Algorithms
(Decrease-and-Conquer)

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

October 19, 2021
1. Decrease by constant
   • Topological sorting
2. Decrease by constant factor
   • Lighter ball
   • Josephus problem
3. Variable size decrease
   • Selection problem
Decrease-and-conquer

Problem($n$)

Subproblem($n'$)

Subsolution

Solution

[Step 1. Decrease]

[Step 2. Conquer]

[Step 3. Combine]
Types of decrease-and-conquer

- Decrease by constant. $n' = n - c$ for some constant $c$
- Decrease by constant factor. $n' = \frac{n}{c}$ for some constant $c$
- Variable size decrease. $n' = n - c$ for some variable $c$
Decrease by constant

- Size of instance is reduced by the same constant in each iteration of the algorithm
- Decrease by 1 is common
- Examples:
  - Array sum
  - Array search
  - Find maximum/minimum element
  - Integer product
  - Exponentiation
  - Topological sorting
Decrease by constant factor

- Size of instance is reduced by the same constant factor in each iteration of the algorithm
- Decrease by factor 2 is common
- Examples:
  - Binary search
  - Search/insert/delete in balanced search tree
  - Fake coin problem
  - Josephus problem
Variable size decrease

- Size of instance is reduced by a variable in each iteration of the algorithm
- Examples:
  - Selection problem
  - Quicksort
  - Search/insert/delete in binary search tree
  - Interpolation search
Topological sorting

Problem

- Topological sorting of vertices of a directed acyclic graph is an ordering of the vertices $v_1, v_2, \ldots, v_n$ in such a way that there is an edge directed towards vertex $v_j$ from vertex $v_i$, then $v_i$ comes before $v_j$. 
Graph

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>B</td>
<td>E</td>
</tr>
</tbody>
</table>

Topological sort

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

Does not exist
Topological sort = Reversal of the order in which the vertices become dead ends in the DFS algorithm.

**TopologicalSort**($G$)

1. Topological sort $T \leftarrow \emptyset$
2. Mark each vertex in $V$ as unvisited
3. for each vertex $v$ in $V$ do
4.   if $v$ is unvisited then
5.       DFS($v$)
6.   return $T$

**DFS**($v$)

1. Mark $v$ as visited
2. for each vertex $w$ in $V$ adjacent to $v$ do
3.   if $w$ is unvisited then
4.       DFS($w$)
5.       $T$.AddFirst($v$)
Source removal algorithm

Topological sort = Order in which those vertices are removed that have 0 indegrees.
source-removal algorithm

**TopologicalSort**($G$)

1. Topological sort $T \leftarrow \emptyset$
2. Mark each vertex in $V$ as unvisited
3. Find $\text{indegree}[v]$ for each vertex $v$ in $V$
4. for each vertex $v$ in $V$ do
5.   if $\text{indegree}[v] = 0$ then
6.     $Q.\text{ENQUEUE}(v)$
7.     Mark $v$ as visited
8. while $Q$ is not empty do
9.   $u \leftarrow Q.\text{DEQUEUE}()$
10.  $T.\text{ADDLAST}(u)$
11. for each vertex $w$ in $V$ adjacent to $u$ do
12.   if $w$ is unvisited then
13.     $\text{indegree}[w] \leftarrow \text{indegree}[w] - 1$
14.   if $\text{indegree}[w] = 0$ then
15.     $Q.\text{ENQUEUE}(w)$
16.     Mark $w$ as visited
17. return $T$
Decrease by Constant Factor
## Lighter ball

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>There are $n \geq 1$ identical-looking balls, but, one of the balls is lighter than the other balls. Design an efficient algorithm to detect the lighter ball using a weighing scale/balance.</td>
</tr>
</tbody>
</table>
Lighter ball among 9 balls: Divide by 2
Decrease-by-half algorithm

\textbf{LighterBall}([\ell, \ell + 1, \ell + 2, \ldots, h])

\textbf{Input:} Set of \((h - \ell + 1)\) balls: \(\ell, \ell + 1, \ell + 2, \ldots, h\)

\textbf{Output:} Index number of the lighter ball

\textbf{Require:} Invocation is \textbf{LighterBall}([0..n - 1]) such that \(n \geq 2\)

1. \textbf{if} \(\ell = h\) \textbf{then}
2. \quad \textbf{return} \(\ell\)
3. \quad \textbf{half} \leftarrow \left\lfloor (h - \ell + 1)/2 \right\rfloor
4. \quad \textbf{A} \leftarrow \text{first half number of balls i.e., } [\ell, \ell + 1, \ldots, \ell + \text{half} - 1]
5. \quad \textbf{B} \leftarrow \text{second half number of balls i.e., } [\ell + \text{half}, \ldots, \ell + 2 \cdot \text{half}]
6. \quad \textbf{C} \leftarrow \text{remaining ball } [h] \text{ if total balls is odd}
7. \quad \text{weigh sets } A \text{ and } B
8. \quad \textbf{if} \ \text{weight}(A) < \text{weight}(B) \ \textbf{then}
9. \quad \quad \textbf{return} \ \textbf{LighterBall}(A)
10. \quad \textbf{else if} \ \text{weight}(A) > \text{weight}(B) \ \textbf{then}
11. \quad \quad \textbf{return} \ \textbf{LighterBall}(B)
12. \quad \textbf{else if} \ \text{weight}(A) = \text{weight}(B) \ \textbf{then}
13. \quad \quad \textbf{return} \ \textbf{LighterBall}(C)
Complexity

- **Weighings.**

\[
W(n) = \begin{cases} 
0 & \text{if } n = 1, \\
W(\lfloor n/2 \rfloor) + 1 & \text{if } n \geq 2.
\end{cases}
\]

Solving, \( W(n) = \lfloor \log_2 n \rfloor \)

- **Time complexity.**

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1, \\
T(n/2) + \Theta(1) & \text{if } n \geq 2.
\end{cases}
\]

Solving, \( T(n) \in \Theta(\log n) \)
Lighter ball among 9 balls: Divide by 3
Decrease-by-third algorithm

\textbf{LighterBall}([\ell, \ell + 1, \ell + 2, \ldots, h])

\textbf{Input:} Set of \((h - \ell + 1)\) balls: \(\ell, \ell + 1, \ell + 2, \ldots, h\)
\textbf{Output:} Index number of the lighter ball
\textbf{Require:} Invocation is \textbf{LighterBall}([0..n - 1]) such that \(n \geq 3\)

1. if \(\ell = h\) then \(\triangleright 1\) ball
2. return \(\ell\)
3. else if \(\ell = h - 1\) then \(\triangleright 2\) balls
4. return lighter ball among \(\ell\) and \(h\)
5. \(\text{third} \leftarrow \left\lfloor (h - \ell + 1)/3 \right\rfloor\)
6. \(A \leftarrow \text{first third number of balls i.e., } [\ell, \ell + 1, \ldots, \ell + \text{third} - 1]\)
7. \(B \leftarrow \text{second third number of balls i.e., } [\ell + \text{third}, \ldots, \ell + 2 \cdot \text{third} - 1]\)
8. \(C \leftarrow \text{remaining balls, i.e., } [\ell + 2 \cdot \text{third}, \ldots, h]\)
9. weigh sets \(A\) and \(B\)
10. if weight\((A) < \text{weight}(B)\) then
11. return \textbf{LighterBall}(A)
12. else if weight\((A) > \text{weight}(B)\) then
13. return \textbf{LighterBall}(B)
14. else if weight\((A) = \text{weight}(B)\) then
15. return \textbf{LighterBall}(C)
Complexity

• Weighings.

\[ W(n) = \begin{cases} 
0 & \text{if } n = 1, \\
1 & \text{if } n = 2, \\
W(\lceil n/3 \rceil) + 1 & \text{if } n \geq 3. 
\end{cases} \]

Solving, \( W(n) = \lceil \log_3 n \rceil \)

• Time complexity.

\[ T(n) = \begin{cases} 
\Theta (1) & \text{if } n = 1 \text{ or } 2, \\
T(n/3) + \Theta (1) & \text{if } n \geq 3. 
\end{cases} \]

Solving, \( T(n) \in \Theta (\log n) \)
Problem

* There are $n$ people numbered from 1 to $n$ in a circle. Starting from person 1, we eliminate every second person until only survivor is left. Design an efficient algorithm to find the survivor’s number $J(n)$. 
Example: \( J(6) = 5 \)
CLL algorithm

- Create a circular linked list (CLL) of size \( n \).
- Node at location \( i \) stores item \( i \).
- Delete alternate nodes until only one node is left.
- Time is \( \Theta(n) \), space is \( \Theta(n) \)
- Is there a more efficient algorithm?
Decrease-by-half algorithm

\[ J(n) = \begin{cases} 
1 & \text{if } n = 1, \\
2J(n/2) - 1 & \text{if } n \geq 2 \text{ and } n \text{ is even}, \\
2J(n/2) + 1 & \text{if } n \geq 2 \text{ and } n \text{ is odd.} 
\end{cases} \]

\begin{tabular}{ | l | l | }
\hline
**JOSEPHUS(\(n\))** & \\
\hline
**Input:** Whole number \(n\) & \\
**Output:** Josephus number \(J(n)\) & \\
1. if \(n = 1\) then & \\
2. return 1 & \\
3. else if \(n\) is even then & \\
4. return \(2 \times \text{JOSEPHUS}(n/2) - 1\) & \\
5. else if \(n\) is odd then & \\
6. return \(2 \times \text{JOSEPHUS}(n/2) + 1\) & \\
\hline
\end{tabular}
Case: \( n \) is even \( (n = 2k) \)

\[
J(6) = 2J(3) - 1
\]

\[
J(2k) = 2J(k) - 1
\]
Case: $n$ is odd ($n = 2k + 1$)

\[ J(7) = 2J(3) + 1 \]

\[ J(2k + 1) = 2J(k) + 1 \]
• **Time complexity.**

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1, \\
T(n/2) + \Theta(1) & \text{if } n > 1.
\end{cases}
\]

Solving, \( T(n) \in \Theta(\log n) \)

• **Space complexity.**

\[
S(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1, \\
S(n/2) + \Theta(1) & \text{if } n > 1.
\end{cases}
\]

Solving, \( S(n) \in \Theta(\log n) \) stack space
Variable Size Decrease
Selection problem

Problem

- Find the $k$th smallest element (or $k$th order statistic) in a given array $A[0..n-1]$.

- **Easiest cases.** Minimum ($k = 1$), maximum ($k = n$)
- **Hardest case.** Median ($k = \lfloor n/2 \rfloor$)
# Selection algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>$k$-smallest items?</th>
<th>Sorted?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorting</td>
<td>$\Theta(n \log n)$</td>
<td>✓, ✓</td>
<td></td>
</tr>
<tr>
<td>Partial selection sort</td>
<td>$\Theta(kn)$</td>
<td>✓, ✓</td>
<td></td>
</tr>
<tr>
<td>Partial heapsort</td>
<td>$\Theta(n + k \log n)$</td>
<td>✓, x</td>
<td></td>
</tr>
<tr>
<td>Online selection</td>
<td>$\Theta(n \log k)$</td>
<td>✓, x</td>
<td></td>
</tr>
<tr>
<td>Rand. quickselect</td>
<td>$\Theta(n^2)$ ((\Theta(n)) avg.)</td>
<td>✓, x</td>
<td></td>
</tr>
<tr>
<td>Linear-time algorithm</td>
<td>$\Theta(n)$</td>
<td>x, x</td>
<td></td>
</tr>
</tbody>
</table>
Partial selection sort

<table>
<thead>
<tr>
<th>PartialSelectionSort(A[0..(n – 1)])</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Run SelectionSort on A[0..(n – 1)] for k iterations to find the k smallest elements in sorted order</td>
</tr>
<tr>
<td>2. return kth smallest element</td>
</tr>
</tbody>
</table>

Time is \( \Theta (kn) \)
### PartialHeapsort($A[0..(n - 1)]$)

1. $H \leftarrow$ Construct a min-heap from $A[0..(n - 1)]$ in-place $\quad \triangleright \Theta (n)$
2. $H.\text{DELETEMIN}()$ $k$ times $\quad \triangleright \Theta (k \log n)$
3. **return** $k$th smallest element

**Time is** $\Theta (n + k \log n)$
Online selection

\begin{algorithm}
\begin{algorithmic}
\STATE OnlineSelection(A[0..(n – 1)])
\STATE 1. $H \leftarrow \text{Construct a } k\text{-sized max-heap from } A[0..(k – 1)] \quad \triangleright \Theta (k)$
\STATE 2. \textbf{for} $i \leftarrow k$ \textbf{to} $(n – 1)$ \textbf{times do} \quad \triangleright \Theta (n \log k)$
\STATE 3. \textbf{if} $A[i]$ \text{is not more than the heap’s maximum} \textbf{then}
\STATE 4. \hspace{1em} $H$.\text{INSERT}(A[i])
\STATE 5. \hspace{1em} $H$.\text{DELETEMAX}()
\STATE 6. \text{\hspace{1em} $k$th smallest element } \leftarrow H$.\text{DELETEMAX}() \quad \triangleright \Theta (\log k)$
\STATE 7. \textbf{return} $k$th smallest element
\end{algorithmic}
\end{algorithm}

Time is $\Theta (n \log k)$
Randomized quickselect

**RandomizedQuickSelect**(\(A[\ell..h], k\))

1. if \(\ell = h\) then
2. return \(A[\ell]\)
3. \(s \leftarrow \text{RandomizedPartition}(A[\ell..h])\)
4. \(\text{size} \leftarrow s - \ell + 1\)
5. if \(k = \text{size}\) then
6. return \(A[s]\)
7. else if \(k < \text{size}\) then
8. return **RandomizedQuickSelect**(\(A[\ell..s - 1], k\))
9. else if \(k > \text{size}\) then
10. return **RandomizedQuickSelect**(\(A[s + 1..h], k - \text{size}\))
Randomized partition
### Randomized Partition (using Lomuto partition)

**RandomizedPartition**($A[\ell..h]$)

1. $i \leftarrow \text{Random}([\ell, \ell + 1, \ldots, h])$
2. \text{Swap}($A[\ell], A[i]$)
3. \text{LomutoPartition}($A[\ell..h]$)

**LomutoPartition**($A[\ell..h]$)

1. $\text{pivot} \leftarrow A[\ell]$ \hspace{1cm} $\triangleright$ first element is the pivot
2. $i \leftarrow \ell$
3. \textbf{for} $j \leftarrow \ell + 1 \textbf{ to } h$ \textbf{do}
4. \hspace{0.5cm} \textbf{if} $A[j] \leq \text{pivot}$ \textbf{then}
5. \hspace{1cm} $i \leftarrow i + 1$
6. \hspace{0.5cm} \text{Swap}($A[i], A[j]$)
7. \text{Swap}($\text{pivot}, A[i]$)
8. \textbf{return} $i$
Lomuto partition

pivot

\[ \begin{array}{cccccc}
\text{l} & 3 & 8 & 6 & 7 & 1 & 5 & 2 & 4 \\
i & \text{j} & & & & & & & \\
\end{array} \]

\[ i \leftarrow \ell \text{ and } j \leftarrow i + 1. \quad A[j] > \text{pivot}. \]

Increment \( j \) until \( A[j] \leq \text{pivot} \).

Increment \( i \).

Swap \( A[i] \) and \( A[j] \).

Increment \( j \) until \( A[j] \leq \text{pivot} \).

Increment \( i \).

Swap \( A[i] \) and \( A[j] \).

Increment \( j \) until \( j = h + 1 \).

Swap \( \text{pivot} \) and \( A[i] \). Return \( i \).
Randomized partition (using Hoare partition)

**RandomizedPartition**($A[\ell..h]$)

1. $i \leftarrow \text{Random} (\{\ell, \ell+1, \ldots, h\})$
2. $\text{Swap}(A[\ell], A[i])$
3. $\text{HoarePartition}(A[\ell..h])$

**HoarePartition**($A[\ell..h]$)

1. $\text{pivot} \leftarrow A[\ell]$ ▷ first element is the pivot
2. $i \leftarrow \ell; j \leftarrow h + 1$
3. **while** true **do**
4. 
5. **while** $A[++i] < \text{pivot}$ **do**
6. \hspace{1em} **if** $i = h$ **then** break
7. **while** $\text{pivot} < A[--j]$ **do**
8. \hspace{1em} **if** $j = \ell$ **then** break
9. \hspace{1em} **if** $i \geq j$ **then** break
10. \hspace{1em} **else** $\text{Swap}(A[i], A[j])$
11. 
12. $\text{Swap}(\text{pivot}, A[j])$
13. **return** $j$
Initially, $i \leftarrow \ell$ and $j \leftarrow h + 1$.


Break loop because $j \leq i$.


Swapping elements: 

Initially, $i \leftarrow \ell$ and $j \leftarrow h + 1$.


Break loop because $j \leq i$.


Swapping elements: