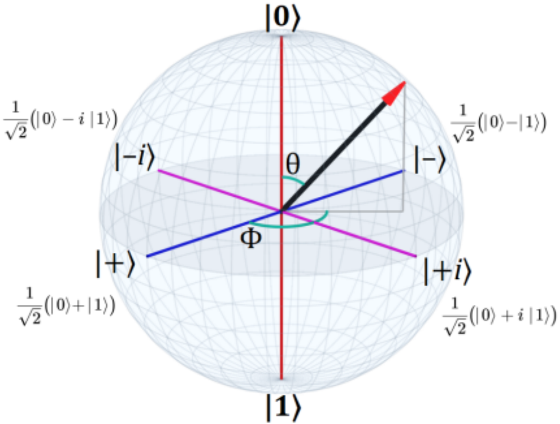
What should be the three basis vectors or axes?

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/

Axis	Basis	Meaning
Z	$\|0\rangle$ and $\|1\rangle$	$\|0\rangle$ and $\|1\rangle$
Y	$\|i\rangle$ and $\|-i\rangle$	$\frac{\|0\rangle+i\|1\rangle}{\sqrt{2}}$ and $\frac{\|0\rangle-i\|1\rangle}{\sqrt{2}}$
X	$\|+\rangle$ and $\|-\rangle$	$\frac{\|0\rangle+\|1\rangle}{\sqrt{2}}$ and $\frac{\|0\rangle-\|1\rangle}{\sqrt{2}}$

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

$$\boxed{U \cdot |\psi_1\rangle = |\psi_2\rangle}$$

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
  $\boxed{H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}}$

What is conjugate transpose of a matrix?

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 $U_1 U_2 U_3 \cdots U_k|\psi_1\rangle = |\psi_2\rangle$, 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)

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

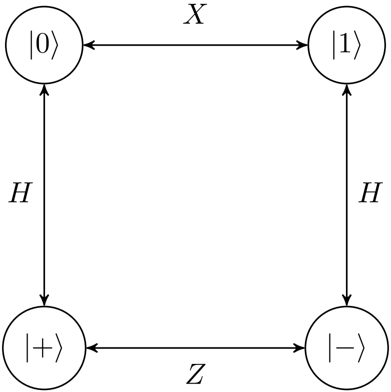
These rotations can be composed to perform

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



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

$$a|0\rangle + b|1\rangle \;-\!\boxed{X}\!-\; b|0\rangle + a|1\rangle$$

$$a|0\rangle + b|1\rangle \;-\!\boxed{Y}\!-\; -ib|0\rangle + ia|1\rangle$$

$$a|0\rangle + b|1\rangle \;-\!\boxed{Z}\!-\; a|0\rangle - b|1\rangle$$

$$a|0\rangle + b|1\rangle \;-\!\boxed{H}\!-\; a|+\rangle - b|-\rangle$$

$$a|0\rangle + b|1\rangle \;-\!\boxed{S}\!-\; a|0\rangle + be^{i\pi/2}|1\rangle$$

$$a|0\rangle + b|1\rangle \;-\!\boxed{T}\!-\; a|0\rangle + be^{i\pi/4}|1\rangle$$

Random Number Generator

HOME

Random number generation

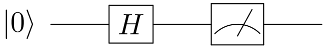
Problem

- Generate a truly random bit.

Solution

- Cannot be solved in classical computing
- Can be solved in quantum computing

Random number generation



Random number generation

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

$$\boxed{\#\text{dimensions is directly proportional to } 2^{\#\text{qubits}}}$$

- 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

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

- 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

- $|00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
- $|01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$
- $|10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$
- $|11\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

Small vs. Large quantum systems

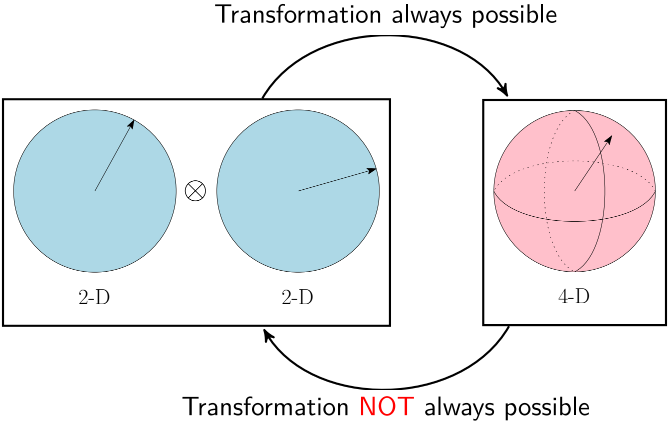
Transformation always possible

n small systems of 2-D
n one-qubits
$|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$

1 large system of 2^n-D
1 n-qubit
$|\Psi\rangle$

Transformation NOT always possible

Small vs. Large quantum systems



Quantum system transformation: Small $\rightarrow$ Large

Definition

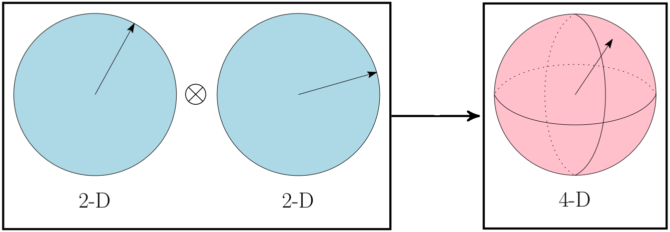
- A joint system of n small 2-D quantum systems, each having 2 quantum states can be thought as a large 2^n-D quantum mechanical system having 2^n quantum states.
- The tensor product $\otimes$ (or Kronecker product) of n one-qubits can be thought to denote a quantum mechanical system having 2^n quantum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

Examples

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

E.g.: $|1\rangle \otimes |0\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |101\rangle.$

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

Definition

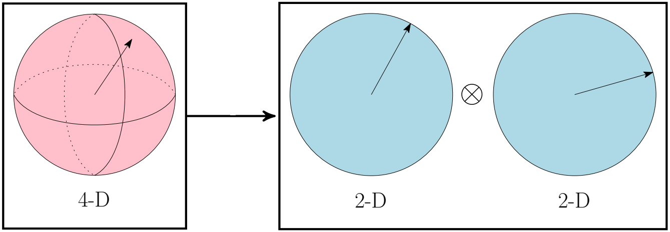
- A large 2^n-D quantum mechanical system having 2^n quantum states can be thought as a joint system of n small 2-D quantum systems, each having 2 quantum states.
- A quantum mechanical system having 2^n quantum states can be thought as a tensor product $\otimes$ (or Kronecker product) of n one-qubits.

Examples

- An 8-D quantum system can be represented using 3 qubits.

E.g.: $|101\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \otimes |0\rangle \otimes |1\rangle$.

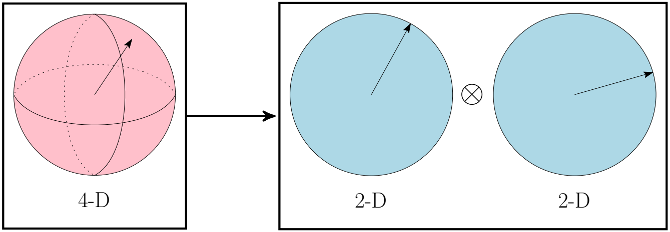
What is a separable state?



Definition

- The two-qubit state $|\Psi\rangle$ is separable if $\boxed{|\Psi\rangle = |\psi\rangle \otimes |\phi\rangle}$ for some one-qubit states $|\psi\rangle$ and $|\phi\rangle$.

What is a separable state?



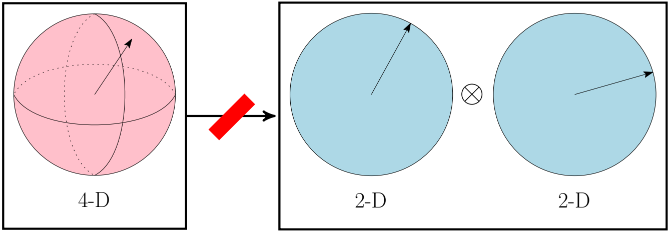
Examples

- 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 $|\phi\rangle = |+\rangle$.

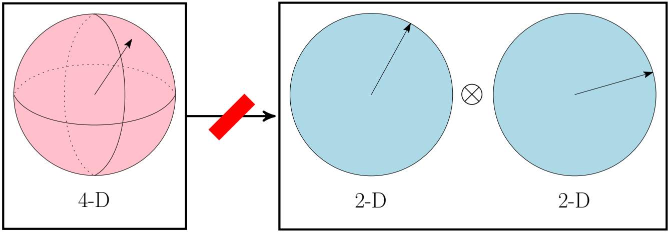
What is an entangled state?



Definition

- The two-qubit state $|\Psi\rangle$ is entangled if $\boxed{|\Psi\rangle \neq |\psi\rangle \otimes |\phi\rangle}$ 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?

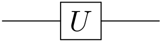
Definition

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

Bell states
$\lvert\Phi^+\rangle = \frac{1}{\sqrt{2}}(\lvert 00\rangle + \lvert 11\rangle)$
$\lvert\Phi^-\rangle = \frac{1}{\sqrt{2}}(\lvert 00\rangle - \lvert 11\rangle)$
$\lvert\Psi^+\rangle = \frac{1}{\sqrt{2}}(\lvert 01\rangle + \lvert 10\rangle)$
$\lvert\Psi^-\rangle = \frac{1}{\sqrt{2}}(\lvert 01\rangle - \lvert 10\rangle)$

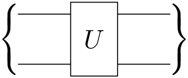
2-qubit gates

Single-qubit gate.

$a|0\rangle + b|1\rangle$  $a'|0\rangle + b'|1\rangle$

$$\begin{bmatrix} a' \\ b' \end{bmatrix} = U \begin{bmatrix} a \\ b \end{bmatrix}$$

Two-qubit gate.

$a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$  $a'|00\rangle + b'|01\rangle + c'|10\rangle + d'|11\rangle$

$$\begin{bmatrix} a' \\ b' \\ c' \\ d' \end{bmatrix} = U \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

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: $CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
- $SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Technical Problems and Solutions

HOME

Majority Vote

HOME

Problem

- 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

- Assumption: Equality comparison $(A[i] = A[j])$ between elements are allowed. Inequality comparisons $(A[i] \leq A[j]$ or $A[i] < A[j])$ between elements are not allowed.

- 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 $\rightarrow$ 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.
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)

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

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^2\right), \Theta\left(1\right) \right\rangle$$

Solutions → Sorting

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

$\textsc{Majority-Sort}(A[1 \dots n])$

$A[1 \dots n] \leftarrow \textsc{Sort}(A[1 \dots n])$

$i \leftarrow 1$
for $j \leftarrow 2$ **to** n **do**
  if $A[j] \neq A[i]$ **then**
    if $(j - i) > \lfloor n/2 \rfloor$ **then**
      return $A[i]$
    $i \leftarrow j$

if $(n - i + 1) > \lfloor n/2 \rfloor$ **then**
  return $A[i]$
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

$\textsc{Majority-D\&C}(A[1 \ldots n])$

return $\mathrm{D\&C}(A[1 \ldots n])$

$\mathrm{D\&C}(A[low \ldots high])$

if $low = high$ **then** **return** $A[low]$
$size \leftarrow (high - low + 1)$; $mid \leftarrow \lfloor (low + high)/2 \rfloor$
$\ell majority \leftarrow \mathrm{D\&C}(A[low \ldots mid])$
$rmajority \leftarrow \mathrm{D\&C}(A[(mid + 1) \ldots high])$
$\ell count \leftarrow \textsc{CountOccurrences}(A[low \ldots high], \ell majority)$
$rcount \leftarrow \textsc{CountOccurrences}(A[low \ldots high], rmajority)$
if $\ell count > \lfloor size/2 \rfloor$ **then** **return** $\ell majority$
if $rcount > \lfloor size/2 \rfloor$ **then** **return** $rmajority$
return -1

$$\langle \text{Time, Space} \rangle = \langle \Theta(n \log n), \Theta(\log n) \rangle$$

Solutions $\rightarrow$ Hashing

1. Store $\langle uniqueelement, frequency \rangle$ pairs in hash map
2. Find majority

$\text{Majority-Hashing}(A[1 \dots n])$

```
Create hash map H to insert (element, frequency) pairs
for i ← 1 to n do
    if H.ContainsKey(A[i]) then
        H.Add(⟨A[i], H.GetValue(A[i]) + 1⟩)
    else
        H.Add(⟨A[i], 1⟩)
    if H.GetValue(A[i]) > ⌊n/2⌋ then
        return A[i]
return −1
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$$

Solutions → Median

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

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

$median \leftarrow \textsc{Selection}(A[1 \ldots n], \lfloor n/2 \rfloor)$
$count \leftarrow \textsc{CountOccurrences}(A[1 \ldots n], median)$
if $count > \lfloor n/2 \rfloor$ **then**
  return $median$
return -1

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Solutions $\rightarrow$ Probabilistic

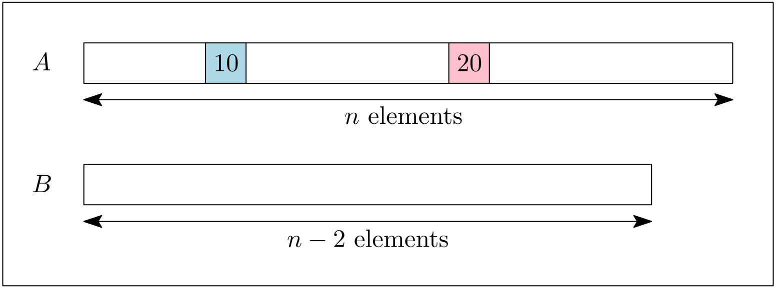
1. Select a random element and check if it is majority
2. Repeat Step 1 for at most $\left\lfloor \log_2 \frac{1}{\epsilon} \right\rfloor$ number of times
3. Return majority

$\textsc{Majority-Probabilistic}(A[1 \ldots n])$

for $i \leftarrow 1$ **to** $\left\lfloor \log_2 \frac{1}{\epsilon} \right\rfloor$ **do**
    $random \leftarrow \textsc{Random}(A[1 \ldots n])$
    $count \leftarrow \textsc{CountOccurrences}(A[1 \ldots n], random)$
    if $count > \lfloor n/2 \rfloor$ **then**
        return $random$
return -1

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n \log \frac{1}{\epsilon}\right), \Theta(1) \right\rangle$$

Core idea (take-home lesson)



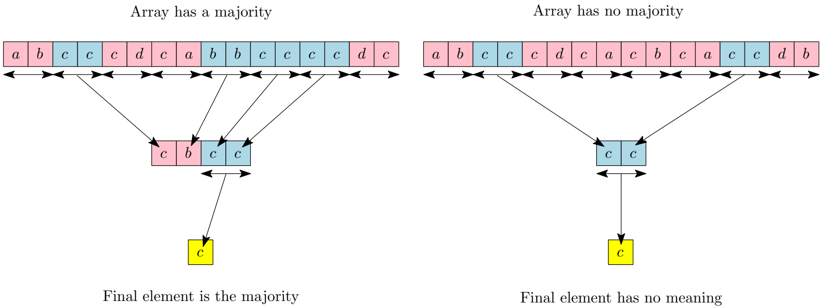
Remove two unequal elements from A to get B

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

Solutions → BoyerMoore-Multipass

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

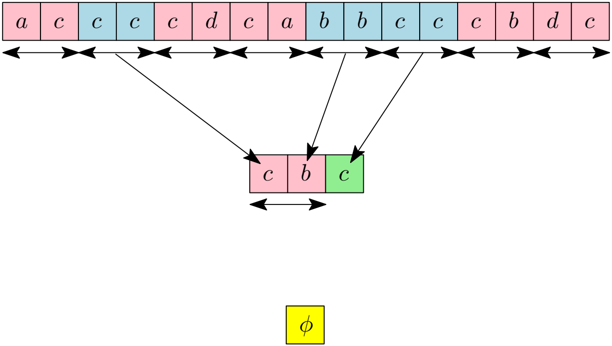
MAJORITY-MULTIPASS($A[1 \ldots n], tiebreaker$)

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 $n \bmod 2 = 1$ **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

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$$

Solutions → BoyerMoore-Multipass

Array has a majority



$tiebreaker = -1$
candidate $= c$

$tiebreaker = c$
candidate $= c$

$tiebreaker = c$
candidate $= c$

Solutions $\rightarrow$ BoyerMoore-Twopass

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.Top()
  |  |  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.Top()
  count ← COUNTOCCURRENCES(A[1 ... n], C)
  if count > ⌊n/2⌋ then  return C
  return −1
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \mathcal{O}(n) \rangle$$

Solutions $\rightarrow$ BoyerMoore-Twopass

i	$A[i]$	S
1	a	$[a]$
2	a	$[a, a]$
3	a	$[a, a, a]$
4	b	$[a, a]$
5	b	$[a]$
6	b	ϕ
7	b	$[b]$

i	$A[i]$	S
1	a	$[a]$
2	b	ϕ
3	a	$[a]$
4	b	ϕ
5	a	$[a]$
6	b	ϕ
7	c	$[c]$

i	$A[i]$	S
1	a	$[a]$
2	b	ϕ
3	a	$[a]$
4	b	ϕ
5	a	$[a]$
6	b	ϕ

Solutions $\rightarrow$ BoyerMoore-Twopass-Inplace

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 ith element is same as C. If same, then increment m, else, decrement m.
4. If m is positive, then check if C is majority

Solutions → BoyerMoore-Twopass-Inplace

```
MAJORITY-BOYERMOORE-TWOPASS-INPLACE(A[1...n])
C ← A[1]; m ← 1
// Stage 1. Eliminate all except one candidate c
for i ← 2 to n do
  | if m = 0 then
  | | { C ← A[i]; m ← 1 }
  | else
  | | if C = A[i] then  m ← m + 1
  | | else  m ← m − 1
// Stage 2. Check whether c is the majority
if m ≠ 0 then
  | count ← COUNTOCCURRENCES(A[1...n], C)
  | if count > ⌊n/2⌋ then  return C
return −1
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Solutions $\rightarrow$ BoyerMoore-Twopass-Inplace

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 ith element is same as C. If same, then increment m, else, decrement m.
4. If m is positive, then check if C is majority

i	$A[i]$	C	m
1	a	a	1
2	a	a	2
3	a	a	3
4	b	b	2
5	b	b	1
6	b	b	0
7	b	b	1

i	$A[i]$	C	m
1	a	a	1
2	b	a	0
3	a	a	1
4	b	a	0
5	a	a	1
6	b	a	0
7	c	c	1

i	$A[i]$	C	m
1	a	a	1
2	b	a	0
3	a	a	1
4	b	a	0
5	a	a	1
6	b	a	0

Solutions $\rightarrow$ FischerSalzberg

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

```
// Stage 1. Find the majority candidate C
Create two stacks S1 and S2
for i ← 1 to n do
|  if S1 is empty or S1.Top() ≠ A[i] then
|  |  S1.Push(A[i])  if S2 is not empty then  S1.Push(S2.Pop())
|  else  S2.Push(A[i])
C ← S1.Top()

// Stage 2. Confirm if the candidate is the majority
while S1 is not empty do
|  item ← S1.Pop()
|  if item = C then
|  |  if S1 is empty then  S2.Push(C)
|  |  else  S1.Pop()
|  else
|  |  if S2 is empty then  return −1
|  |  else  S2.Pop()
if S2 is not empty then  return C
return −1
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$$

Solutions → FischerSalzberg

i	$A[i]$	S_1	S_2
1	a	$[a]$	ϕ
2	a	$[a]$	$[a]$
3	a	$[a]$	$[a, a]$
4	b	$[a, b, a]$	$[a]$
5	b	$[a, b, a, b, a]$	ϕ
6	b	$[a, b, a, b, a, b]$	ϕ
7	b	$[a, b, a, b, a, b]$	$[b]$
		$[a, b, a, b]$	$[b]$
		$[a, b]$	$[b]$
		ϕ	$[b]$

i	$A[i]$	S_1	S_2
1	a	$[a]$	ϕ
2	b	$[a, b]$	ϕ
3	a	$[a, b, a]$	ϕ
4	b	$[a, b, a, b]$	ϕ
5	a	$[a, b, a, b, a]$	ϕ
6	b	$[a, b, a, b, a, b]$	ϕ
7	c	$[a, b, a, b, a, b, c]$	ϕ
		$[a, b, a, b, a]$	ϕ
		$[a, b, a, b]$	ϕ

i	$A[i]$	S_1	S_2
1	a	$[a]$	ϕ
2	b	$[a, b]$	ϕ
3	a	$[a, b, a]$	ϕ
4	b	$[a, b, a, b]$	ϕ
5	a	$[a, b, a, b, a]$	ϕ
6	b	$[a, b, a, b, a, b]$	ϕ
		$[a, b, a, b]$	ϕ
		$[a, b]$	ϕ
		ϕ	ϕ

Complexity

Algorithm	Time	Extra Space	Comments
Brute force	$\Theta\left(n^2\right)$	$\Theta(1)$	$-$
Sorting-based	$\Theta(n \log n)$	$\Theta(n)$	Can't solve.
Divide-and-conquer	$\Theta(n \log n)$	$\Theta(\log n)$	$-$
Probabilistic	$\Theta\left(n \log \frac{1}{\epsilon}\right)$	$\Theta(1)$	Can't solve. Success $> 1 - \epsilon$.
Hashing-based	$\Theta(n)$	$\Theta(n)$	Can't solve.
Median-based	$\Theta(n)$	$\Theta(n)$	Can't solve.
BoyerMoore multipass	$\Theta(n)$	$\Theta(n)$	$-$
BoyerMoore twopass	$\Theta(n)$	$\mathcal{O}(n)$	$-$
BoyerMoore twopass inplace	$\Theta(n)$	$\Theta(1)$	$-$
FischerSalzberg	$\Theta(n)$	$\Theta(n)$	$-$

References

- Puzzle book

Longest Palindromic Substring

Problem

Problem

- Design an algorithm to compute a longest palindromic substring of a string.

- Input: $s = bababaeabaed$
  Output: $s[4, 10] = abaeaba$

Core data structures

- $oddp[i]$ = largest radius of odd-sized palindrome centered at index i

$s[i]$	$\cdots$	a	t	v	k	z	d	z	k	v	t	x	$\cdots$
$oddp[i]$							4						

- $evenp[i]$ = largest radius of even-sized palindrome centered at indices i and $i+1$

$s[i]$	$\cdots$	a	t	v	k	z	z	k	v	t	x	$\cdots$
$evenp[i]$						4						

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 $s[(i - evenp[i] + 1) \ldots (i + evenp[i])]$

Goal

$\boxed{\text{Compute } oddp[1 \ldots n] \text{ and } evenp[1 \ldots n]}$

$\boxed{\text{Computing the LPS from } oddp[1 \ldots n] \text{ and } evenp[1 \ldots n] \text{ is easy}}$

Solutions → Brute force algorithm

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] ← [0...0] and evenp[1...n] ← [0...0]
for i ← 1 to n do
    for j ← i to n do
        if isPalindrome(s[i...j]) then
            length ← j − i + 1                 // palindrome length
            radius ← ⌊length/2⌋                // palindrome radius
            center ← ⌊(i+j)/2⌋                 // palindrome center
            if length is odd then
                oddp[center] ← Max(oddp[center], radius)
            else
                evenp[center] ← 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
    ℓ ← i − oddp[i] − 1
    r ← i + oddp[i] + 1
    while ℓ ≥ 1 and r ≤ n and s[ℓ] = s[r] do
        oddp[i] ← oddp[i] + 1
        ℓ ← ℓ − 1
        r ← r + 1
// For each palindrome center i, compute evenp[i]
for i ← 1 to n do
    ℓ ← i − evenp[i]
    r ← i + evenp[i] + 1
    while ℓ ≥ 1 and r ≤ n and s[ℓ] = s[r] do
        evenp[i] ← evenp[i] + 1
        ℓ ← ℓ − 1
        r ← r + 1
return (oddp, evenp)
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n^2\right), \Theta\left(n\right) \right\rangle$$

Solutions → Rabin-Karp + Binary search

```
BinarySearch(s[1 ... n])
low ← 1; high ← n
longestpalindrome ← empty string
while low ≤ high do
    mid ← ⌊(high−low)/2⌋
    palindrome ← 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 ← mid − 1
    // There is a palindrome with size mid, maybe it is too small
    // So check for palindromes with larger sizes
    else
        longestpalindrome ← palindrome
        low ← mid + 1
return longestpalindrome
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n \cdot \log n\right) \text{w.h.p}, \Theta\left(1\right) \rangle$$

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n^2 \log n\right), \Theta\left(1\right) \right\rangle$$

Solutions $\rightarrow$ Rabin-Karp + Binary search

```
FindPalindrome(s[1...n], m)

// Step 1: Initialize parameters and variables
p ← RandomLargePrime([1...nm^2])
b ← Size of ASCII set
h ← b^(m−1) mod p, r ← Reverse of string s
hash1 ← Hash(s[1...m], b, p)
hash2 ← Hash(r[n − m + 1...n], b, p)

// Step 2: Create a rolling hash for substrings of size m
for i ← 1 to (n − m + 1) do
    j ← n − m − i + 2                              // index of substring in r
    // Probability of hash collision is less than 1/n
    if hash1 = hash2 and s[i...(i + m − 1)] = r[j...(j + m − 1)] then
        return s[i...(i + m − 1)]
    // Rolling hash: Compute hash value of the next text window using
       the current text window in Θ(1) time
    if i ≠ (n − m + 1) then
        hash1 ← RollingHash-LtoR(hash1, s[i] × h, s[i + m], b, p)
        hash2 ← RollingHash-RtoL(hash2, r[j+m−1], r[i−1] × h, b, p)
return empty string
```

Solutions → Rabin-Karp + Binary search

RollingHash-LtoR($hash, subtract, add, b, p$)
$hash \leftarrow ((hash - subtract) \times b + add) \bmod p$ **return** $hash$

RollingHash-RtoL($hash, subtract, add, b, p$)
$hash \leftarrow \left((hash - subtract) \times b^{-1} + add\right) \bmod p$ **return** $hash$

Time $= \Theta(1)$

b^{-1} is modulo multiplicative inverse of b

b^{-1} always exists if the modulus is w.r.t a prime

b^{-1} can be computed using extended Euclidean algorithm

More information: Modulo multiplicative inverse

Solutions → Manacher's algorithm

- This algorithm is an optimization over $\mathcal{O}\left(n^2\right)$ 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

Solutions $\rightarrow$ Manacher's algorithm

indices	1	2	3	4	5	6	7	8	9	10	11	12
$s[i]$	b	a	b	a	b	a	e	a	b	a	e	d
positions				L	j				i	R		
$oddp[i]$	0	1	2	2	1	0	3	0				

- Suppose we have $oddp[1 \dots 8]$; we want to compute $oddp[9]$
- Let $s[L \dots R] =$ palindromic substring that ends as far to the right as possible so far
- For $i = 9$, $s[L \dots R] = s[4 \dots 10] = abaeaba$ 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 $oddp[$mirror image of $i]$ to compute $oddp[i]$

Solutions → Manacher's algorithm

indices	1	2	3	4	5	6	7	8	9	10	11	12
$s[i]$	b	a	b	a	b	a	e	a	b	a	e	d
positions				L	j				i	R		
$oddp[i]$	0	1	2	2	1	0	3	0				

- 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 $oddp[$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 $s[L \ldots R]$.

Solutions → Manacher's algorithm

Case 1. $i \geq R$

	1	2	3	4	5	6	7	8	9	10	11	12
indices	1	2	3	4	5	6	7	8	9	10	11	12
$s[i]$	b	a	e	a	b	a	e	a	c	a	e	a
positions					j	L		R	i			
$oddp[i]$	0	0	2	0	4	0	1	0				

In this case, compute $oddp[i]$ using standard algorithm logic

Solutions → Manacher's algorithm

Case 1. $i \geq R$

indices	1	2	3	4	5	6	7	8	9	10	11	12
$s[i]$	b	a	e	a	b	a	e	a	c	a	e	a
positions					j	L		R	i			
$oddp[i]$	0	0	2	0	4	0	1	0				

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

Solutions $\rightarrow$ Manacher's algorithm

Case 2. $i < R$ and largest palindrome at center j is completely inside $s[L \ldots R]$

indices	1	2	3	4	5	6	7	8	9	10	11	12
$s[i]$	b	a	b	a	b	a	e	a	b	a	e	d
positions				L	j				i	R		
$oddp[i]$	0	1	2	2	1	0	3	0				

In this case, set $oddp[i]$ to $oddp[j]$, and extend $oddp[i]$

Solutions $\rightarrow$ Manacher's algorithm

Case 2. $i < R$ and largest palindrome at center j is completely inside $s[L \dots R]$

indices	1	2	3	4	5	6	7	8	9	10	11	12
$s[i]$	b	a	b	a	b	a	e	a	b	a	e	d
positions				L	j				i	R		
$oddp[i]$	0	1	2	2	1	0	3	0				

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 $s[L \dots R]$, i.e., $s[4 \dots 6]$ is inside $s[4 \dots 10]$
- Therefore, set $oddp[9]$ to $oddp[5]$, i.e., $oddp[9] = 1$, and extend $oddp[9]$ as per standard algorithm logic
- We have $oddp[9] = 2$. We save one computation

Solutions → Manacher's algorithm

Case 3. $i < R$ and largest palindrome at center j is not completely inside $s[L \ldots R]$

indices	1	2	3	4	5	6	7	8	9	10	11	12
$s[i]$	b	a	e	a	b	a	e	a	b	a	b	d
positions				L	j				i	R		
$oddp[i]$	0	0	2	0	4	0	3	0				

In this case, set $oddp[i]$ to $(R - i)$, and extend $oddp[i]$

Solutions → Manacher's algorithm

Case 3. $i < R$ and largest palindrome at center j is

not completely inside $s[L \dots R]$

indices	1	2	3	4	5	6	7	8	9	10	11	12
$s[i]$	b	a	e	a	b	a	e	a	b	a	b	d
positions				L	j				i	R		
$oddp[i]$	0	0	2	0	4	0	3	0				

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 $s[L \dots R]$, i.e., $s[1 \dots 9]$ is not completely inside $s[4 \dots 10]$
- 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

```
ManacherOdd(s[1 ... n])

Create array oddp[1 ... n] ← [0 ... 0]
L ← 1
R ← −1
for i ← 1 to n do
    if i < R then
        oddp[i] ← Min(oddp[L + R − i], R − i)
    ℓ ← i − oddp[i] − 1
    r ← i + oddp[i] + 1
    while ℓ ≥ 1 and r ≤ n and s[ℓ] = s[r] do
        oddp[i] ← oddp[i] + 1
        ℓ ← ℓ − 1
        r ← r + 1
    if i + oddp[i] > R then
        L ← i − oddp[i]
        R ← i + oddp[i]
return oddp
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$$

Solutions → Manacher's algorithm

MANACHEREVEN($s[1 \dots n]$)

Create array $evenp[1 \dots n] \leftarrow [0 \dots 0]$
$L \leftarrow 1$
$R \leftarrow -1$
for $i \leftarrow 1$ **to** n **do**
  if $i < R$ **then**
    $evenp[i] \leftarrow$ 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**
    $evenp[i] \leftarrow evenp[i] + 1$
    $\ell \leftarrow \ell - 1$
    $r \leftarrow r + 1$
  if $i + evenp[i] > R$ **then**
    $L \leftarrow i - evenp[i] + 1$
    $R \leftarrow i + evenp[i]$
return $evenP$

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$$

Complexity

Algorithm	Time	Space	All pal. substr?
Brute force	$\mathcal{O}\left(n^3\right)$	$\Theta\left(n\right)$	✓
Standard algo	$\mathcal{O}\left(n^2\right)$	$\Theta\left(n\right)$	✓
Rabin-Karp + bin. search	$\mathcal{O}\left(n \log n\right)$ w.h.p	$\Theta\left(1\right)$	✗
	$\mathcal{O}\left(n^2 \log n\right)$	$\Theta\left(1\right)$	✗
Manacher	$\Theta\left(n\right)$	$\Theta\left(n\right)$	✓
Suffix trees	$\Theta\left(n\right)$	$\Theta\left(n\right)$	✓

Selection Two Sorted Arrays HOME

Problem

- Find the kth smallest element among two sorted arrays $A[1 \dots m]$ and $B[1 \dots n]$, 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

Solutions $\to$ Concatenate and sort

1. Concatenate the two arrays and sort it
2. Return the kth smallest element

$\textsc{Selection-Concatenate\&Sort}(A[1\ldots m], B[1\ldots n], k)$	
Create an array $M[1\ldots(m+n)]$	
$M[1\ldots m] \leftarrow A[1\ldots m]$	// Copy the first array to M
$M[(m+1)\ldots(m+n)] \leftarrow B[1\ldots n]$	// Copy the second array to M
$\textsc{Sort}(M[1\ldots(m+n)])$	// Sort M
return $M[k]$	// Return the kth 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 $\rightarrow$ Concatenate and heapify

1. Concatenate the two arrays and build a heap
2. Return the kth smallest element

$\textsc{Selection-Concatenate\&Heapify}(A[1\ldots m], B[1\ldots n], k)$

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

$\textsc{Heapify}(H[1\ldots(m+n)])$ // Construct heap from H in linear time

// RemoveMin from H for a total of k times
for $i \leftarrow 1$ **to** k **do**
| $result \leftarrow H$.RemoveMin()

return $result$

$$\langle \text{Time, Space} \rangle = \langle \Theta\left((m+n) + k\log(m+n)\right), \Theta\left(m+n\right) \rangle$$

Solutions $\rightarrow$ Merge

1. Merge the two sorted arrays
2. Return the kth smallest element

```
SELECTION-MERGE(A[1...m], B[1...n], k)

i ← 1; j ← 1; ℓ ← 1
Create an array M[1...(m+n)]

// Merge the two sorted arrays to M until an array becomes empty
while i ≤ m and j ≤ n do
    if A[i] ≤ B[j] then { M[ℓ] ← A[i]; i ← i+1 }
    else { M[ℓ] ← B[j]; j ← j+1 }
    ℓ ← ℓ+1

// Copy the remaining elements to M
if i > m then M[ℓ...(m+n)] ← B[j...n]
else if j > n then M[ℓ...(m+n)] ← A[i...m]

// Return the kth smallest element of M
return M[k]
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(m+n), \Theta(m+n) \rangle$$

Solutions $\rightarrow$ Merge optimized

1. Merge the two sorted arrays without using extra space to find the kth smallest element

```
SELECTION-MERGEOPTIMIZED(A[1...m], B[1...n], k)
i ← 1; j ← 1
// Iterate for k elements
while k > 0 do
    // If one of the arrays is reached
    if i > m then return B[j + k − 1]
    if j > n then return A[i + k − 1]

    // Merge-like operations
    if A[i] < B[j] then i ← i + 1
    else if A[i] ≥ B[j] then j ← j + 1

    k ← k − 1
return min(A[i], B[j])
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(k), \Theta(1) \rangle$$

Solutions → Decrease-and-conquer (recursive)

1. Find the middle indices $mid1$ and $mid2$ of the two arrays
2. Compare $mid1 + mid2$ and k and compare $A[mid1]$ and $B[mid2]$. Recursively call the algorithm on a smaller subproblem depending on the four cases.

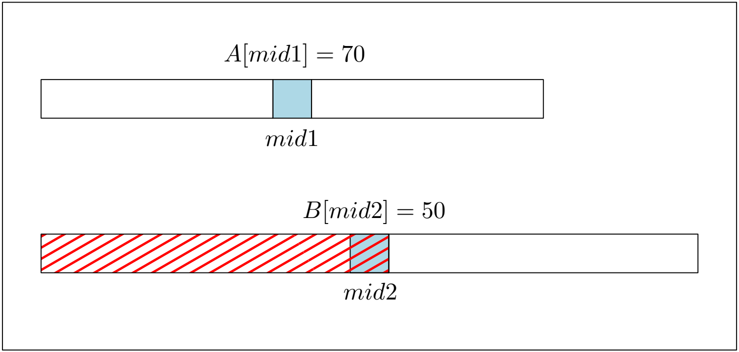
Solutions $\rightarrow$ Decrease-and-conquer (recursive)

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

Solutions → Decrease-and-conquer (recursive)

- Case 1: $mid1 + mid2 < k$ and $A[mid1] > B[mid2]$



kth smallest can't be in the first half of the second array B

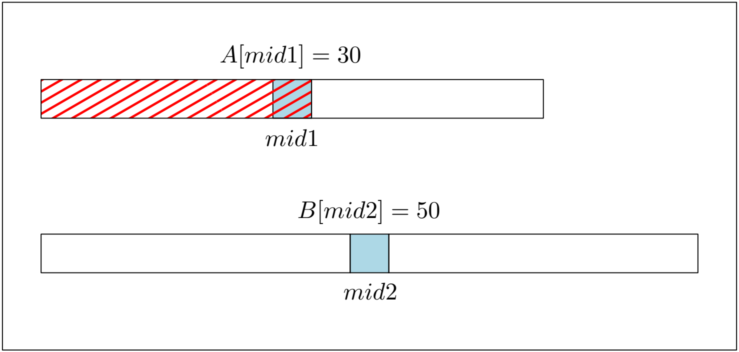
Eliminate the first half of the second array B

Find and return $(k - mid2)$th smallest in the remaining elements

$\text{SELECTION-DE\&C}(A, B[(mid2 + 1) \ldots n], k - mid2)$

Solutions → Decrease-and-conquer (recursive)

- Case 2: $mid1 + mid2 < k$ and $A[mid1] \leq B[mid2]$



kth smallest can't be in the first half of the first array A

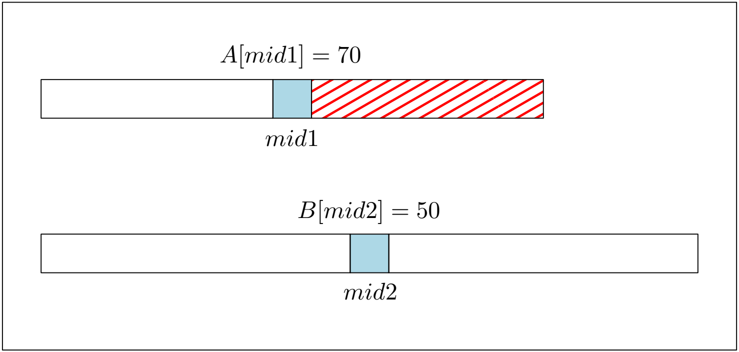
Eliminate the first half of the first array A

Find and return $(k - mid1)$th smallest in the remaining elements

$\textsc{Selection-De\&C}(A[(mid1 + 1) \ldots m], B, k - mid1)$

Solutions $\rightarrow$ Decrease-and-conquer (recursive)

- Case 3: $mid1 + mid2 \geq k$ and $A[mid1] > B[mid2]$



kth smallest can't be in the second half of the first array A

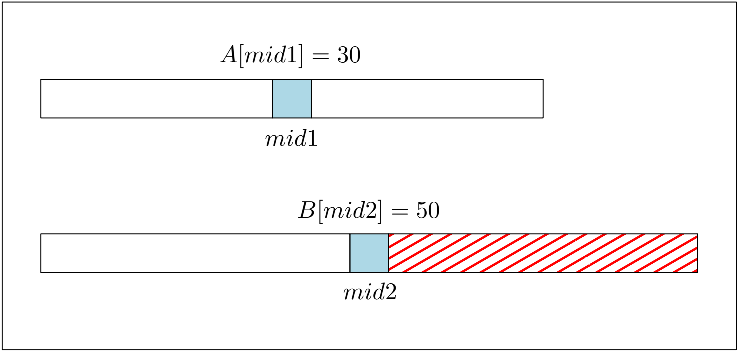
Eliminate the second half of the first array A

Find and return kth smallest in the remaining elements

$\textsc{Selection-De\&C}(A[1 \ldots mid1], B, k)$

Solutions → Decrease-and-conquer (recursive)

- Case 4: $mid1 + mid2 \geq k$ and $A[mid1] \leq B[mid2]$



kth 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 kth smallest in the remaining elements

$\textsc{Selection-De\&C}(A, B[1 \ldots mid2], k)$

Solutions $\rightarrow$ Decrease-and-conquer (recursive)

$\textsc{Selection-De\&C}(A[1 \ldots m], B[1 \ldots n], k)$

// If an array is empty, return kth 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
$mid1 \leftarrow \lfloor m/2 \rfloor$; $mid2 \leftarrow \lfloor n/2 \rfloor$

if $mid1 + mid2 < k$ **and** $A[mid1] > B[mid2]$ **then**
  // kth smallest can't be in the first half of the second array B
  return $\textsc{Selection-De\&C}(A, B[(mid2 + 1) \ldots n], k - mid2)$
else if $mid1 + mid2 < k$ **and** $A[mid1] \leq B[mid2]$ **then**
  // kth smallest can't be in the first half of the first array A
  return $\textsc{Selection-De\&C}(A[(mid1 + 1) \ldots m], B, k - mid1)$
else if $mid1 + mid2 \geq k$ **and** $A[mid1] > B[mid2]$ **then**
  // kth smallest can't be in the second half of the first array A
  return $\textsc{Selection-De\&C}(A[1 \ldots mid1], B, k)$
else if $mid1 + mid2 \geq k$ **and** $A[mid1] \leq B[mid2]$ **then**
  // kth smallest can't be in the second half of the second array B
  return $\textsc{Selection-De\&C}(A, B[1 \ldots mid2], k)$

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(\log(m+n)), \mathcal{O}(\log(m+n)) \rangle$$

Solutions $\rightarrow$ Binary search (recursive)

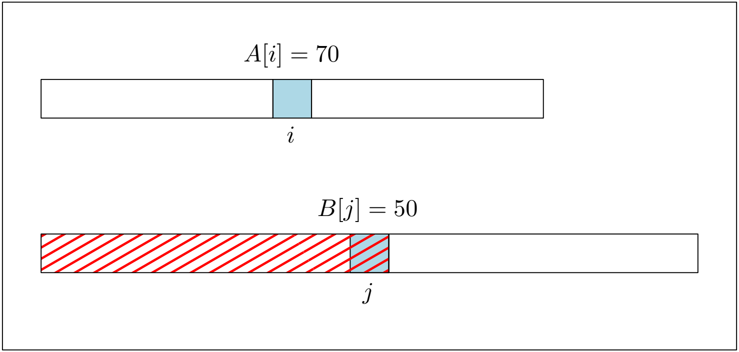
1. Select index i in the first array A, where $i = \min(m, \lfloor k/2 \rfloor)$
2. Select index j in the second array B, where $j = k - i$
3. Compare $A[i]$ and $B[j]$
4. If $A[i] > B[j]$, discard the first j elements of B
   Find the $(k - j)$th smallest element recursively
5. If $A[i] \leq B[j]$, 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 $\rightarrow$ Binary search (recursive)

- Case 1: $i + j \leq k$ and $A[i] > B[j]$



kth smallest can't be in the first j elements of the second array B

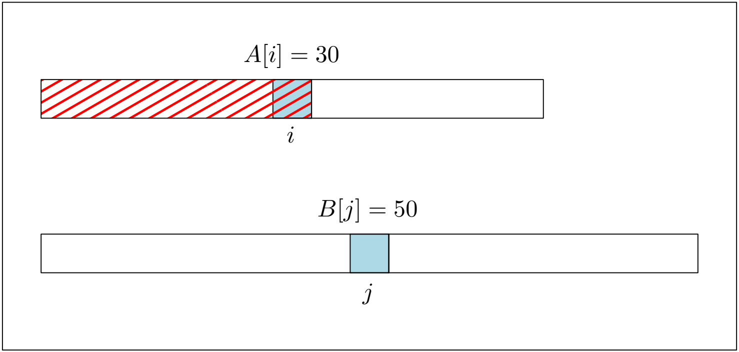
Eliminate the first j elements of the second array B

Find and return $(k - j)$th smallest in the remaining elements

$\textsc{Selection-BinarySearch}(A, B[j + 1 \ldots n], k - j)$

Solutions → Binary search (recursive)

- Case 2: $i + j \leq k$ and $A[i] \leq B[j]$



kth 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

$\textsc{Selection-BinarySearch}(A[i+1 \ldots n], B, k-i)$

Solutions $\rightarrow$ Binary search (recursive)

```
SELECTION-BINARYSEARCH(A[1...m], B[1...n], k)

  // First array should be the shorter of the two arrays
  if m > n then
  |  return SELECTION-BINARYSEARCH(B[1...n], A[1...m], k)

  // If first array is empty, return the kth 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 ← min(m, ⌊k/2⌋); j ← 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...n], k − j)
  // If A[i] ≤ B[j], then discard the first i elements of A
  else if A[i] ≤ B[j] then
  |  return SELECTION-BINARYSEARCH(A[i+1...n], B, k − i)
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(\log k\right), \Theta\left(\log k\right) \rangle$$

Complexity

Algorithm	Time	Extra Space
Concatenate-sort	$\Theta\left((m+n)\log(m+n)\right)$	$\Theta\left(m+n\right)$
Concatenate-heapify	$\Theta\left((m+n)+k\log(m+n)\right)$	$\Theta\left(m+n\right)$
Merge	$\Theta\left(m+n\right)$	$\Theta\left(m+n\right)$
Merge optimized	$\Theta\left(k\right)$	$\Theta\left(1\right)$
De&C (recursive)	$\Theta\left(\log(m+n)\right)$	$\Theta\left(\log(m+n)\right)$
Binary search (recursive)	$\Theta\left(\log k\right)$	$\Theta\left(\log k\right)$

Largest Subarray Sum HOME

Problem

- Given an array of reals, find the subarray with the largest sum, and return its sum.
- Input: $[-2, 1, -3, \underbrace{4, -1, 2, 1}, -5, 4]$

  Output: 6
- Input: $[\underbrace{5, 4, -1, 7, 8}]$

  Output: 23

Solutions → Brute force

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

```
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
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^3\right), \Theta\left(1\right) \right\rangle$$

Solutions → Optimized brute force

- For all subarrays, calculate the sum of the subarray as you travel the array.

OptimizedBruteForce($A[1 \dots n]$)

```
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
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^2\right), \Theta\left(1\right) \right\rangle$$

Solutions → Optimized brute force

i	j	Array	sum	max	
-5		-5, 4, -1, 7	0	$-\infty$	sum = 0; $max = -\infty$
-5	-5	-5, 4, -1, 7	-5	-5	sum = -5; As $sum > max$, $max = sum$
-5	4	-5, 4, -1, 7	-1	-1	sum = -1; As $sum > max$, $max = sum$
-5	-1	-5, 4, -1, 7	-2	-1	sum = -2; As $sum \leq max$, max is not updated
-5	7	-5, 4, -1, 7	5	5	sum = 5; As $sum > max$, $max = sum$
4		-5, 4, -1, 7	0	5	sum = 0
4	4	-5, 4, -1, 7	4	5	sum = 4; As $sum \leq max$, max is not updated
4	-1	-5, 4, -1, 7	3	5	sum = 3; As $sum \leq max$, max is not updated
4	7	-5, 4, -1, 7	10	10	sum = 10; As $sum > max$, $max = sum$
-1		-5, 4, -1, 7	0	10	sum = 0
-1	-1	-5, 4, -1, 7	-1	10	sum = -1; As $sum \leq max$, max is not updated
-1	7	-5, 4, -1, 7	6	10	sum = 6; As $sum \leq max$, max is not updated
7		-5, 4, -1, 7	0	10	sum = 0
7	7	-5, 4, -1, 7	7	10	sum = 7; As $sum \leq max$, max is not updated
				10	Largest Subarray Sum

Solutions $\rightarrow$ Divide-and-conquer

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

$\textsc{DivideAndConquer}(A[1 \ldots n])$

if $n = 1$ **then** **return** $A[1]$
$mid \leftarrow \left\lfloor \frac{n}{2} \right\rfloor$
$S_{\text{left}} \leftarrow \textsc{DivideAndConquer}(A[1 \ldots mid])$
$S_{\text{right}} \leftarrow \textsc{DivideAndConquer}(A[mid + 1 \ldots n])$
$S_{\text{merge}} \leftarrow \textsc{Merge}(A[1 \ldots n], mid)$
return Max$(S_{\text{left}}, S_{\text{right}}, S_{\text{merge}})$

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n \log n\right), \Theta\left(\log n\right) \rangle$$

Solutions → Divide-and-conquer

$\textsc{Merge}(A[1 \ldots n], mid)$

// Find the maximum suffix in the first half

$suffixmax \leftarrow -\infty$; $sum \leftarrow 0$

for $i \leftarrow mid$ **to** 1 **do**

  $sum \leftarrow sum + A[i]$

  if $sum > suffixmax$ **then** $suffixmax \leftarrow sum$

// Find the maximum prefix in the second half

$prefixmax \leftarrow -\infty$; $sum \leftarrow 0$

for $i \leftarrow mid + 1$ **to** n **do**

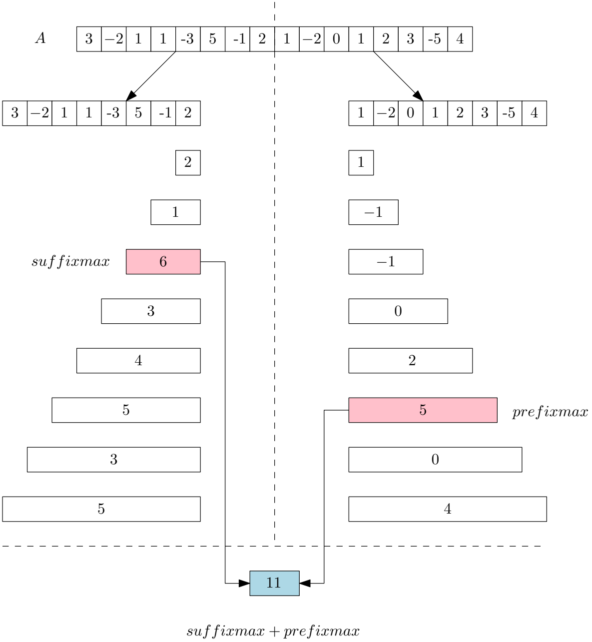
  $sum \leftarrow sum + A[i]$

  if $sum > prefixmax$ **then** $prefixmax \leftarrow sum$

// Max subarray sum so that subarray crosses the midpoint

return $(suffixmax + prefixmax)$

Solutions → Divide-and-conquer



Solutions → Improved divide-and-conquer

1. Create a class called Node for any subproblem (subarray)
2. For any subproblem/subarray, we define a node with four values: sum = largest subarray sum, $total$ = total subarray sum $prefixmax$ = max prefix sum, $suffixmax$ = max suffix sum
3. Compute and return the sum corresponding to the node of $A[1 \dots n]$ using D&C

$\textsc{ImprovedDivideAndConquer}(A[1 \dots n])$

Node $answer \leftarrow \textsc{GetMaxSumSubarray}(A[1 \dots n])$
return $answer.sum$

$\textsc{GetMaxSumSubarray}(A[low \dots high])$

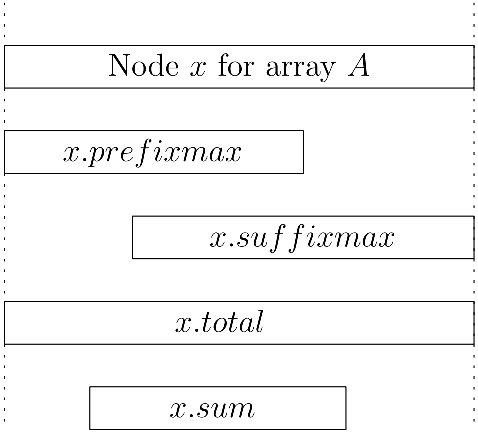
if $low = high$ **then** **return** $\textsc{Node}(A[low])$
$mid \leftarrow \lfloor (low + high)/2 \rfloor$
$node_\ell \leftarrow \textsc{GetMaxSumSubarray}(A[low \dots mid])$
$node_r \leftarrow \textsc{GetMaxSumSubarray}(A[mid + 1 \dots high])$
return $\textsc{Merge}(node_\ell, node_r)$

Solutions $\rightarrow$ Improved divide-and-conquer

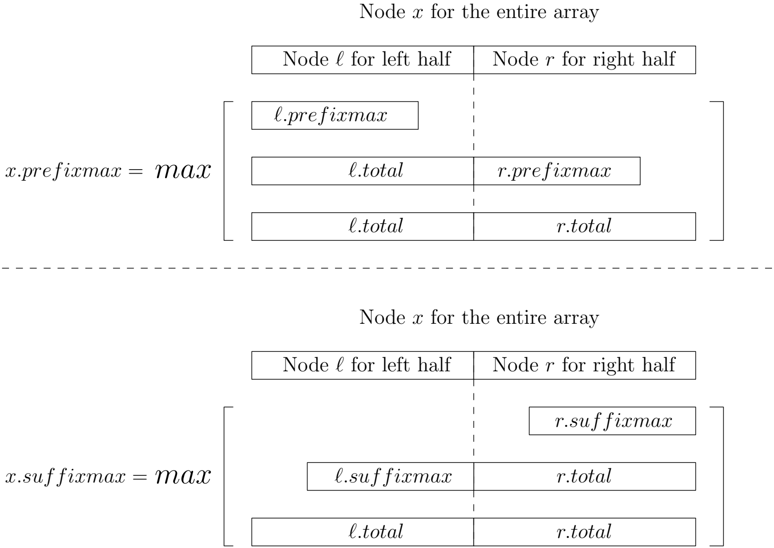
```
MERGE(ℓ, r)
x ← NODE(0)
  // Max prefix subarray sum
x.prefixmax ← MAX(ℓ.prefixmax, ℓ.total + r.prefixmax, ℓ.total + r.total)
  // Max suffix subarray sum
x.suffixmax ← MAX(r.suffixmax, r.total + ℓ.suffixmax, ℓ.total + r.total)
  // Total subarray sum
x.total ← ℓ.total + r.total
  // Max subarray sum
x.sum ← MAX(x.prefixmax, x.suffixmax, x.total, ℓ.sum, r.sum
                ℓ.suffixmax + r.prefixmax)
return x
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(\log n) \rangle$$

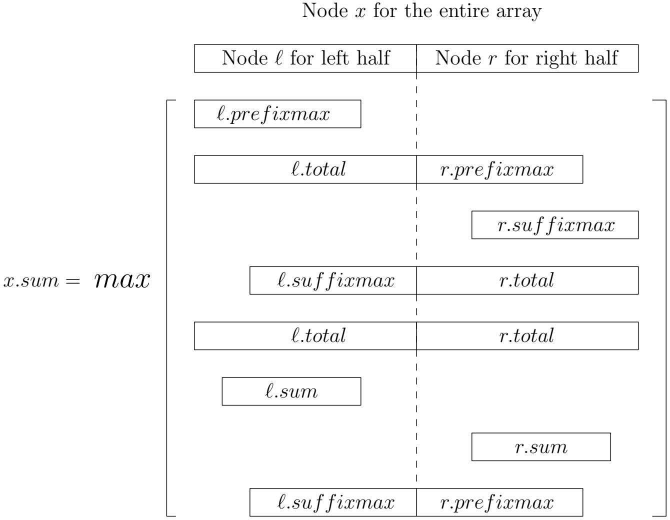
Solutions → Improved divide-and-conquer



Solutions → Improved divide-and-conquer



Solutions → Improved divide-and-conquer



Solutions $\rightarrow$ Kadane's algorithm

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.

$\textsc{KadaneAlgorithm}(A[1 \ldots n])$

$max \leftarrow A[1]$; $sum \leftarrow A[1]$
for $i \leftarrow 2$ **to** n **do**
    // If sum is negative, throw it away. Otherwise, keep adding to it.
    $sum \leftarrow \text{Max}(A[i], sum + A[i])$
    $max \leftarrow \text{Max}(max, sum)$
return max

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Solutions → Kadane's algorithm

								sum	max	
								-2	-2	Initialize sum & max to the first element
-2	**-3** (i)	4	-1	-2	1	5	-3	-3	-2	As $arr[i] > sum + arr[i]$, $sum = arr[i]$; As $sum < max$, max isn't updated
-2	-3	**4** (i)	-1	-2	1	5	-3	4	4	As $arr[i] > sum + arr[i]$, $sum = arr[i]$; As $sum > max$, $max = sum$
-2	-3	4	**-1** (i)	-2	1	5	-3	3	4	As $arr[i] < sum + arr[i]$, $sum \mathrel{+}= arr[i]$; As $sum < max$, max isn't updated
-2	-3	4	-1	**-2** (i)	1	5	-3	1	4	As $arr[i] < sum + arr[i]$, $sum \mathrel{+}= arr[i]$; As $sum < max$, max isn't updated
-2	-3	4	-1	-2	**1** (i)	5	-3	2	4	As $arr[i] < sum + arr[i]$, $sum \mathrel{+}= arr[i]$; As $sum < max$, max isn't updated
-2	-3	4	-1	-2	1	**5** (i)	-3	7	7	As $arr[i] < sum + arr[i]$, $sum \mathrel{+}= arr[i]$; As $sum > max$, $max = sum$
-2	-3	4	-1	-2	1	5	**-3** (i)	4	7	As $arr[i] < sum + arr[i]$, $sum \mathrel{+}= arr[i]$; As $sum < max$, max isn't updated

$\boxed{7}$ Largest Subarray Sum

Complexity

Algorithm	Time	Space
Brute force	$\Theta\left(n^3\right)$	$\Theta\left(1\right)$
Optimized brute force	$\Theta\left(n^2\right)$	$\Theta\left(1\right)$
Divide-and-conquer	$\Theta\left(n \log n\right)$	$\Theta\left(\log n\right)$
Improved divide-and-conquer	$\Theta\left(n\right)$	$\Theta\left(\log n\right)$
Kadane's algorithm	$\Theta\left(n\right)$	$\Theta\left(1\right)$

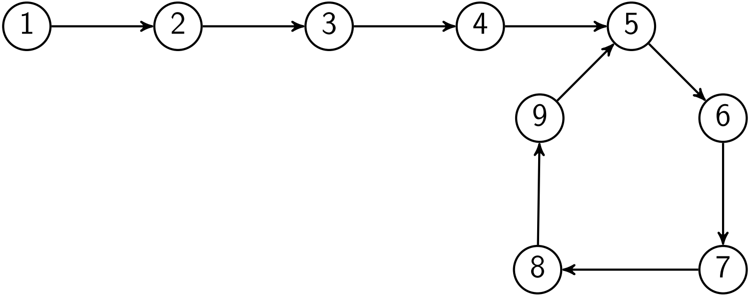
Loop in a Linked List HOME

Problem

- 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:



Output: (true, node 5, 5)

Solutions $\rightarrow$ Storing length

- 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($head$)

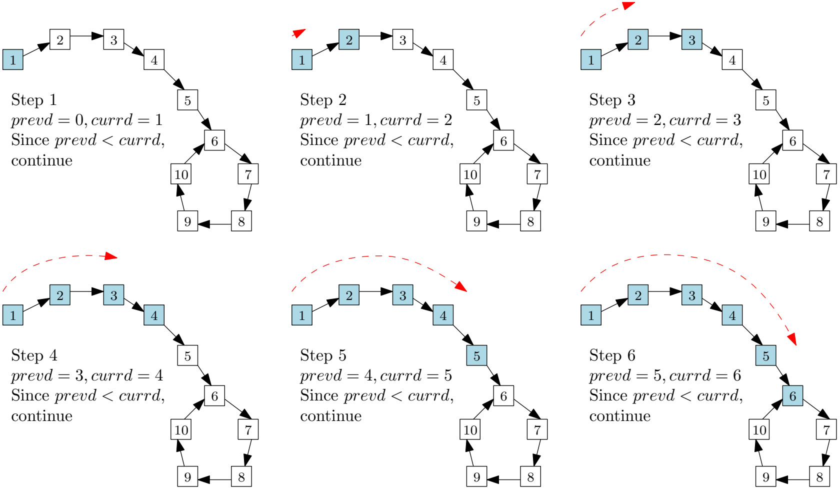
```
// Step 1. Create and initialize variables
curr ← head; prev ← head; currd ← 1; prevd ← 0
// Step 2. Move curr and prev pointers by one unit until end of SLL is reached
   or curr pointer is at a smaller distance than that of prev pointer from head
while curr ≠ null and currd > prevd do
|   prev ← 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($first, last$)

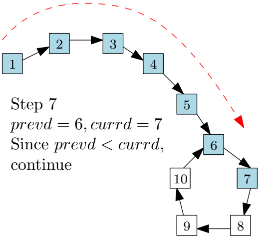
```
// Find the number of nodes between first and last pointers
count ← 1; current ← first
while current ≠ last do
|   count ← count + 1; current ← current.next
return count
```

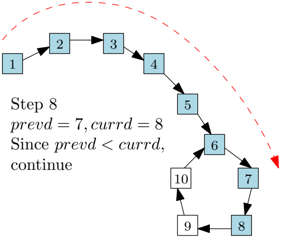
$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n^2\right), \mathcal{O}\left(1\right) \right\rangle$$

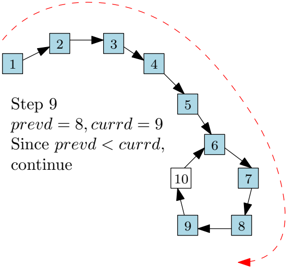
Solutions → Storing length

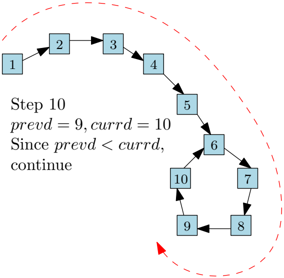


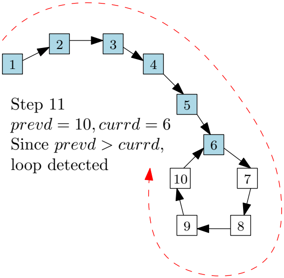
Solutions → Storing length











Solutions $\rightarrow$ Hashing

- 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.

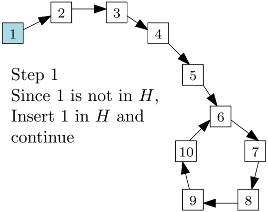
Solutions $\rightarrow$ Hashing

```
LoopInALinkedList(head)
Create a hash map H
count ← 0; curr ← head
// Traverse the singly linked list using the curr pointer
while curr ≠ null do
    // If curr node exists in the hash map, there is a loop
    if H.Contains(curr) then
        looplength ← (count − H[curr])
        return (true, curr, looplength)

    // If curr node is visited for the first time, add it to the map
    H.Add(curr, count)
    count ← count + 1; curr ← curr.next

// If the end of the linked list is reached, there is no loop
return (false, null, null)
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n), \mathcal{O}(n) \rangle$$

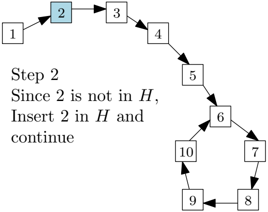
Solutions → Hashing

H: []



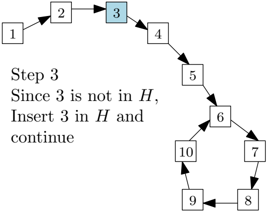
Step 1
Since 1 is not in H,
Insert 1 in H and
continue

H: [1]



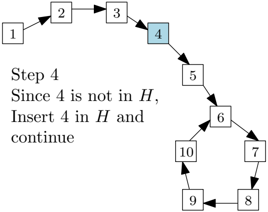
Step 2
Since 2 is not in H,
Insert 2 in H and
continue

H: [1, 2]



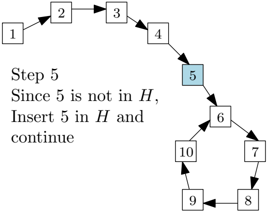
Step 3
Since 3 is not in H,
Insert 3 in H and
continue

H: [1, 2, 3]



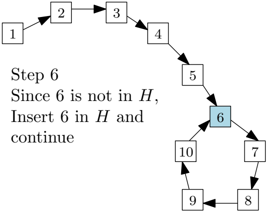
Step 4
Since 4 is not in H,
Insert 4 in H and
continue

H: [1, 2, 3, 4]



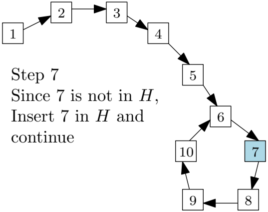
Step 5
Since 5 is not in H,
Insert 5 in H and
continue

H: [1, 2, 3, 4, 5]



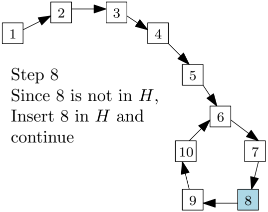
Step 6
Since 6 is not in H,
Insert 6 in H and
continue

H: [1, 2, 3, 4, 5, 6]



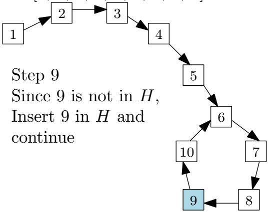
Step 7
Since 7 is not in H,
Insert 7 in H and
continue

H: [1, 2, 3, 4, 5, 6, 7, 8]



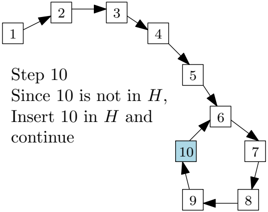
Step 8
Since 8 is not in H,
Insert 8 in H and
continue

H: [1, 2, 3, 4, 5, 6, 7, 8, 9]



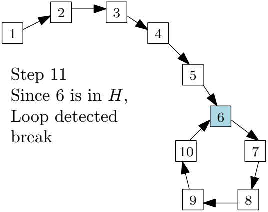
Step 9
Since 9 is not in H,
Insert 9 in H and
continue

H: [1, 2, 3, 4, 5, 6, 7, 8, 9,10]



Step 10
Since 10 is not in H,
Insert 10 in H and
continue

H: [1, 2, 3, 4, 5, 6, 7, 8, 9,10]



Step 11
Since 6 is in H,
Loop detected
break

Solutions → Slow and fast pointers

- 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:
  $looplength$: count the list length starting from and ending at $tortoise$
  $intersection$: start $tortoise$ from $head$ and $hare$ from $looplength$ distance from $head$; advance both pointers one node at a time until they meet; this node is the intersection node

Solutions → Slow and fast pointers

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
looplength ← LoopLength(tortoise)
intersection ← Intersection(head, looplength)
return (true, intersection, looplength)
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n), \mathcal{O}(1) \rangle$$

Solutions → Slow and fast pointers

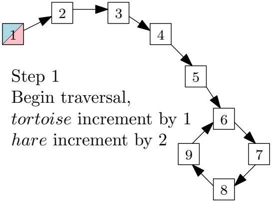
$\textsc{LoopLength}(curr)$

$looplength \leftarrow 1$; $loop \leftarrow curr.next$
while $loop \neq curr$ **do**
  $loop \leftarrow loop.next$
  $looplength \leftarrow looplength + 1$
return $looplength$

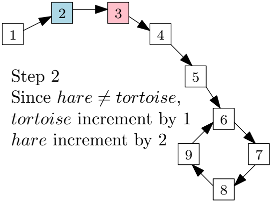
$\textsc{Intersection}(head, looplength)$

$tortoise \leftarrow head$; $hare \leftarrow head$
// $hare$ starts from $looplength$ distance from $head$
while $looplength > 0$ **do**
  $hare \leftarrow hare.next$
  $looplength \leftarrow looplength - 1$
// Advance both pointers and compare until match
while $tortoise \neq hare$ **do**
  $hare \leftarrow hare.next$
  $tortoise \leftarrow tortoise.next$
// Meeting node is the intersection node
return $tortoise$

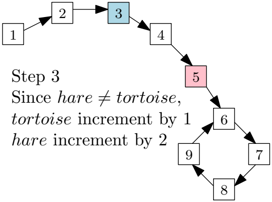
Solutions → Slow and fast pointers



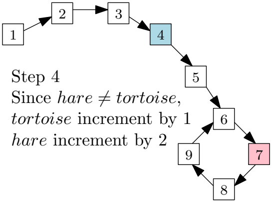
Step 1
Begin traversal,
tortoise increment by 1
hare increment by 2



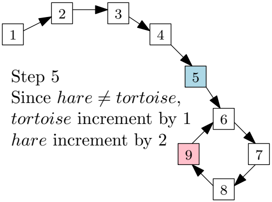
Step 2
Since *hare* $\neq$ *tortoise*,
tortoise increment by 1
hare increment by 2



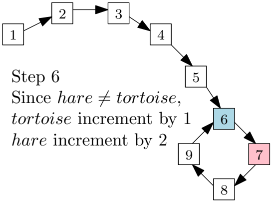
Step 3
Since *hare* $\neq$ *tortoise*,
tortoise increment by 1
hare increment by 2



Step 4
Since *hare* $\neq$ *tortoise*,
tortoise increment by 1
hare increment by 2

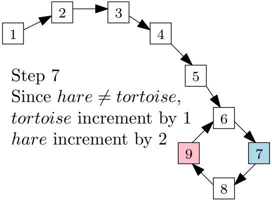


Step 5
Since *hare* $\neq$ *tortoise*,
tortoise increment by 1
hare increment by 2

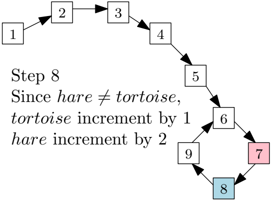


Step 6
Since *hare* $\neq$ *tortoise*,
tortoise increment by 1
hare increment by 2

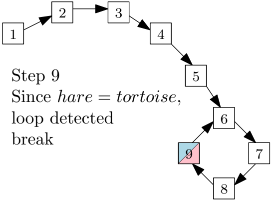
Solutions → Slow and fast pointers



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

Solutions $\rightarrow$ Slow and fast pointers alternative

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 $tortoise$

Solutions → Slow and fast pointers alternative

```
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)
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n), \mathcal{O}(1) \rangle$$

Solutions $\rightarrow$ Brent's algorithm

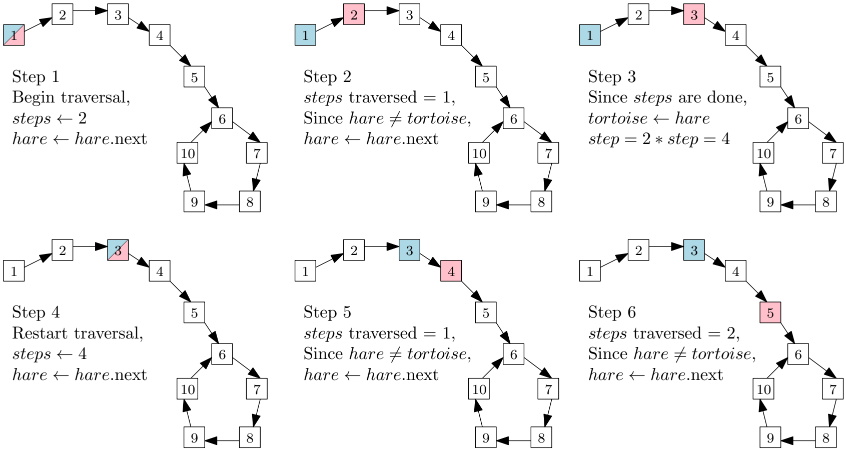
- Step 1. Use two pointers to scan the linked list:
  $tortoise$: slow pointer transports directly to fast pointer position
  $hare$: fast pointer moves $steps$ nodes at a time
  where, $steps$ is initially 2
- Step 2. Compare $tortoise$ and $hare$ for each step of $hare$ until $steps$
  Increase $steps$ to $\mathcal{F}(steps)$ (Brent used $\mathcal{F}(x) = 2x$)
  $tortoise$ 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 $looplength$ and $intersection$ as before

Solutions → Brent's algorithm

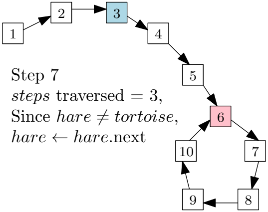
```
LoopInALinkedList(head)
  // Step 1. tortoise and hare are slow and fast pointers, respectively
  tortoise ← head; hare ← head; steps ← 1
  // Step 2. Scan the linked list and compare tortoise and hare. Advance
  //    hare by F(steps) at a time and tortoise to hare position.
  while true do
      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                        // transport tortoise to hare
      steps ← F(steps)                       // 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
  looplength ← LoopLength(tortoise)
  intersection ← Intersection(head, looplength)
  return (true, intersection, looplength)
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n), \mathcal{O}(1) \rangle$$

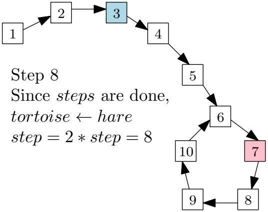
Solutions → Brent's algorithm



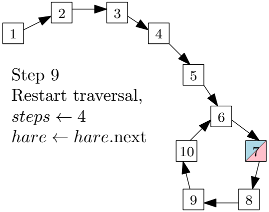
Solutions → Brent's algorithm



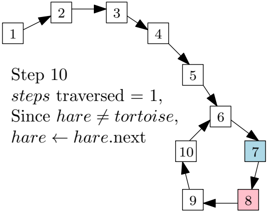
Step 7
steps traversed $= 3$,
Since *hare* $\neq$ *tortoise*,
hare $\leftarrow$ *hare*.next



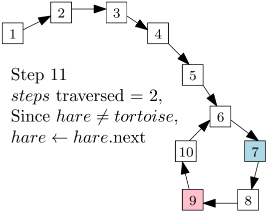
Step 8
Since *steps* are done,
tortoise $\leftarrow$ *hare*
step $= 2 * $ *step* $= 8$



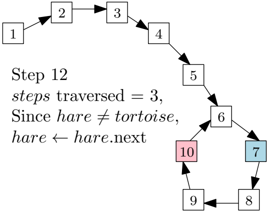
Step 9
Restart traversal,
steps $\leftarrow 4$
hare $\leftarrow$ *hare*.next



Step 10
steps traversed $= 1$,
Since *hare* $\neq$ *tortoise*,
hare $\leftarrow$ *hare*.next

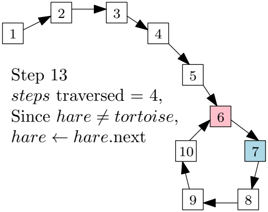


Step 11
steps traversed $= 2$,
Since *hare* $\neq$ *tortoise*,
hare $\leftarrow$ *hare*.next

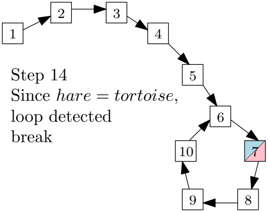


Step 12
steps traversed $= 3$,
Since *hare* $\neq$ *tortoise*,
hare $\leftarrow$ *hare*.next

Solutions $\rightarrow$ Brent's algorithm



Step 13
steps traversed $= 4$,
Since $hare \neq tortoise$,
$hare \leftarrow hare$.next



Step 14
Since $hare = tortoise$,
loop detected
break

Complexity

Algorithm	Time	Extra space
Storing length	$\mathcal{O}\left(n^2\right)$	$\mathcal{O}\left(1\right)$
Hashing	$\mathcal{O}\left(n\right)$	$\mathcal{O}\left(n\right)$
Slow-fast pointers	$\mathcal{O}\left(n\right)$	$\mathcal{O}\left(1\right)$
Slow-fast pointers alternative	$\mathcal{O}\left(n\right)$	$\mathcal{O}\left(1\right)$
Brent's algorithm	$\mathcal{O}\left(n\right)$	$\mathcal{O}\left(1\right)$

Y-shaped Linked List HOME

Problem

- 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., $head1$ and $head2$.

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 $\rightarrow$ 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 S1 and S2
ptr1 ← head1; ptr2 ← head2; result ← null
// Store all nodes of list 1 in stack 1 and list 2 in stack 2
while ptr1 ≠ null do { S1.Push(ptr1); ptr1 ← ptr1.next }
while ptr2 ≠ null do { S2.Push(ptr2); ptr2 ← ptr2.next }
// If the two stack tops are different then there is no intersection
if S1.Top() ≠ S2.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 S1 is not empty and S2 is not empty and S1.Top() = S2.Top() do
|  { result ← S1.Pop(); S2.Pop() }
// Return the intersecting node
return result
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(m+n), \Theta(m+n) \rangle$$

Solutions $\rightarrow$ 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

$\textsc{YShapedLinkedList-Hashset}(head1, head2)$

```
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
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(m+n), \Theta(m) \rangle$$

Solutions $\rightarrow$ Difference count

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
```

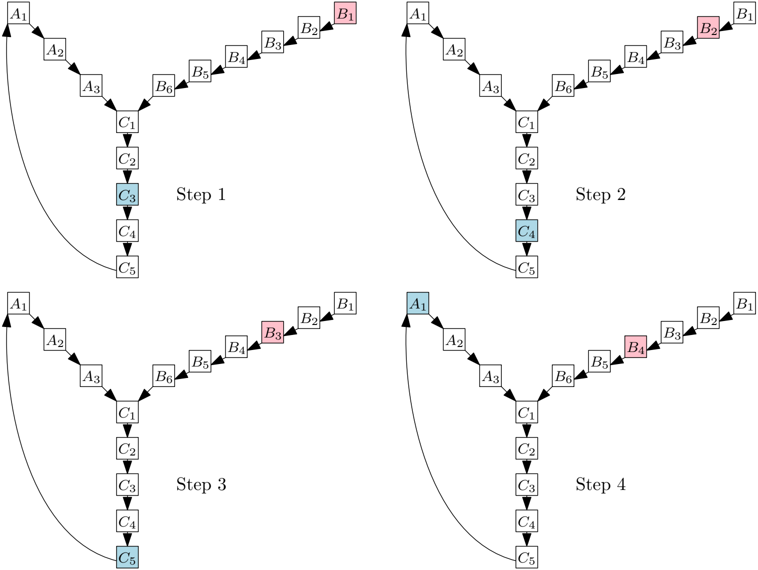
$$\langle \text{Time, Space} \rangle = \langle \Theta(m+n), \Theta(1) \rangle$$

Solutions → Loop in a linked list

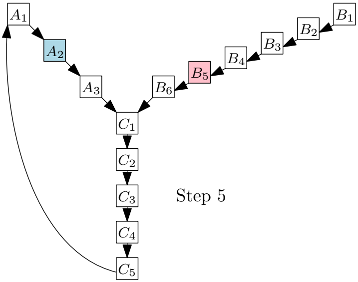
1. Traverse the first linked list, count the elements, and make a circular linked list. Remember the last node so that we can break the circle later on.
2. Transformed problem: Finding the loop in the second linked list.
3. Since we already know the length of the loop (size of the first linked list), 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.

Solutions → Loop in a linked list

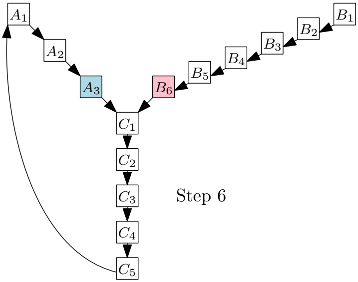
Length of the first list = length of the loop = 8
*pointer*1 (at C_3) is 8 steps ahead of *pointer*2 (at B_1)



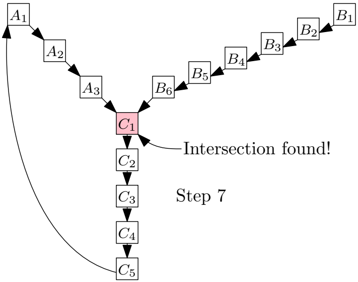
Solutions → Loop in a linked list



Step 5



Step 6



Step 7

Solutions → Loop in a linked list

$\textsc{YShapedLinkedList-LoopInALinkedList}(head1, head2)$

$pointer1 \leftarrow head1$; $pointer2 \leftarrow head2$; $lastnode \leftarrow null$

// Traverse the first linked list and make it circular

$length1 \leftarrow 0$

while $pointer1.next \neq null$ **do**

| $pointer1 \leftarrow pointer1.next$; $length1 \leftarrow length1 + 1$

$lastnode \leftarrow pointer1$; $pointer1.next \leftarrow head1$

// Set one of the pointers ahead

$pointer1 \leftarrow head2$

while $length1 > 0$ **do**

| $pointer1 \leftarrow pointer1.next$; $length1 \leftarrow length1 - 1$

// Traverse until they are equal, which is the intersection point

while $pointer1 \neq pointer2$ **do**

| $pointer1 \leftarrow pointer1.next$; $pointer2 \leftarrow pointer2.next$

// Remove the circular structure from the linked list

$lastnode.next \leftarrow null$

return $pointer1$

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(m+n\right), \Theta\left(1\right) \rangle$$

Solutions $\rightarrow$ System of linear equations

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:

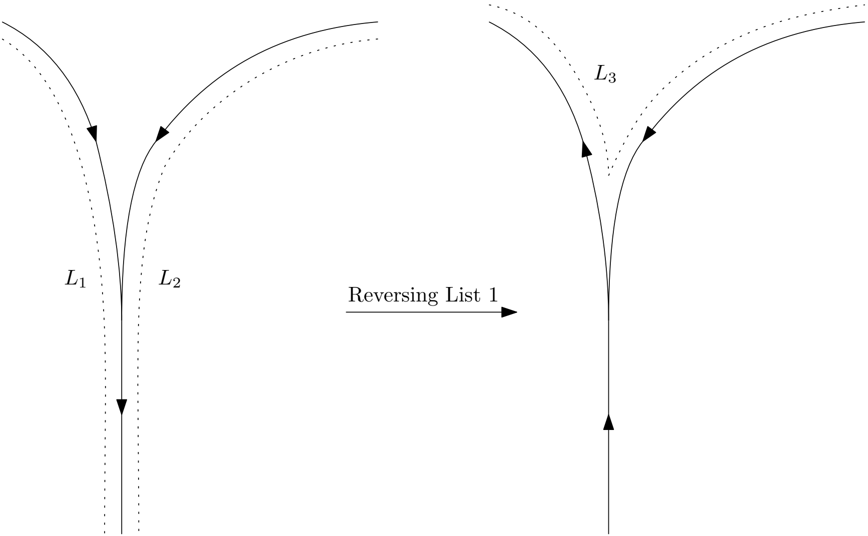
$$X + Z = L_1; \qquad Y + Z = L_2; \qquad X + Y = L3$$

5. We get:

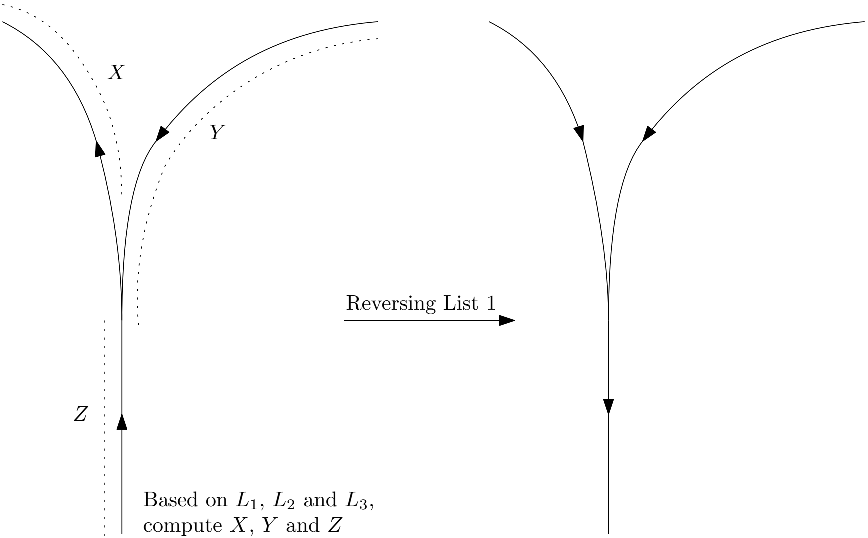
$$X = \frac{1}{2} \cdot (L_1 + L_3 - L_2); \qquad Y = \frac{1}{2} \cdot (L_2 + L_3 - L_1); \qquad Z = \frac{1}{2} \cdot (L_1 + L_2 - L_3)$$

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



Solutions → System of linear equations



Solutions → System of linear equations

$\textsc{YShapedLinkedList-LinearEquations}(head1, head2)$

// Compute the length of the second and first linked lists
$L_1 \leftarrow \textsc{GetLinkedListLength}(head1)$
$L_2 \leftarrow \textsc{GetLinkedListLength}(head2)$

// Reverse first linked list and compute L_3
$reversedhead1 \leftarrow \textsc{ReverseLinkedList}(head1)$
$L_3 \leftarrow \textsc{GetLinkedListLength}(head2)$

// 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
$answer \leftarrow head2$
for $i \leftarrow 1$ **to** Y **do** $answer \leftarrow answer.next$

// Restore first linked list by reversing it again
$head1 \leftarrow \textsc{ReverseLinkedList}(reversedhead1)$

return $answer$

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(m+n\right), \Theta\left(1\right) \rangle$$

Solutions → System of linear equations

$\textsc{GetLinkedListLength}(head)$

$L \leftarrow 0$; $pointer \leftarrow head$
while $pointer \neq null$ **do**
  $L \leftarrow L + 1$; $pointer \leftarrow pointer.next$
return L

$\textsc{ReverseLinkedList}(head)$

$current \leftarrow head$; $previous \leftarrow null$; $nextcurrent \leftarrow null$
// Iterate through the list and reverse pointers
while $current \neq null$ **do**
  $nextcurrent \leftarrow current.next$
  $current.next \leftarrow previous$
  $previous \leftarrow current$
  $current \leftarrow nextcurrent$
return $previous$

Solutions → Two pointers

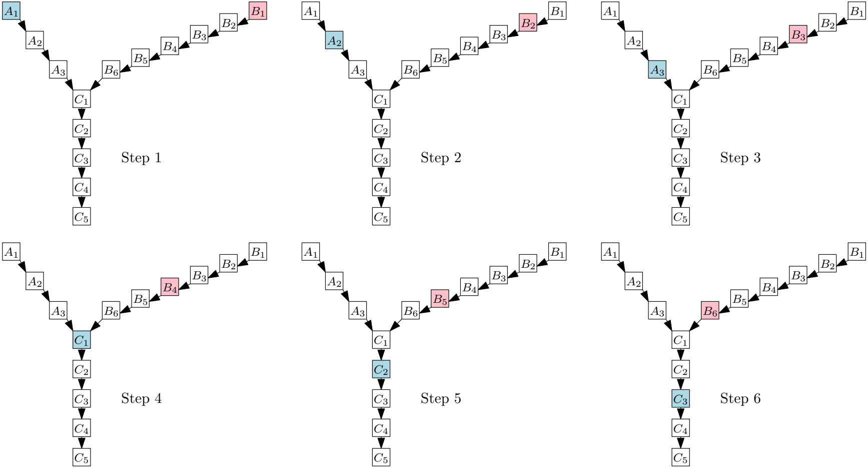
1. Scan list 1 using $pointer1$. Scan list 2 using $pointer2$.
2. If $pointer1$ reaches list 1 end, then start from list 2. If $pointer2$ reaches list 2 end, then start from list 1.
3. At any moment, when the two node references are same, it is the intersection node. Else, return $null$.

$\textsc{YShapedLinkedList-TwoPointers}(head1, head2)$

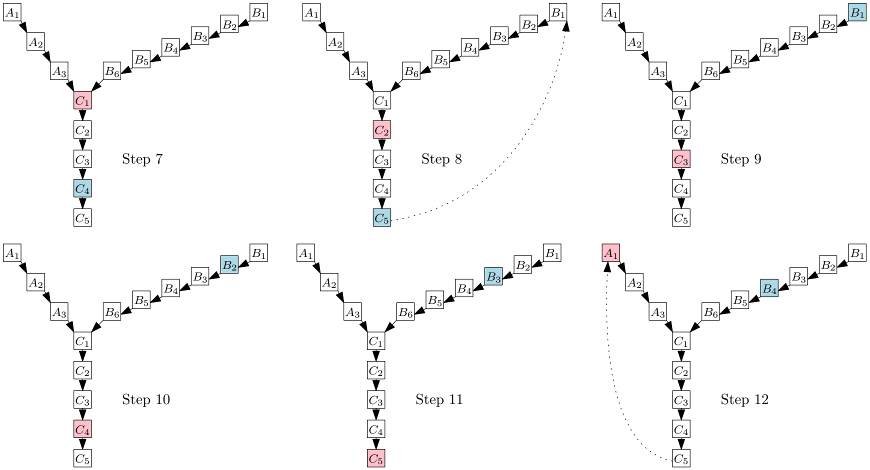
```
pointer1 ← head1; pointer2 ← head2
// If one of the lists is empty, then there is no intersection node
if pointer1 = null or pointer = null then return null
// Traverse the lists until the intersection node is found
while pointer1 ≠ pointer2 do
    pointer1 ← pointer1.next(); pointer2 ← pointer2.next()
    if pointer1 = pointer2 then return pointer1
    // If a pointer reaches its list end, then start from other list
    if pointer1 = null then pointer1 ← head2
    if pointer2 = null then pointer2 ← head1
return pointer1
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(m+n), \Theta(1) \rangle$$

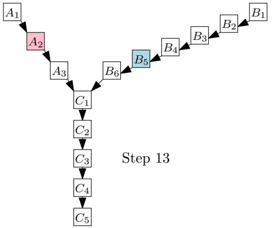
Solutions → Two pointers

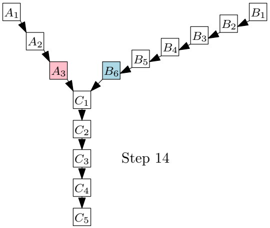


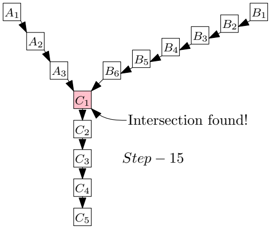
Solutions → Two pointers



Solutions → Two pointers







Complexity

Algorithm	Time	Extra Space
Brute force	$\mathcal{O}(mn)$	$\Theta(1)$
Two stacks	$\Theta(m+n)$	$\Theta(m+n)$
Hashset	$\mathcal{O}(m+n)$	$\Theta(m)$
Difference count	$\Theta(m+n)$	$\Theta(1)$
Loop in a linked list	$\Theta(m+n)$	$\Theta(1)$
Linear equations	$\Theta(m+n)$	$\Theta(1)$
Two pointers	$\mathcal{O}(m+n)$	$\Theta(1)$

Search Sorted Matrix HOME

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

```
SearchInSortedMatrix(A[1...m, 1...n], k)
for i ← 1 to m do
    for j ← 1 to n do
        if A[i, j] = k then
            return (i, j)
return −1
```

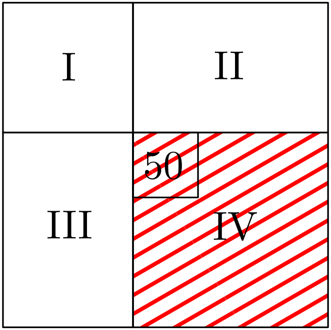
$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(mn), \Theta(1) \rangle$$

Solutions → D&C

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 $k = 30$

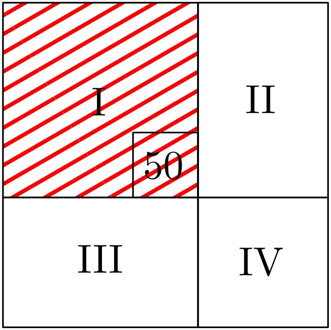


k can't be in quadrant IV

Search for k in quadrants I, II, and III

Solutions → D&C

- 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

Solutions $\rightarrow$ D&C

$\textsc{SearchInSortedMatrix}(A[1\ldots m, 1\ldots n], k)$

return $\text{D\&C}(A[1\ldots m, 1\ldots n], k)$

$\text{D\&C}(A[r_\ell \ldots r_h, c_\ell \ldots c_h], k)$

if $r_\ell > r_h$ **or** $c_\ell > c_h$ **then** **return** -1
$r_m \leftarrow (r_\ell + r_h)/2$; $c_m \leftarrow (c_\ell + c_h)/2$
if $A[r_m, c_m] = k$ **then** **return** (r_m, c_m)
// In this case, element is definitely not in the fourth quadrant
else if $A[r_m, c_m] > k$ **then**
  return
  $\text{D\&C}(A[r_\ell \ldots r_m - 1, c_\ell \ldots c_m - 1], k)$ **or** // first quadrant
  $\text{D\&C}(A[r_\ell \ldots r_{m-1}, c_m \ldots c_h], k)$ **or** // second quadrant
  $\text{D\&C}(A[r_m \ldots r_h, c_\ell \ldots c_m - 1], k)$ **or** // third quadrant
// In this case, element is definitely not in the first quadrant
else if $A[r_m, c_m] < k$ **then**
  return
  $\text{D\&C}(A[r_\ell \ldots r_m, c_m + 1 \ldots c_h], k)$ **or** // second quadrant
  $\text{D\&C}(A[r_{m+1} \ldots r_h, c_\ell \ldots c_m], k)$ **or** // third quadrant
  $\text{D\&C}(A[r_m + 1 \ldots r_h, c_m + 1 \ldots c_h], k)$ **or** // fourth quadrant

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(\min(m,n)^{\log_2 3}\right), \mathcal{O}\left(\log \max(m,n)\right) \right\rangle$$

Solutions $\rightarrow$ D&C improved

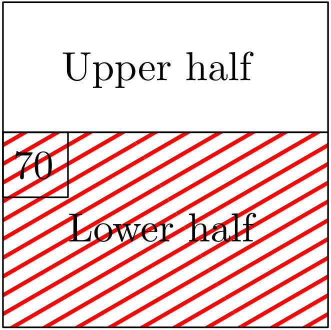
Core idea

- 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

Solutions → D&C improved

- Case 1: there is no index
  i.e., suppose the first element in the mid row is 70,
  which is greater than $k = 50$



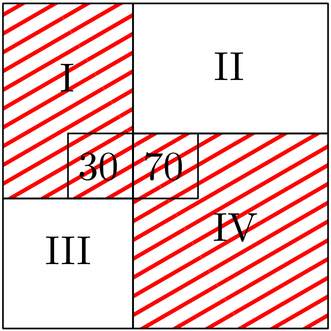
k can't be in the lower half

Search for k in the upper half

Area of lower half is $> 50\%$

Solutions → D&C improved

- 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 $> 50\%$

Solutions → D&C improved

$k = 10$

Binary search row

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

k not found,
9 is max element < k,
16 is min element > k

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

Eliminating red zones,
Recursively searching white top-right and bottom-left sub-matrices

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

Solutions → D&C improved

$\textsc{SearchInSortedMatrix}(A[1 \ldots m, 1 \ldots n], k)$

return $\textsc{D\&C-Improved}(A[1 \ldots m, 1 \ldots n], k)$

$\textsc{D\&C-Improved}(A[r_\ell \ldots r_h, c_\ell \ldots c_h], k)$

if $c_\ell > c_h$ **or** $r_\ell > r_h$ **then** **return** -1
// Binary search returns the largest index j in $[c_\ell \ldots c_h]$ for which $A[r_m, j] \leq k$. If no such index exists, it returns -1
$r_m \leftarrow (r_\ell + r_h)/2$
$j \leftarrow \textsc{BinarySearch}(A[r_m, c_\ell \ldots c_h])$
if $j = -1$ **then**
  return $\textsc{D\&C-Improved}(A[r_\ell \ldots r_m - 1, c_\ell \ldots c_h], k)$ // upper half
else if $A[r_m, j] = k$ **then**
  return (r_m, j)
else if $A[r_m, j] < k$ **then**
  return
  $\textsc{D\&C-Improved}(A[r_\ell \ldots r_m - 1, j+1 \ldots c_h], k)$ **or** // region II
  $\textsc{D\&C-Improved}(A[r_m + 1 \ldots r_h, c_\ell \ldots j], k)$ // region III

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(m \log n), \mathcal{O}(\log m) \rangle$$

Solutions → Binary search

$k = 8$

\# columns < \# rows, so binary search on each column

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

	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

		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

			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

Element found!

Solutions $\rightarrow$ Binary search

- Perform a binary search for k in each row or column

```
SearchInSortedMatrix(A[1...m, 1...n], k)
  // #rows < #columns
  if m < n then
  |  for i ← 1 to m do
  |  |  j ← BinarySearch(A[i, 1...n], k)
  |  |  if j ≠ −1 then  return (i, j)
  // #columns ≤ #rows
  else
  |  for j ← 1 to n do
  |  |  j ← BinarySearch(A[1...m, j], k)
  |  |  if j ≠ −1 then  return (i, j)
  return −1
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}\left(\min(m,n)\log\max(m,n)\right), \Theta\left(1\right) \rangle$$

Solutions $\rightarrow$ Decrease-and-conquer

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

```
SearchInSortedMatrix(A[1...m, 1...n], k)
  // Start from the top right element
  row ← 1; col ← n
  while row ≤ m and col ≥ 1 do
    if A[row, col] = k then
      return (row, col)
    // In this case, go left as column col (down elements) can't have k
    else if k < A[row, col] then
      col ← col − 1
    // In this case, go down as row row (left elements) can't have k
    else if k > A[row, col] then
      row ← row + 1
  return −1
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(m+n), \Theta(1) \rangle$$

Solutions $\rightarrow$ Decrease-and-conquer

$k = 10$

Starting from top right

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

Element found!

Solutions → Decrease-and-conquer

- 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?

Complexity

Algorithm	Time	Space
Linear search	$\mathcal{O}(mn)$	$\Theta(1)$
D&C	$\mathcal{O}\left(\min(m,n)^{\log 3}\right)$	$\mathcal{O}(\log \max(m,n))$
Improved D&C	$\mathcal{O}(m \log n)$	$\mathcal{O}(\log m)$
Binary search	$\mathcal{O}(\min(m,n) \log \max(m,n))$	$\Theta(1)$
Decrease-and-conquer	$\mathcal{O}(m+n)$	$\Theta(1)$

First Missing Positive

Problem

- Given an array $A[1 \dots n]$ 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 $A[1 \dots n]$ for $i \in [1 \dots n]$
2. Stop and return the smallest such i, otherwise return $n + 1$

```
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
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n^2\right), \Theta\left(1\right) \right\rangle$$

Solutions → Brute force 2

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 $n+1$

```
FirstMissingPositive-BruteForce2(A[1...n])

// Create a sorted set to store the natural numbers
Create an empty sorted set S using a balanced search tree
for i ← 1 to n do
    if A[i] > 0 then
        S.Add(A[i])
// Find the first missing natural number from A[1...n] using S
for i ← 1 to n do
    if S does not contain i then
        return i
return n + 1
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n \log n), \mathcal{O}(n) \rangle$$

Solutions → Scan

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 $n + 1$

```
FirstMissingPositive-Scan(A[1...n])

  // Sort the array in-place
  Sort(A[1...n])
  // Skip negative numbers and zero from the array
  index ← 1
  while A[index] < 1 do  index ← index + 1
  i ← 1
  // Find the missing natural number from the sorted input array
  for j ← index to n do
  |  if A[j] = i then  i ← i + 1
  |  else if A[j] > i then
  |  return i
  return i
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n \log n), \Theta(1) \rangle$$

Solutions $\rightarrow$ In-place hashing

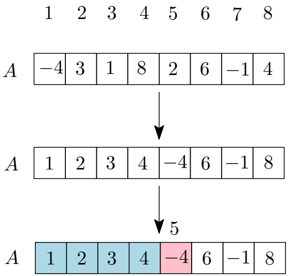
1. Use $i \in [1 \ldots n]$ of the same array to mark the presence of the numbers
2. If $A[i]$ is a natural number and $i \leq n$, swap & place it in $A[A[i]]$
3. Stop and return smallest i where $A[i] \neq i$, otherwise return $n+1$

FirstMissingPositive-InPlaceHashing$(A[1 \ldots n])$

```
// Swap natural number A[i] to A[i]th index if A[i] ∈ [1...n]
for i ← 1 to n do
    while A[i] ≥ 1 and A[i] ≤ n and A[i] ≠ A[A[i]] do
        Swap(A[i],A[A[i]])
// Find the first natural number that is not A[i] ≠ i in A[1...n]
for i ← 1 to n do
    if A[i] ≠ i then
        return i
return n + 1
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Solutions → In-place hashing



$A[1 \dots n]$

For each $i \in [1, n]$, do this in a loop.

If $A[i] \in [1, n]$ and $A[i] \neq A[A[i]]$, then Swap($A[i]$,$A[A[i]]$)

Return the first i which is not equal to $A[i]$

That is, return 5

Solutions $\rightarrow$ In-place hashing

	1	2	3	4	5	6	7	8	
A	-4	3	1	8	2	6	-1	4	$i = 1$; $A[i] \notin [1, n]$, so do nothing
A	-4	3	1	8	2	6	-1	4	$i = 2$; $A[i] \in [1, n]$ and $A[i] \neq A[A[i]]$, so $\textsc{Swap}(A[i], A[A[i]])$
A	-4	1	3	8	2	6	-1	4	$i = 2$; $A[i] \in [1, n]$ and $A[i] \neq A[A[i]]$, so $\textsc{Swap}(A[i], A[A[i]])$
A	1	-4	3	8	2	6	-1	4	$i = 2$; $A[i] \notin [1, n]$, so do nothing
A	1	-4	3	8	2	6	-1	4	$i = 3$; $A[i] \in [1, n]$ but $A[i] = A[A[i]]$, so do nothing
A	1	-4	3	8	2	6	-1	4	$i = 4$; $A[i] \in [1, n]$ and $A[i] \neq A[A[i]]$, so $\textsc{Swap}(A[i], A[A[i]])$
A	1	-4	3	4	2	6	-1	8	$i = 4$; $A[i] \in [1, n]$ but $A[i] = A[A[i]]$, so do nothing
A	1	-4	3	4	2	6	-1	8	$i = 5$; $A[i] \in [1, n]$ and $A[i] \neq A[A[i]]$, so $\textsc{Swap}(A[i], A[A[i]])$
A	1	2	3	4	-4	6	-1	8	$i = 5$; $A[i] \notin [1, n]$, so do nothing
A	1	2	3	4	-4	6	-1	8	$i = 6$; $A[i] \in [1, n]$ but $A[i] = A[A[i]]$, so do nothing
A	1	2	3	4	-4	6	-1	8	$i = 7$; $A[i] \notin [1, n]$, so do nothing
A	1	2	3	4	-4	6	-1	8	$i = 8$; $A[i] \in [1, n]$ but $A[i] = A[A[i]]$, so do nothing

Solutions $\rightarrow$ Hash table

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($A[1\ldots n]$)

```
// 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
```

$$\langle \text{Time, Space} \rangle = \langle \mathcal{O}(n), \mathcal{O}(n) \rangle$$

This Solution Might Not Always Work. Why?

Complexity

Algorithm	Time	Space
Brute Force 1	$\mathcal{O}\left(n^2\right)$	$\Theta\left(1\right)$
Brute Force 2	$\mathcal{O}\left(n \log n\right)$	$\mathcal{O}\left(n\right)$
Scan	$\mathcal{O}\left(n \log n\right)$	$\Theta\left(1\right)$
In-Place Hashing	$\Theta\left(n\right)$	$\Theta\left(1\right)$
In-Place Hashing & Partition	$\Theta\left(n\right)$	$\Theta\left(1\right)$

Celebrity Problem HOME

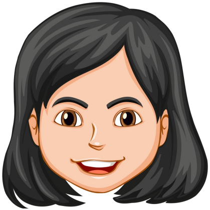
Problem

- The knowledge of n guests in a party is represented by a binary matrix $M[1 \dots n, 1 \dots n]$, 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:**

$i : j$	1	2	3
1	0	1	1
2	0	0	0
3	1	1	0

Output: 2

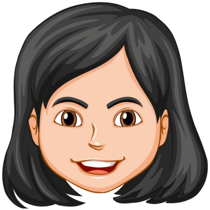
Core idea (take-home lesson)



i knows j, i.e., $M[i, j] = 1$

i is not a celebrity

j is a potential celebrity



i doesn't know j, i.e., $M[i, j] = 0$

j is not a celebrity

i 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
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n^2\right), \Theta\left(1\right) \right\rangle$$

Solutions $\rightarrow$ Graph

- $Indegree$ 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).

Solutions → Graph

```
FindCelebrity(M[1...n, 1...n])
  // Step 1. Calculate outdegree and indegree of each node
  outdegree[1...n] ← [0...0]; indegree[1...n] ← [0...0]
  for i ← 1 to n do
  |  for j ← 1 to n do
  |  |  // As i knows j, increment outdegree of i and indegree of j
  |  |  if M[i, j] = 1 then
  |  |  |  outdegree[i] ← outdegree[i] + 1
  |  |  |  indegree[j] ← indegree[j] + 1
  // Step 2. Finding the celebrity
  for i ← 1 to n do
  |  if outdegree[i] = 0 and indegree[i] = n − 1 then
  |  |  return i                                    // celebrity found
  return −1
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^2\right), \Theta\left(n\right) \right\rangle$$

Solutions → Graph

	1	2	3	*outdegree*	*indegree*
1	0	1	1	0	0
2	0	0	0	0	0
3	1	1	0	0	0

	1	2	3	*outdegree*	*indegree*
1	0	1	1	0	0
2	0	0	0	0	0
3	1	1	0	0	0

	1	2	3	*outdegree*	*indegree*
1	0	1	1	1	0
2	0	0	0	0	1
3	0	1	0	0	0

	1	2	3	*outdegree*	*indegree*
1	0	1	1	2	0
2	0	0	0	0	1
3	0	1	0	0	1

	1	2	3	*outdegree*	*indegree*
1	0	1	1	2	0
2	0	0	0	0	1
3	1	1	0	0	1

	1	2	3	*outdegree*	*indegree*
1	0	1	1	2	0
2	0	0	0	0	1
3	1	1	0	0	1

	1	2	3	*outdegree*	*indegree*
1	0	1	1	2	0
2	0	0	0	0	1
3	1	1	0	0	1

	1	2	3	*outdegree*	*indegree*
1	0	1	1	2	1
2	0	0	0	0	1
3	1	1	0	1	1

	1	2	3	*outdegree*	*indegree*
1	0	1	1	2	1
2	0	0	0	0	2
3	1	1	0	2	1

	1	2	3	*outdegree*	*indegree*
1	0	1	1	2	1
2	0	0	0	0	2
3	1	1	0	2	1

	outdegree	*indegree*
1	2	1
2	0	2
3	2	1

Celebrity

Solutions $\rightarrow$ Recursion

- 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.

Solutions → Recursion

```
FINDCELEBRITY(M[1...n, 1...n])
  // Step 1. Find the celebrity candidate
  candidate ← FINDPOTENTIALCELEBRITY(M, n)
  if candidate = −1 then
  |  return −1                                   // no celebrity found
  // Step 2. Check if the candidate is a celebrity
  outdegree ← 0; indegree ← 0
  for i ← 1 to n do
  |  if i ≠ candidate then
  |  |  outdegree ← outdegree + M[candidate, i]
  |  |  indegree ← indegree + M[i, candidate]
  if outdegree = 0 and indegree = n − 1 then
  |  return candidate
  return −1                                      // no celebrity found
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$$

Solutions → Recursion

$\textsc{FindPotentialCelebrity}(M[1\ldots n, 1\ldots n], m)$

```
if m = 0 then return −1
// Recursively find celebrity in the first m − 1 persons
candidate ← FindPotentialCelebrity(M, m − 1)
// If there is no candidate in the first m − 1 people, m is the new candidate
if candidate = −1 then return m
// If candidate knows m person, then m is the new candidate
if M[candidate, m] = 1 then return m
// If m knows candidate, then candidate is the new candidate
if M[m, candidate] = 1 then return candidate
return −1
```

Solutions $\rightarrow$ Recursion

M

0	1	1	0
0	0	0	0
0	1	0	0
1	1	0	0

$N = 4$

FindPotentialCelebrity$(M, 4)$

FindPotentialCelebrity$(M, 3)$

FindPotentialCelebrity$(M, 2)$

FindPotentialCelebrity$(M, 1)$

FindPotentialCelebrity$(M, 0)$
Base case
Exiting with -1

$candidate = -1$, so exiting with $m = 1$

$candidate \neq -1$
$M[candidate, m] = M[1, 2] = 1$, so exiting with $m = 2$

$candidate \neq -1$
$M[candidate, m] = M[2, 3] \neq 1$, so checking ahead
$M[m, candidate] = M[3, 2] = 1$, so exiting with $m = 2$

$candidate \neq -1$
$M[candidate, m] = M[2, 4] \neq 1$, so checking ahead
$M[m, candidate] = M[4, 2] = 1$, so exiting with $m = 2$
Potential celebrity is 2

Solutions $\rightarrow$ Elimination technique

- 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.

Solutions → Elimination technique

```
FindCelebrity(M[1...n, 1...n])
  // Step 1. Find a potential celebrity
  Create a stack S ← [ ] to store all potential celebrities
  for i ← 1 to n do  S.Push(i)
  while Stack S has greater than 1 element do
  |  i ← S.Pop(); j ← 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 ← S.Pop()

  // Step 2. Check if the candidate is a celebrity
  for i ← 1 to n do
  |  if i ≠ candidate then
  |  |  if M[i, candidate] = 0 then  return −1
  |  |  if M[candidate, i] = 1 then  return −1
  return candidate
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(n) \rangle$$

Solutions → Elimination technique

M	1	2	3	4	Stack	Popped Elements
1	0	1	1	0		
2	0	0	0	0		
3	0	1	0	0		
4	1	1	0	0		

M	1	2	3	4	Stack	Popped Elements
1	0	1	1	0	4	
2	0	0	0	0	3	$i = 4$
3	0	1	0	0	2	$j = 3$
4	1	1	0	0	1	

M	1	2	3	4	Stack	Popped Elements
1	0	1	1	0		
2	0	0	0	0	4	$i = 4$
3	0	1	0	0	2	$j = 2$
4	1	1	0	0	1	

M	1	2	3	4	Stack	Popped Elements
1	0	1	1	0		
2	0	0	0	0		$i = 2$
3	0	1	0	0	2	$j = 1$
4	1	1	0	0	1	

M	1	2	3	4	Stack	Popped Elements
1	0	1	1	0		
2	0	0	0	0		
3	0	1	0	0		
4	1	1	0	0	2	Potential celebrity is 2

Solutions $\rightarrow$ Efficient elimination technique

- 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.

Solutions → Efficient elimination technique

```
FindCelebrity(M[1...n, 1...n])
  // Step 1. Find the celebrity candidate
  r ← 1
  for i ← 2 to n do
  |  if M[r, i] = 1 then
  |  |  M[r, r] ← ⋆                          // r can't be a celebrity
  |  |  r ← i                                     // update r to i
  |  else
  |  |  M[i, i] ← ⋆                           // i can't be a celebrity
  // The single candidate will have its diagonal cell as 0
  candidate ← 1
  while candidate ≤ n do
  |  if M[candidate, candidate] = 0 then break
  // Step 2. Check if the candidate is really the celebrity
  for i ← 1 to n do
  |  if i ≠ 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

Solutions → Efficient elimination technique

M	Pointers	Action
$\begin{array}{c\|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 \\ 4 & 1 & 1 & 0 & 0 \end{array}$	$r = 1$ $i = 2$	Since $M[r, i] = M[1, 2] = 1$, Set: $M[r, r] = M[1, 1] \leftarrow \star$ and $r \leftarrow i = 2$
$\begin{array}{c\|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \star & 1 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 \\ 4 & 1 & 1 & 0 & 0 \end{array}$	$r = 2$ $i = 3$	Since $M[r, i]$ is 0, Set: $M[i, i] \leftarrow 1$
$\begin{array}{c\|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \star & 1 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & \star & 0 \\ 4 & 1 & 1 & 0 & 0 \end{array}$	$r = 2$ $i = 4$	Since $M[r, i]$ is 0, Set: $M[i, i] \leftarrow 1$
$\begin{array}{c\|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \star & 1 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & \star & 0 \\ 4 & 1 & 1 & 0 & \star \end{array}$		Check Diagonals Potential celebrity is 2

Solutions $\rightarrow$ Two pointers

- 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.

Solutions → Two pointers

```
FindCelebrity-TwoPointers(M[1...n, 1...n])
// Step 1. Find the celebrity candidate
i ← 1, j ← n
while i < j do
    if M[j, i] = 1 then
        j ← j − 1      // person j knows person i, so j can't be a celebrity
    else
        i ← i + 1      // person j doesn't know i, so i can't be a celebrity
candidate ← i          // person i is the celebrity candidate
// Step 2. Check if the candidate is really the celebrity
for j ← 1 to n do
    if j ≠ candidate then
        if M[j, candidate] = 0 or M[candidate, j] = 1 then
            return −1      // candidate is not a celebrity
return candidate       // candidate is the celebrity
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n), \Theta(1) \rangle$$

Solutions → Two pointers (example 1)

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	1	1	0	1	1

People	Pointers	Action
1 2 3 4 5	$i = 1$, $j = 5$	Since $M[j, i] = M[5, 1] = 1$, j cannot be a celebrity. Hence, $j \leftarrow j - 1$.
1 2 3 4 5	$i = 1$, $j = 4$	Since $M[j, i] = M[4, 1] = 1$, j cannot be a celebrity. Hence, $j \leftarrow j - 1$.
1 2 3 4 5	$i = 1$, $j = 3$	Since $M[j, i] = M[3, 1] = 0$, i cannot be a celebrity. Hence, $i \leftarrow i + 1$.
1 2 3 4 5	$i = 2$, $j = 3$	Since $M[j, i] = M[3, 2 = 1$, j cannot be a celebrity. Hence, $j \leftarrow j - 1$.
1 2 3 4 5	$i = 2$, $j = 2$	Since $i = j = 2$ i.e. $i \geq j$, Potential celebrity is 2

$i = 2$ turns out to be a celebrity, return 2

Solutions → Two pointers (example 2)

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	Pointers	Action
1 (i) 2 3 4 5 (j)	$i = 1$ $j = 5$	Since $M[j, i] = M[5, 1] = 0$, i cannot be a celebrity. Hence, $i \leftarrow i + 1$.
1 2 (i) 3 4 5 (j)	$i = 2$ $j = 5$	Since $M[j, i] = M[5, 2] = 0$, i cannot be a celebrity. Hence, $i \leftarrow i + 1$.
1 2 3 (i) 4 5 (j)	$i = 3$ $j = 5$	Since $M[j, i] = M[5, 3] = 0$, i cannot be a celebrity. Hence, $i \leftarrow i + 1$.
1 2 3 4 (i) 5 (j)	$i = 4$ $j = 5$	Since $M[j, i] = M[5, 4] = 1$, j cannot be a celebrity. Hence, $j \leftarrow j - 1$.
1 2 3 4 (i) 5	$i = 4$ $j = 4$	Since $i = j = 4$ i.e. $i \geq j$, Potential celebrity is 4

$i = 4$ turns out to not be a celebrity,
return -1

Complexity

Algorithm	Time	Space
Brute force	$\mathcal{O}(n^2)$	$\Theta(1)$
Graph	$\Theta(n^2)$	$\Theta(n)$
Recursion	$\Theta(n)$	$\Theta(n)$
Elimination	$\Theta(n)$	$\Theta(n)$
Efficient elimination	$\Theta(n)$	$\Theta(1)$
Two pointers	$\Theta(n)$	$\Theta(1)$

Random Permutation

Problem

- 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 $1/n!$.

Solutions $\rightarrow$ Algorithm 1

$\textsc{RandomPermutation}(A[1\ldots n])$

for $i \leftarrow 1$ **to** $n-1$ **do**
  $x \leftarrow \textsc{Random}([1\ldots n])$
  $y \leftarrow \textsc{Random}([1\ldots n])$
  $\textsc{Swap}(A[x], A[y])$

Solutions → Algorithm 1

$\textsc{RandomPermutation}(A[1 \ldots n])$

```
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 $= \left(n^2\right)^k$
- 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 ($\left(3^2\right)^k$)

Solutions $\rightarrow$ Algorithm 2

$\textsc{RandomPermutation}(A[1 \ldots n])$

for $i \leftarrow 1$ **to** $n - 1$ **do**
  $x \leftarrow i$
  $y \leftarrow \textsc{Random}([1 \ldots n])$
  $\textsc{Swap}(A[x], A[y])$

Solutions → Algorithm 2

$\textsc{RandomPermutation}(A[1 \ldots n])$

for $i \leftarrow 1$ **to** $n - 1$ **do**
    $x \leftarrow i$
    $y \leftarrow \textsc{Random}([1 \ldots n])$
    $\textsc{Swap}(A[x], A[y])$

The algorithm is incorrect.

Counterexample:

- Suppose $k = \#$iterations for which actual swap takes place
  total $\#$permutations $= n!$
  total $\#$outcomes $= n^k$
- 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 (n^k)

Solutions $\rightarrow$ Algorithm 3

$\textsc{RandomPermutation}(A[1\ldots n])$

for $i \leftarrow 1$ **to** $n-1$ **do**
    $x \leftarrow i$
    $y \leftarrow \textsc{Random}([i+1\ldots n])$
    $\textsc{Swap}(A[x], A[y])$

Solutions → Algorithm 3

$\textsc{RandomPermutation}(A[1 \ldots n])$

for $i \leftarrow 1$ **to** $n-1$ **do**
  $x \leftarrow i$
  $y \leftarrow \textsc{Random}([i+1 \ldots n])$
  $\textsc{Swap}(A[x], A[y])$

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 ($(n-1)!$)

Solutions → Algorithm 4

$\textsc{RandomPermutation}(A[1 \ldots n])$

```
Choose a random hash function for the hash table H
for i ← 1 to n do
|   H.Add(A[i])                    // add the element even if the key exists
for i ← 1 to n do
|   A[i] ← ith element in hash table H
```

Solutions $\rightarrow$ Fisher-Yates

$\textsc{RandomPermutation}(A[1 \dots n])$

for $i \leftarrow n$ **downto** 2 **do**
  $x \leftarrow \textsc{Random}([1 \dots i])$
  for $j \leftarrow x + 1$ **downto** i **do**
    $A[j-1] \leftarrow A[j]$
  $A[i] \leftarrow A[x]$

The algorithm is correct.

Proof:

- We want to prove that probability of producing random permutation is $1/n!$

$$\text{Probability} = \frac{1}{n} \times \frac{1}{n-1} \times \cdots \times \frac{1}{2} = \frac{1}{n!}$$

Solutions → Durstenfeld

$\textsc{RandomPermutation}(A[1 \ldots n])$

for $i \leftarrow 1$ **to** $n - 1$ **do**
  $x \leftarrow i$
  $y \leftarrow \textsc{Random}([i \ldots n])$
  $\textsc{Swap}(A[x], A[y])$

can also be written as

$\textsc{RandomPermutation}(A[1 \ldots n])$

for $i \leftarrow n$ **downto** 2 **do**
  $x \leftarrow i$
  $y \leftarrow \textsc{Random}([1 \ldots i])$
  $\textsc{Swap}(A[x], A[y])$

The algorithm is correct.

Solutions → Durstenfeld

```
RandomPermutation(A[1...n])

for i ← n downto 2 do
    x ← i
    y ← Random([1...i])
    Swap(A[x], A[y])
```

Proof:

- Let $X = [1, 2, \ldots, n]$ and $Y = [p_1, p_2, \ldots, p_n]$
- We want to prove that probability of producing Y from X is $1/n!$

$$
\begin{aligned}
P(Y) &= P(Y[1] = p_1, Y[2] = p_2, \ldots, Y[n] = p_n) \\
&= P(Y[n] = p_n) \\
&\qquad \times P(Y[n-1] = p_{n-1} \mid Y[n] = p_n) \\
&\qquad \times P(Y[n-2] = p_{n-2} \mid Y[n] = p_n, Y[n-1] = p_{n-1}) \\
&\qquad \times \cdots \\
&\qquad \times P(Y[1] = p_1 \mid Y[n] = p_n, Y[n-1] = p_{n-1}, \ldots, Y[2] = p_2) \\
&= \frac{1}{n} \times \frac{1}{n-1} \times \cdots \times \frac{1}{1} = \frac{1}{n!}
\end{aligned}
$$

Count Distinct Pairs

HOME

Problem

- Given an array of unique integers $A[1 \dots n]$ 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: $[1, 1, 1, 1, 2, 2, 2, 2]$, $k = 1$
  Input has duplicates, so this type of input is not allowed

Solutions → Brute force

- Consider every pair of elements and increment count if the difference equals k

```
CountPairs-BruteForce(A[1...n], k)
count ← 0
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        if absolute(A[i] − A[j]) = k then
            count ← count + 1
return count
```

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^2\right), \Theta\left(1\right) \right\rangle$$

Solutions → Sorting + binary search

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...n], k)

Sort(A[1...n])
count ← 0
for i ← 1 to n − 1 do
    // Check if A[i] + k in A[i + 1 ... n] using binary search
    if BinarySearch(A[i + 1 ... n], A[i] + k) ≠ −1 then
        count ← count + 1
return count
```

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n \log n\right), \Theta\left(1\right) \rangle$$

Solutions $\rightarrow$ Hashing

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 $count/2$

```
CountingPairs-Hashing(A[1...n],k)
─────────────────────────────────
Create an empty HashTable H
for i ← 1 to n do  H.Add(A[i])

count ← 0
for i ← 1 to n do
|   if H.Contains(A[i] + k) then  count ← count + 1
|   if H.Contains(A[i] − k) then  count ← count + 1

return count/2
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n)^*, \mathcal{O}(n) \rangle$$

Solutions $\rightarrow$ Sorting + two pointers

1. Sort $A[1 \dots n]$, initialize two pointers low and $high$ to 1
2. Calculate the difference $diff = A[high] - A[low]$.
   While $high \leq n$
   - If $diff = k$, increment low, $high$, and $count$
   - If $diff > k$, increment low
   - If $diff < k$, increment $high$

$\textsc{CountingPairs-TwoPointers}(A[1 \dots n], k)$

$\textsc{Sort}(A[1 \dots n])$

$count \leftarrow 0$; $low \leftarrow 1$; $high \leftarrow 1$
while $high \leq n$ **do**
  $diff = A[high] - A[low]$
  if $diff = k$ **then**
    $count \leftarrow count + 1$; $low \leftarrow low + 1$; $high \leftarrow high + 1$
  else if $diff > k$ **then** $low \leftarrow low + 1$
  else if $diff > k$ **then** $high \leftarrow high + 1$
return $count$

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n \log n\right), \Theta\left(1\right) \rangle$$

Solutions → Sorting + two pointers

$k = 3$

	1	2	3	4	5	6	
A	1	2	3	5	6	7	This logic applies only for sorted arrays
A	1	2	3	5	6	7	$count = 0$; $A[h] - A[\ell] < k$, so increment h
	ℓ h						
A	1	2	3	5	6	7	$count = 0$; $A[h] - A[\ell] < k$, so increment h
	ℓ	h					
A	1	2	3	5	6	7	$count = 0$; $A[h] - A[\ell] < k$, so increment h
	ℓ		h				
A	1	2	3	5	6	7	$count = 0$; $A[h] - A[\ell] > k$, so increment ℓ
	ℓ			h			
A	1	2	3	5	6	7	$count = 0$; $A[h] - A[\ell] = k$, so increment ℓ, h, $count$
		ℓ		h			
A	1	2	3	5	6	7	$count = 1$; $A[h] - A[\ell] = k$, so increment ℓ, h, $count$
			ℓ		h		
A	1	2	3	5	6	7	$count = 2$; $A[h] - A[\ell] < k$, so increment h
				ℓ		h	

Counting pairs summary

Algorithm	Time	Space
Brute force	$\Theta\left(n^2\right)$	$\Theta\left(1\right)$
Sorting + Binary Search	$\Theta\left(n \log n\right)$	$\Theta\left(1\right)$
Hashing	$\Theta\left(n\right)^*$	$\mathcal{O}\left(n\right)$
Sorting + Two Pointers	$\Theta\left(n \log n\right)$	$\Theta\left(1\right)$

Solve the problem when there are duplicates in the array.

Maximum and Minimum

HOME

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: $[20, -2]$

Solutions $\rightarrow$ Brute force

- Traverse the array and compare each element with max and min.

```
BruteForce(A[1...n])

max ← A[1]; min ← A[1]
for i ← 2 to n do
    if A[i] > max then  max ← A[i]
    else if A[i] < min then  min ← A[i]
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.

```
IncrementByTwo(A[1 ... n])

if n is odd then { max ← A[1], min ← A[1], i ← 2 }
else
    if A[1] < A[2] then { max ← A[2]; min ← A[1] }
    else { max ← A[1]; min ← A[2] }
    i ← 3
while i < n do
    if A[i] < A[i + 1] then
        if A[i] < min then min ← A[i]
        if A[i + 1] > max then max ← A[i + 1]
    else
        if A[i] > max then max ← A[i]
        if A[i + 1] < min then min ← A[i + 1]
    i ← i + 2
return (max, min)
```

$$\langle \text{Time, Space} \rangle = \left\langle \frac{3}{2}\left(n - 1 - \boxed{n \text{ is even}}\right), \Theta(1) \right\rangle$$

Solutions $\rightarrow$ 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.

Solutions → Divide-and-conquer

```
DivideAndConquer(A[low ... high])

size ← high − low + 1
if size = 1 then { max ← A[low]; min ← A[low] }
else if size = 2 then
    if A[low] < A[high] then { max ← A[high]; min ← A[low] }
    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)
```

$$T(n) = \left\{ \begin{array}{ll} n-1 & \text{if } n = 1 \text{ or } 2, \\ 2T(n/2)+2 & \text{if } n > 2. \end{array} \right\}$$

$$\langle \text{Time, Space} \rangle = \left\langle \frac{3n}{2} - 2, \Theta\left(\log n\right) \right\rangle$$

Complexity

Algorithm	Time	Space
Brute force	$2n - 2$	$\Theta(1)$
Increment by two	$\frac{3}{2}\left(n - 1 - \boxed{n \text{ is even}}\right)$	$\Theta(1)$
Divide-and-conquer	$\frac{3n}{2} - 2$	$\Theta(\log n)$

Sorting Algorithms HOME

Problem

- Design an efficient algorithm to sort a given array $A[1 \dots n]$.
- Input: $[80, 30, 90, 50, 40, 20, 100]$
  Output: $[20, 30, 40, 50, 80, 90, 100]$
- Input: $[23, 15, 40, 15, 10]$
  Output: $[10, 15, 15, 23, 40]$

Solutions → Permutation sort

$\textsc{PermutationSort}(A[1 \ldots n])$

```
while true do
    RandomPermute(A[1...n])
    if IsSorted(A[1...n]) then
        break
```

$\textsc{RandomPermute}(A[1 \ldots n])$

```
for i ← 1 to n − 1 do
    A[i] ← Swap(A[i], A[Random(i...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.

```
SlowSort(A[low ... high])

if i ≥ j then  return

  // Sort the two halves recursively
mid ← (low + high)/2
SlowSort(A[low ... mid])
SlowSort(A[mid + 1 ... high])

  // The largest element of the subarray should go to its correct position
if A[high] < A[mid] then
|  Swap(A[high], A[mid])

  // Sort the remaining subarray
SlowSort(A[low ... high − 1])
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n^{\frac{\log_2 n}{2}}\right), \Theta\left(n\right) \right\rangle$$

Solutions → Pancake sort

1. Suppose the index of the maximum element in $A[1 \ldots n]$ is $maxindex$.
2. Reverse $A[1 \ldots maxindex]$ to move the largest element in the array to index 1.
3. Reverse $A[1 \ldots n]$ to move the largest element to $A[n]$.
4. Recursively sort $A[1 \ldots n-1]$.

```
PancakeSort(A[1...n])
    // The ith iteration finds the ith largest element
    for i ← n downto 2 do
        // Step 1. Find the index of the max(A[1...i])
        maxindex ← 1
        for j ← 2 to i do
            if A[j] > A[maxindex] then
                maxindex ← j
        // Step 2. Move max(A[1...maxindex]) to index 1
        Reverse(A[1...maxindex])
        // Step 3. Move A[1] to its correct position
        Reverse(A[1...i])
```

$$\langle \text{Time, Space} \rangle = \left\langle \mathcal{O}\left(n^2\right), \Theta\left(1\right) \right\rangle$$

Solutions → Pancake sort

7	9	7	8	6
9	7	7	8	6
6	8	7	7	9

$i = 5$; $maxindex = 2$

Reverse $A[1 \ldots maxindex]$

Reverse $A[1 \ldots i]$

6	8	7	7	9
8	6	7	7	9
7	7	6	8	9

$i = 4$; $maxindex = 2$

Reverse $A[1 \ldots maxindex]$

Reverse $A[1 \ldots i]$

7	7	6	8	9
7	7	6	8	9
6	7	7	8	9

$i = 3$; $maxindex = 1$

Reverse $A[1 \ldots maxindex]$

Reverse $A[1 \ldots i]$

6	7	7	8	9
7	6	7	8	9
6	7	7	8	9

$i = 2$; $maxindex = 2$

Reverse $A[1 \ldots maxindex]$

Reverse $A[1 \ldots i]$

Solutions $\rightarrow$ 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

$\textsc{StoogeSort}(A[\ell \ldots h])$

$size \leftarrow h - \ell + 1$
if $(A[\ell] > A[h])$ **then**
  $\textsc{Swap}(A[\ell], A[h])$
if $(size > 2)$ **then**
  $third \leftarrow size/3$
  $\textsc{StoogeSort}(A[\ell \ldots h - third])$
  $\textsc{StoogeSort}(A[\ell + third \ldots h])$
  $\textsc{StoogeSort}(A[\ell \ldots h - third])$

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n^{\log_{1.5} 3}\right), \Theta\left(\log n\right) \right\rangle$$

Solutions → Stooge sort

9	3	8	6	7	1	5	2	4	The original array
9	3	8	6	7	1	5	2	4	First (2/3)rd of the array
1	3	6	7	8	9	5	2	4	Sort the first (2/3)rd of the array
1	3	6	7	8	9	5	2	4	Last (2/3)rd of the array
1	3	6	2	4	5	7	8	9	Sort the last (2/3)rd of the array
1	3	6	2	4	5	7	8	9	First (2/3)rd of the array
1	2	3	4	5	6	7	8	9	Sort the first (2/3)rd of the array
1	2	3	4	5	6	7	8	9	The original array is sorted

Solutions → Counting sort

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 frequences
3. Compute the cumulative frequencies of items for indices in the range $[0, k]$
4. Find the sorted array

A	2	5	3	0	2	3	0	3	Unsorted array $A[1..n]$
C	2	0	2	3	0	1			Frequencies array $C[0..k]$
C	2	2	4	7	7	8			Cumulative frequencies array $C[0..k]$
B	0	0	2	2	3	3	3	5	Sorted array $B[1..n]$

Solutions → Counting sort

```
CountingSort(A[1...n])

k ← max(A[1...n])
Create new array B[1...n]
Create new array C[0...k] and initialize it to 0

// Find the frequencies of items
// After this step, C[i] will contain #elements equal to i
for j ← 1 to n do
|  C[A[j]] ← 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 ← 1 to k do
|  C[i] ← C[i] + C[i − 1]

// Get the sorted array in B
for j ← n downto 1 do
|  B[C[A[j]] ← A[j]
|  C[A[j]] ← C[A[j]] − 1

// Copy the sorted array to A
for j ← 1 to n do
|  A[j] ← B[j]
```

Solutions → Counting sort

A	2	5	3	0	2	3	0	3	$A[8] = 3$
C	2	2	4	7	7	8			$C[3] = 7$
B							3		$B[7] = 3$
C	2	2	4	6	7	8			$C[3] - -$

A	2	5	3	0	2	3	0	3	$A[7] = 0$
C	2	2	4	6	7	8			$C[0] = 2$
B		0					3		$B[2] = 0$
C	1	2	4	6	7	8			$C[0] - -$

A	2	5	3	0	2	3	0	3	$A[6] = 3$
C	1	2	4	6	7	8			$C[3] = 6$
B		0				3	3		$B[6] = 3$
C	1	2	4	5	7	8			$C[3] - -$

A	2	5	3	0	2	3	0	3	$A[5] = 2$
C	1	2	4	5	7	8			$C[2] = 4$
B		0		2		3	3		$B[4] = 2$
C	1	2	3	5	7	8			$C[2] - -$

A	2	5	3	0	2	3	0	3	$A[4] = 0$
C	1	2	3	5	7	8			$C[0] = 1$
B	0	0		2		3	3		$B[1] = 0$
C	0	2	3	5	7	8			$C[0] - -$

A	2	5	3	0	2	3	0	3	$A[3] = 3$
C	0	2	3	5	7	8			$C[3] = 5$
B	0	0		2	3	3	3		$B[5] = 3$
C	0	2	3	4	7	8			$C[3] - -$

A	2	5	3	0	2	3	0	3	$A[2] = 5$
C	0	2	3	4	7	8			$C[5] = 8$
B	0	0		2	3	3	3	5	$B[8] = 5$
C	0	2	3	4	7	7			$C[5] - -$

A	2	5	3	0	2	3	0	3	$A[1] = 2$
C	0	2	3	4	7	7			$C[2] = 3$
B	0	0	2	2	3	3	3	5	$B[3] = 2$
C	0	2	2	4	7	7			$C[2] - -$

Solutions → Counting sort variant

- 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

Solutions $\rightarrow$ Counting sort variant

```
CountingSortVariant(A[1...n])

(max, min) ← MaxMin(A[1...n])
size ← max − min + 1                    // size of range [min, max]
Create an array B[1...size] ← [0...0]

// Distribute array A elements to buckets in B
for j ← 1 to n do
|  i ← A[j] − min + 1; B[i] ← B[i] + 1

// Construct the sorted array A based on the bucket array
index ← 1
for i ← 1 to size do
|  while B[i] > 0 do
|  |  A[index] ← i + min − 1
|  |  index ← index + 1
|  |  B[i] ← B[i] − 1
```

Let $\#buckets = \max(A[1\ldots n]) - \min(A[1\ldots n])$

$$\langle \text{Time, Space} \rangle = \langle \Theta\left(n + \#buckets\right), \Theta\left(\#buckets\right) \rangle$$

Solutions → Counting sort variant

A	5	7	9	3	5	3	4	5
B	2	1	3	0	1	0	1	

B	$\boxed{2}$	1	3	0	1	0	1	
A	3							
B	$\boxed{1}$	1	3	0	1	0	1	

$index = 1$, $i = 1$

$A[index] = i + min - 1$

$B[i] - -$

B	$\boxed{1}$	1	3	0	1	0	1	
A	3	3						
B	$\boxed{0}$	1	3	0	1	0	1	

$index = 2$, $i = 1$

$A[index] = i + min - 1$

$B[i] - -$

B	0	$\boxed{1}$	3	0	1	0	1	
A	3	3	4					
B	0	$\boxed{0}$	3	0	1	0	1	

$index = 3$, $i = 2$

$A[index] = i + min - 1$

$B[i] - -$

B	0	0	$\boxed{3}$	0	1	0	1	
A	3	3	4	5				
B	0	0	$\boxed{2}$	0	1	0	1	

$index = 4$, $i = 3$

$A[index] = i + min - 1$

$B[i] - -$

B	0	0	$\boxed{2}$	0	1	0	1	
A	3	3	4	5	5			
B	0	0	$\boxed{1}$	0	1	0	1	

$index = 5$, $i = 3$

$A[index] = i + min - 1$

$B[i] - -$

B	0	0	$\boxed{1}$	0	1	0	1	
A	3	3	4	5	5	5		
B	0	0	$\boxed{0}$	0	1	0	1	

$index = 6$, $i = 3$

$A[index] = i + min - 1$

$B[i] - -$

B	0	0	0	$\boxed{0}$	1	0	1	
A	3	3	4	5	5	5		
B	0	0	0	$\boxed{0}$	1	0	1	

$index = 4$, $i = 4$

B	0	0	0	0	$\boxed{1}$	0	1	
A	3	3	4	5	5	5	7	
B	0	0	0	0	$\boxed{0}$	0	1	

$index = 7$, $i = 5$

$A[index] = i + min - 1$

$B[i] - -$

B	0	0	0	0	0	$\boxed{0}$	1	
A	3	3	4	5	5	5	7	
B	0	0	0	0	0	$\boxed{0}$	1	

$index = 8$, $i = 6$

B	0	0	0	0	0	0	$\boxed{1}$	
A	3	3	4	5	5	5	7	9
B	0	0	0	0	0	0	$\boxed{0}$	

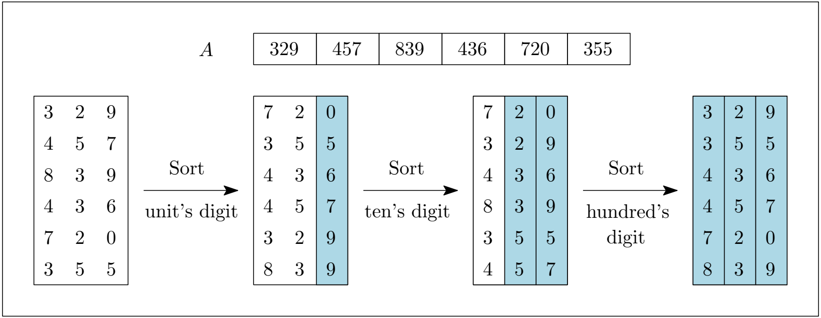
$index = 8$, $i = 7$

$A[index] = i + min - 1$

$B[i] - -$

Solutions → Radix sort

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



Solutions → Radix sort

```
RadixSort(A[1...n])
max ← Max(A[1...n]); exp ← 1
// Sort the array for each digit
while exp ≤ max do
    C[0...9] ← [0...0]; B[1...n] ← [0...0]
    // Find the cumulative frequencies of items
    for i ← 1 to n do
        { index ← ⌊A[i]/exp⌋ mod 10; C[index] ← C[index] + 1 }
    for i ← 1 to 9 do  C[i] ← C[i] + C[i − 1]
    // Populate output using count array
    for i ← n downto 1 do
        index ← ⌊A[i]/exp⌋ mod 10
        B[C[index]] ← A[i]
        C[index] ← C[index] − 1
    A[1...n] ← B[1...n]; exp ← exp × 10
```

$$\langle \text{Time, Space} \rangle = \langle \Theta(n \log n), \Theta(n) \rangle$$

Solutions → Radix sort

- Sort numbers based on the digits at unit's place

A	329	457	839	436	720	355					$A[6] = 355$
C	1	1	1	1	1	2	3	4	4	6	$C[5] = 2$
B		355									$B[2] = 355$
C	1	1	1	1	1	1	3	4	4	6	$C[5] - -$

A	329	457	839	436	720	355					$A[3] = 839$
C	0	1	1	1	1	1	2	4	4	6	$C[9] = 6$
B	720	355	436			839					$B[6] = 839$
C	0	1	1	1	1	1	2	4	4	5	$C[9] - -$

A	329	457	839	436	720	355					$A[5] = 720$
C	1	1	1	1	1	1	3	4	4	6	$C[0] = 1$
B	720	355									$B[1] = 720$
C	0	1	1	1	1	1	3	4	4	6	$C[0] - -$

A	329	457	839	436	720	355					$A[2] = 457$
C	0	1	1	1	1	1	2	4	4	5	$C[7] = 4$
B	720	355	436	457		839					$B[4] = 457$
C	0	1	1	1	1	1	2	3	4	5	$C[7] - -$

A	329	457	839	436	720	355					$A[4] = 436$
C	0	1	1	1	1	1	3	4	4	6	$C[6] = 3$
B	720	355	436								$B[3] = 436$
C	0	1	1	1	1	1	2	4	4	6	$C[6] - -$

A	329	457	839	436	720	355					$A[1] = 329$
C	0	1	1	1	1	1	2	3	4	5	$C[9] = 5$
B	720	355	436	457	329	839					$B[5] = 329$
C	0	1	1	1	1	1	2	3	4	4	$C[9] - -$

Solutions → Radix sort

- Sort numbers based on the digits at ten's place
- B from the previous iteration will be the A for this iteration.

A	720	355	436	457	329	839				
C	0	0	2	4	4	6	6	6	6	6
B				839						
C	0	0	2	3	4	6	6	6	6	6

$A[6] = 839$
$C[3] = 4$
$B[4] = 839$
$C[3] --$

A	720	355	436	457	329	839				
C	0	0	2	3	4	6	6	6	6	6
B		329		839						
C	0	0	1	3	4	6	6	6	6	6

$A[5] = 329$
$C[2] = 1$
$B[1] = 329$
$C[2] --$

A	720	355	436	457	329	839				
C	0	0	1	3	4	6	6	6	6	6
B		329		839		457				
C	0	0	1	3	4	5	6	6	6	6

$A[4] = 457$
$C[5] = 6$
$B[6] = 457$
$C[5] --$

A	720	355	436	457	329	839				
C	0	0	1	3	4	5	6	6	6	6
B		329	436	839		457				
C	0	0	1	2	4	5	6	6	6	6

$A[3] = 436$
$C[3] = 3$
$B[3] = 436$
$C[3] --$

A	720	355	436	457	329	839				
C	0	0	1	2	4	5	6	6	6	6
B		329	436	839	355	457				
C	0	0	1	2	4	4	6	6	6	6

$A[2] = 355$
$C[5] = 5$
$B[5] = 355$
$C[5] --$

A	720	355	436	457	329	839				
C	0	0	1	2	4	4	6	6	6	6
B	720	329	436	839	355	457				
C	0	0	0	2	4	4	6	6	6	6

$A[1] = 720$
$C[2] = 1$
$B[1] = 720$
$C[2] --$

Solutions → Radix sort

- Sort numbers based on the digits at hundred's place
- B from the previous iteration will be the A for this iteration.

A	720		329		436		839		355		457	
C	0	0	0	2	$\boxed{4}$	4	4	5	6	6		
B							457					
C	0	0	0	2	$\boxed{3}$	4	4	5	6	6		

$A[6] = 457$
$C[4] = 4$
$B[4] = 457$
$C[4] - -$

A	720		329		436		839		355		457	
C	0	0	0	$\boxed{2}$	3	4	4	5	6	6		
B			355				457					
C	0	0	0	$\boxed{1}$	3	4	4	5	6	6		

$A[5] = 355$
$C[3] = 2$
$B[2] = 355$
$C[3] - -$

A	720		329		436		839		355		457	
C	0	0	0	1	3	4	4	5	$\boxed{6}$	6		
B			355				457				839	
C	0	0	0	1	3	4	4	5	$\boxed{5}$	6		

$A[4] = 839$
$C[8] = 6$
$B[6] = 839$
$C[8] - -$

A	720		329		436		839		355		457	
C	0	0	0	1	$\boxed{3}$	4	4	5	5	6		
B			355		436		457				839	
C	0	0	0	1	$\boxed{2}$	4	4	5	5	6		

$A[3] = 436$
$C[4] = 3$
$B[3] = 436$
$C[4] - -$

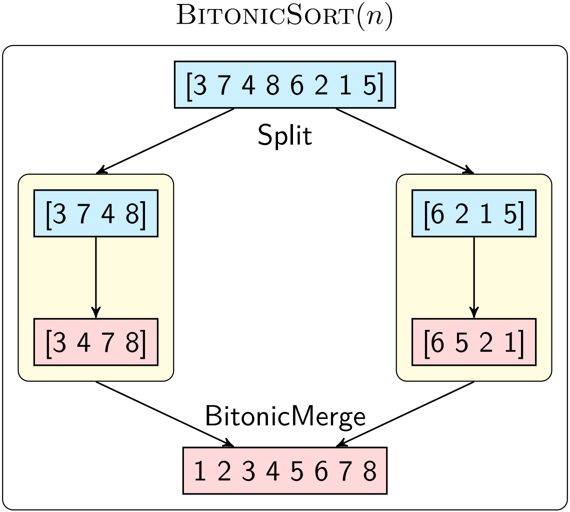
A	720		329		436		839		355		457	
C	0	0	0	$\boxed{1}$	2	4	4	5	5	6		
B	329		355		436		457				839	
C	0	0	0	$\boxed{0}$	2	4	4	5	5	6		

$A[2] = 329$
$C[3] = 1$
$B[1] = 329$
$C[3] - -$

A	720		329		436		839		355		457	
C	0	0	0	0	2	4	4	$\boxed{5}$	5	6		
B	329		355		436		457		720		839	
C	0	0	0	0	2	4	6	$\boxed{4}$	5	6		

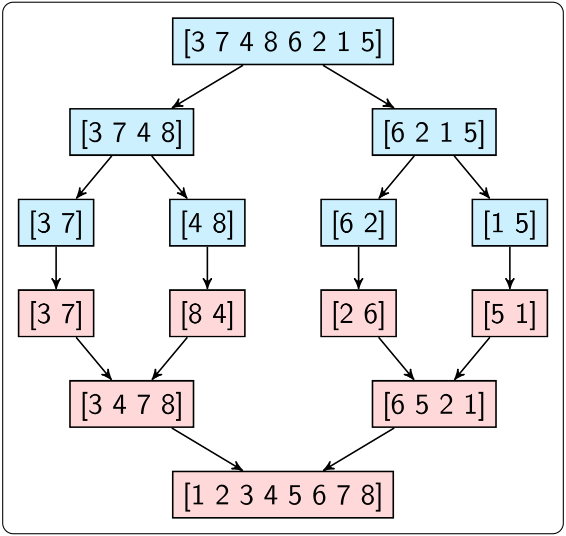
$A[1] = 720$
$C[7] = 5$
$B[5] = 720$
$C[7] - -$

Solutions → Bitonic sort



Solutions → Bitonic sort

BITONICSORT(n)



Solutions $\rightarrow$ Bitonic sort

- Invoke $\textsc{BitonicSort}(A[1 \ldots n], ascending)$

$\textsc{BitonicSort}(A[\ell \ldots h], order)$

$size \leftarrow h - \ell + 1$
if $size > 1$ **then**
    $m \leftarrow (\ell + h)/2$
    $\textsc{BitonicSort}(A[\ell \ldots m], ascending)$
    $\textsc{BitonicSort}(A[m+1 \ldots h], descending)$
    $\textsc{BitonicMerge}(A[\ell \ldots h], order)$

Solutions $\rightarrow$ Bitonic sort

$\textsc{BitonicMerge}(A[\ell \ldots h], order)$

Input: Array $A[\ell \ldots h]$, ascending/descending order
Output: Bitonic merge the array
$size \leftarrow h - \ell + 1$
if $size > 1$ **then**
  $m \leftarrow (\ell + h)/2$
  $\textsc{Compare\&Swap}(A[\ell \ldots h], order)$
  $\textsc{BitonicMerge}(A[\ell \ldots m], order)$
  $\textsc{BitonicMerge}(A[m + 1 \ldots h], order)$

$\textsc{Compare\&Swap}(A[\ell \ldots h], order)$

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**
    $\textsc{Swap}(A[i], A[j])$

Solutions → Bitonic sort

$$\langle \text{Time, Space} \rangle = \left\langle \Theta\left(n \log^2 n\right), \Theta\left(n\right) \right\rangle$$

$$T(n) = \left\{ \begin{array}{ll} \Theta\left(1\right) & \text{if } n = 1, \\ 2T(n/2) + T^{\text{merge}}(n/2) & \text{if } n > 1. \end{array} \right\}$$

$$T^{\text{merge}}(n) = \left\{ \begin{array}{ll} \Theta\left(1\right) & \text{if } n = 1, \\ 2T^{\text{merge}}(n/2) + \Theta\left(n\right) & \text{if } n > 1. \end{array} \right\}$$

Contributors

Tejas Bhatia, Usha Vudatha, Abiyaz Chowdhury, Taha Kothawala, Ajay Hegde, Sai Sujith Bezawada