Algorithms
(Decrease-and-Conquer)

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

May 30, 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. $n' = n/c$ 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
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
Decrease by Constant
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 $v_j$. 

**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 $v_j$. 

Example

Graph

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

Topological sort

\[ [A, B, C, D, E] \]
\[ [B, A, C, D, E] \]

\[ [A, B, C, D, E] \]
\[ [B, A, C, D, E] \]

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

<table>
<thead>
<tr>
<th><strong>TOPOLOGICALSORT</strong>($G'$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Topological sort $T \leftarrow \emptyset$</td>
</tr>
<tr>
<td>2. Mark each vertex in $V$ as unvisited</td>
</tr>
<tr>
<td>3. for each vertex $v$ in $V$ do</td>
</tr>
<tr>
<td>4. if $v$ is unvisited then</td>
</tr>
<tr>
<td>5. DFS($v$)</td>
</tr>
<tr>
<td>6. return $T$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>DFS($v$)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Mark $v$ as visited</td>
</tr>
<tr>
<td>2. for each vertex $w$ in $V$ adjacent to $v$ do</td>
</tr>
<tr>
<td>3. if $w$ is unvisited then</td>
</tr>
<tr>
<td>4. DFS($w$)</td>
</tr>
<tr>
<td>5. $T$.addFirst($v$)</td>
</tr>
</tbody>
</table>
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.     $Q$.enqueue($v$)
7.   Mark $v$ as visited
8. while $Q$ is not empty do
9.   $u \leftarrow Q$.dequeue()
10. $T$.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.     $Q$.enqueue($w$)
16.   Mark $w$ as visited
17. return $T$
Decrease by Constant Factor
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.
Lighter ball among 9 balls: Divide by 2
**Decrease-by-half algorithm**

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

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

**Output:** Index number of the lighter ball

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

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

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

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

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

**JOSEPHUS(n)**

**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 \( 2 \times \text{JOSEPHUS}(n/2) + 1 \)
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

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Find the $k$th smallest element (or $k$th order statistic) in a given array $A[0..n-1]$.</td>
</tr>
</tbody>
</table>

• **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>✓, ✓</td>
<td></td>
</tr>
<tr>
<td>Online selection</td>
<td>$\Theta(n \log k)$</td>
<td>✓, ✓</td>
<td></td>
</tr>
<tr>
<td>Rand. quickselect</td>
<td>$\Theta(n^2)$ ($\Theta(n)$ avg.)</td>
<td>✓, ✓</td>
<td></td>
</tr>
<tr>
<td>Linear-time algorithm</td>
<td>$\Theta(n)$</td>
<td>✓, ✓</td>
<td></td>
</tr>
</tbody>
</table>
Partial selection sort

\[
\text{PARTIALSELECTIONSORT}(A[0..(n − 1)])
\]

1. Run \text{SELECTIONSORT} on \(A[0..(n − 1)]\) for \(k\) iterations to find the \(k\) smallest elements in sorted order
2. \text{return} \(k\)th smallest element

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

1. Construct a min-heap from $A[0..(n-1)]$ in-place \(\triangleright \Theta(n)\)
2. DeleteMin $k$ times \(\triangleright \Theta(k \log n)\)
3. **return** $k$th smallest element

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

1. Construct a separate $k$-sized max-heap from $A[0..(k-1)]$ \(\triangleright \Theta (k)\)
2. for $i \leftarrow k$ to $(n-1)$ times do \(\triangleright \Theta (n \log k)\)
3. if $A[i]$ is not more than the heap’s maximum then
4. Insert $A[i]$ to the heap
5. DeleteMax
6. $k$th smallest element $\leftarrow$ DeleteMax \(\triangleright \Theta (\log k)\)
7. return $k$th smallest element

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

\[ \text{RandomizedQuickSelect}(A[\ell..h], k) \]

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 \( \text{RandomizedQuickSelect}(A[\ell..s - 1], k) \)
9. else if \( k > \text{size} \) then
10. return \( \text{RandomizedQuickSelect}(A[s + 1..h], k - \text{size}) \)
Randomized partition

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

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

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

Increment \( i \).

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

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

Increment \( i \).

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

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

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

```
RandomizedPartition(A[ℓ..h])

1. i ← Random({ℓ, ℓ + 1, . . . , h})
2. Swap(A[ℓ], A[i])
3. HoarePartition(A[ℓ..h])

HoarePartition(A[ℓ..h])

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


Break loop because $j \leq i$.