Week 4.3, Friday, Sept 13

Homework 2 Due: September 16th, 2019 @ 11:59PM on Gradescope
September 16: Guest Lecture (Prof. Hambrusch)
Office Hours: Monday (9/16) at 2:30PM - Friday (today) at 4:30 PM
Homework 2 Reminders

- Must include collaborator/resource statement
  - No credit for solutions that don’t include CR statement
- Must type solutions (expectation to use math notation $\sqrt{n}$ vs $n^{(1/2)}$)
  - Only allowed to scan hand drawn graphs/diagrams
  - No credit if the entire solution is a scan of your handwritten homework
- Remember to select the appropriate pages for each problem on Gradescope
Majority Element Problem

Input: Array A[1...n] of numbers (not sorted)
Output:

x      ---- if x=A[i] for more than n/2 array elements
“N/A”  ---- if no majority element exists

Example 1:
Input:  A = [ 1 7 2 9 7 2 7]
Output: “N/A”

Example 2:
Input:  A = [ 1 7 2 7 7 2 7]
Output: 7

Observation: If A does contain a majority element x then x=Median(A)
Suppose $A$ is an array of size $n$ containing increasingly sorted entries. We can determine whether $A$ has a majority element in what time (check best bound):

- A. $O(1)$
- B. $O(\log n)$
- C. $O(\log^2 n)$
- D. $O(n)$
- E. $O(n \log n)$
**Majority Element Problem**

**Input:** Array $A[1...n]$ of numbers (not sorted)

**Output:**
- $x$ ---- if $x = A[i]$ for more than $n/2$ array elements
- “N/A” ---- if no majority element exists

**Solution 1:** Sort array, find median $x$ and binary search $O(n \log n)$
- $i :=$ unique location s.t. $A[i] \neq x$ but $A[i+1] = x$ (first occurrence of $x$)
- $j :=$ unique location s.t. $A[j] \neq x$ but $A[j-1] = x$ (last occurrence of $x$)
- Return
  \[
  \begin{cases}
    \text{N/A} & j - i - 1 \leq \frac{n}{2} \\
    x = A\left[\left\lceil \frac{n}{2} \right\rceil \right] & j - i - 1 > \frac{n}{2}
  \end{cases}
  \]

**Example:**
- Input: $A = [1, 2, 7, 7, 7, 7, 8]$
- Output: 7
Majority Element Problem

**Input:** Array $A[1...n]$ of numbers (not sorted)

**Output:**
- $x$ ---- if $x=A[i]$ for more than $n/2$ array elements
- "N/A" ---- if no majority element exists

**Solution 2:** Find Median and Scan Array to Count Matches

- $X = \text{Median}(A)$
- Count = 0
- For $i = 1$ to $n$
  - If $X = A[i]$ then Count = Count + 1;
- Return
  \[
  \begin{cases}
  N/A & \text{count} \leq \frac{n}{2} \\
  x = A \left\lfloor \frac{m}{2} \right\rfloor & \text{count} > \frac{n}{2}
  \end{cases}
  \]

**Running time:** $O(n) + O(n) = O(n)$
4.1 Interval Scheduling
Interval scheduling.

- Job $j$ starts at $s_j$ and finishes at $f_j$.
- Two jobs compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.
Interval Scheduling: Greedy Algorithms

**Greedy template.** Consider jobs in some natural order. Take each job provided it's compatible with the ones already taken.

- **[Earliest start time]** Consider jobs in ascending order of $s_j$.
- **[Earliest finish time]** Consider jobs in ascending order of $f_j$.
- **[Shortest interval]** Consider jobs in ascending order of $f_j - s_j$.
- **[Fewest conflicts]** For each job $j$, count the number of conflicting jobs $c_j$. Schedule in ascending order of $c_j$. 
Greedy template. Consider jobs in some natural order. Take each job provided it's compatible with the ones already taken.
[Fewest conflicts] For each job $j$, count the number of conflicting jobs $c_j$. Schedule in ascending order of $c_j$.

Fewest conflicts fails to produce the optimal solution on which of the following inputs?

A. Inputs 1 and 2
B. Inputs 2 and 3
C. Succeeds on all inputs
D. Fails on all inputs
E. Input 1 only
Clicker Question

[Fewest conflicts] For each job $j$, count the number of conflicting jobs $c_j$. Schedule in ascending order of $c_j$.

Fewest conflicts fails to produce the optimal solution on which of the following inputs?

A. Inputs 1 and 2
B. Inputs 2 and 3
C. Succeeds on all inputs
D. Fails on all inputs
E. Input 1 only
Interval Scheduling: Greedy Algorithm

Greedy algorithm. Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

Sort jobs by finish times so that \( f_1 \leq f_2 \leq \ldots \leq f_n \).

A \leftarrow \emptyset

\textbf{for} j = 1 \textbf{ to } n \{ \\
\quad \textbf{if} (\text{job j compatible with A}) \\
\quad \quad A \leftarrow A \cup \{j\}
\}

\textbf{return} A

Implementation. \( O(n \log n) \).

- Remember job \( j^* \) that was added last to \( A \).
- Job \( j \) is compatible with \( A \) if \( s_j \geq f_{j^*} \).
Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots, i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots, j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of $r$. 

**Diagram:**

**Greedy:**

| $i_1$ | $i_2$ | ... | $i_r$ | $i_{r+1}$ | ... |

**OPT:**

| $j_1$ | $j_2$ | ... | $j_r$ | $j_{r+1}$ | ... |

- **why not replace job $j_{r+1}$ with job $i_{r+1}$?**
Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let’s see what happens.
- Let $i_1, i_2, \ldots, i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots, j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of $r$.

Graphically:

**Greedy:**

- $i_1$
- $i_2$
- $i_r$
- $i_{r+1}$

**OPT:**

- $j_1$
- $j_2$
- $j_r$
- $i_{r+1}$
- $\ldots$

Job $i_{r+1}$ finishes before $j_{r+1}$

Solution still feasible and optimal, but contradicts maximality of $r$. 
4.2 Scheduling to Minimize Lateness
Scheduling to Minimizing Lateness

Minimizing lateness problem.

- Single resource processes one job at a time.
- Job \( j \) requires \( t_j \) units of processing time and is due at time \( d_j \).
- If \( j \) starts at time \( s_j \), it finishes at time \( f_j = s_j + t_j \).
- Lateness: \( \ell_j = \max \{ 0, f_j - d_j \} \).
- Goal: schedule all jobs to minimize maximum lateness \( L = \max \ell_j \).

Ex:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_j )</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>( d_j )</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

- \( d_3 = 9 \), \( d_2 = 8 \), \( d_6 = 15 \), \( d_1 = 6 \), \( d_5 = 14 \), \( d_4 = 9 \)
- lateness = 2
- lateness = 0
- max lateness = 6
Minimizing Lateness: Greedy Algorithms

**Greedy template.** Consider jobs in some order.

- **[Shortest processing time first]** Consider jobs in ascending order of processing time $t_j$.

- **[Earliest deadline first]** Consider jobs in ascending order of deadline $d_j$.

- **[Smallest slack]** Consider jobs in ascending order of slack $d_j - t_j$. 
**Minimizing Lateness: Greedy Algorithms**

**Greedy template.** Consider jobs in some order.

- **[Shortest processing time first]** Consider jobs in ascending order of processing time $t_j$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$d_j$</td>
<td>100</td>
<td>10</td>
</tr>
</tbody>
</table>

  counterexample

- **[Smallest slack]** Consider jobs in ascending order of slack $d_j - t_j$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$d_j$</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

  counterexample
Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $t_j$.
- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

\[
\begin{array}{|c|c|}
\hline
j & 1 & 2 \\
\hline
\text{t}_j & 9 & 5 \\
\text{d}_j & 2 & 10 \\
\hline
\end{array}
\]

Which greedy algorithms outputs the optimal schedule (minimizes the maximum lateness)?

A. Smallest Slack
B. Shortest Processing Time First
C. Both
D. Neither
Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $t_j$.
- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>$d_j$</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Which greedy algorithms outputs the optimal schedule (minimizes the maximum lateness)?

A. Smallest Slack  
B. Shortest Processing Time First  
C. Both  
D. Neither