Algorithms
(Decrease-and-Conquer)

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

August 22, 2021
Concepts
1. Decrease by constant
   • Topological sorting
2. Decrease by constant factor
   • Lighter ball
   • Josephus problem
3. Variable size decrease
   • Selection problem

Problems
• Stooge sort
Contributors

- Ayush Sharma
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 \textit{reduced by a variable} in each iteration of the algorithm
- Examples:
  - \textit{Selection problem}
  - Quicksort
  - Search/insert/delete in binary search tree
  - Interpolation search
Decrease by Constant
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 $v_j$. 
Example

<table>
<thead>
<tr>
<th>Graph</th>
<th>Topological sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>A → C → D → E</td>
<td>[A, B, C, D, E]</td>
</tr>
<tr>
<td>B → C → D → E</td>
<td>[B, A, C, D, E]</td>
</tr>
<tr>
<td>A → C → D → E</td>
<td>Does not exist</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>TopologicalSort($G$)</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. Find $\text{indegree}[v]$ for each vertex $v$ in $V$</td>
</tr>
<tr>
<td>4. for each vertex $v$ in $V$ do</td>
</tr>
<tr>
<td>5. if $\text{indegree}[v] = 0$ then</td>
</tr>
<tr>
<td>6. $Q$.enqueue($v$)</td>
</tr>
<tr>
<td>7. Mark $v$ as visited</td>
</tr>
<tr>
<td>8. while $Q$ is not empty do</td>
</tr>
<tr>
<td>9. $u \leftarrow Q$.dequeue()</td>
</tr>
<tr>
<td>10. $T$.addLast($u$)</td>
</tr>
<tr>
<td>11. for each vertex $w$ in $V$ adjacent to $u$ do</td>
</tr>
<tr>
<td>12. if $w$ is unvisited then</td>
</tr>
<tr>
<td>13. $\text{indegree}[w] \leftarrow \text{indegree}[w] - 1$</td>
</tr>
<tr>
<td>14. if $\text{indegree}[w] = 0$ then</td>
</tr>
<tr>
<td>15. $Q$.enqueue($w$)</td>
</tr>
<tr>
<td>16. Mark $w$ as visited</td>
</tr>
<tr>
<td>17. return $T$</td>
</tr>
</tbody>
</table>
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

**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 **LighterBall**([0..n − 1]) such that \(n \geq 2\)

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

**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 **LighterBall**([0..\(n - 1\)]) such that \(n \geq 3\)

1. **if** \(\ell = h\) **then** ▷ 1 ball
2. **return** \(\ell\)
3. **else if** \(\ell = h - 1\) **then** ▷ 2 balls
4. **return** lighter ball among \(\ell\) and \(h\)
5. \(\text{third} \leftarrow \lceil (h - \ell + 1)/3 \rceil\)
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** **LighterBall**(\(A\))
12. **else if** weight\((A) > \text{weight}(B)\) **then**
13. **return** **LighterBall**(\(B\))
14. **else if** weight\((A) = \text{weight}(B)\) **then**
15. **return** **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, \( T(n) \in \Theta(\log n) \)
Josephus problem

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
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>✓, 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

PartialSelectionSort(A[0..(n − 1)])

1. Run SelectionSort on A[0..(n − 1)] for k iterations to find the k smallest elements in sorted order
2. return kth smallest element

Time is $\Theta(kn)$
Partial heapsort

**PartialHeapsort**($A[0..(n-1)]$)

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

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

\[
\text{OnlineSelection}(A[0..(n-1)])
\]

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

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

1. if $\ell = h$ then
2. return $A[\ell]$
3. $s \leftarrow$ RandomizedPartition($A[\ell..h]$)
4. $size \leftarrow s - \ell + 1$
5. if $k = size$ then
6. return $A[s]$
7. else if $k < size$ then
8. return RandomizedQuickSelect($A[\ell..s - 1], k$)
9. else if $k > size$ then
10. return RandomizedQuickSelect($A[s + 1..h], k - size$)
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. Swap$(A[\ell], A[i])$
3. **LomutoPartition**(A[\ell..h])

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

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

\[
\begin{array}{ccccccc}
\text{pivot} & \ell & 3 & 8 & 6 & 7 & 1 & 5 & 2 & 4 & h \\
& i & j \\
\end{array}
\]

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

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

**HoarePartition**(A[ℓ..h])

1. \( \text{pivot} \leftarrow A[\ell] \) \( \triangleright \) 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. if \( i = h \) then break
7. while \( \text{pivot} < A[--j] \) do
8. if \( j = \ell \) then break
9. if \( i \geq j \) then break
10. 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 $j \leq i$.

Step 1. Problem

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Sort a given $n$-sized array in nondecreasing order.</td>
</tr>
</tbody>
</table>
Step 2. Subproblem

\( \text{Sort}(A[\ell..h]) = \text{Sort all elements in subarray } A[\ell..h] \)
\text{in nondecreasing order.}

\text{Compute } \text{Sort}(A[1..n]).
### Step 3. Core idea

<table>
<thead>
<tr>
<th>9</th>
<th>3</th>
<th>8</th>
<th>6</th>
<th>7</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>The original array</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9</th>
<th>3</th>
<th>8</th>
<th>6</th>
<th>7</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>First (2/3)rd of the array</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>5</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort the first (2/3)rd of the array</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>5</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last (2/3)rd of the array</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>6</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort the last (2/3)rd of the array</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>6</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>First (2/3)rd of the array</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort the first (2/3)rd of the array</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>The original array is sorted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 4. Algorithm

StoogeSort(A[\ell..h])

Input: An array A[\ell..h]
Output: Array A[\ell..h] sorted in nondecreasing order
1. \(size \leftarrow h - \ell + 1\)
2. if \(size > 1\) then
3. if (A[\ell] > A[h]) then Swap(A[\ell], A[h])
4. if (size > 2) then
5. \(third \leftarrow size/3\)
6. StoogeSort(A[\ell..h - third])
7. StoogeSort(A[\ell + third..h])
8. StoogeSort(A[\ell..h - third])
Step 5. Complexity

- Time complexity.

\[ T_{\text{SORT}}(n) = \begin{cases} 
\Theta(1) & \text{if } n = 2, \\
3T_{\text{SORT}}(2n/3) + \Theta(n) & \text{if } n > 2. 
\end{cases} \]

Solving, \( T_{\text{SORT}}(n) \in \Theta(n^{\log_{1.5}3}) \)

- Space complexity.

\[ S(n) \in \Theta(n) \]