Week 6.1, Wed, Sept 25

Practice Midterm 1: Solutions Released
Midterm 1: September 25 (tonight)
Announcements

- No PSOs this week (due to Midterm)

- Yes, we do have class today 😊
  - Classes canceled on October 28th and Dec 6th
  - Make up for two evening midterm exams

- Homework 3 solutions released on Piazza
  - Submission server is closed for homework 3
Midterm 1: Logistics

- 90 minutes (8:00-9:30PM)
  - Tuesday/Thursday PSOs: SMTH 108  (Exam Capacity =115)
  - Friday PSO: MTHW 210  (Exam Capacity = 111)

- 1 Page of Handwritten Notes (Single-Sided)

- Standard paper (or A4) is acceptable

- Bring number 2 pencil (for scanned exam)

- Closed book, no calculators, no smartphones, no smartwatches, no laptops etc…
Practice Midterm Solutions
- Advice: Try to solve each problem yourself before checking answers

Topics:
- Induction
- Big-O
- Divide and Conquer
  - Sorting, Counting Inversions, Maximum Subarray, Skyline Problem, Karatsuba Multiplication
- Recurrences
  - Deriving a Recurrence
  - Unrolling
  - Recursion Trees
  - Master Theorem
- Greedy Algorithms

No Dynamic Programming Required (until Midterm 2)
6.1 Weighted Interval Scheduling
Weighted interval scheduling problem.

- Job $j$ starts at $s_j$, finishes at $f_j$, and has weight or value $v_j$.
- Two jobs compatible if they don't overlap.
- Goal: find maximum weight subset of mutually compatible jobs.

![Interval Scheduling Diagram](image-url)
Previously Showed: Greedy algorithm works if all weights are 1.

- Solution: Sort requests by finish time (ascending order)

Observation. Greedy algorithm can fail spectacularly if arbitrary weights are allowed.
**Weighted Interval Scheduling**

**Notation.** Label jobs by finishing time: $f_1 \leq f_2 \leq \ldots \leq f_n$.

**Def.** $p(j) =$ largest index $i < j$ such that job $i$ is compatible with $j$.

**Ex:** $p(8) = 5$, $p(7) = 3$, $p(2) = 0$. 
Dynamic Programming: Binary Choice

**Notation.** $OPT(j)$ = value of optimal solution to the problem consisting of job requests 1, 2, ..., $j$.

- **Case 1:** $OPT$ selects job $j$.
  - collect profit $v_j$
  - can't use incompatible jobs $\{ p(j) + 1, p(j) + 2, ..., j - 1 \}$
  - must include optimal solution to problem consisting of remaining compatible jobs $1, 2, ..., p(j)$

- **Case 2:** $OPT$ does not select job $j$.
  - must include optimal solution to problem consisting of remaining compatible jobs $1, 2, ..., j-1$

$$OPT(j) = \begin{cases} 
0 & \text{if } j = 0 \\
\max \{ v_j + OPT(p(j)), OPT(j-1) \} & \text{otherwise}
\end{cases}$$
Weighted Interval Scheduling: Brute Force

Brute force algorithm.

**Input:** \( n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n \)

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

Compute \( p(1), p(2), \ldots, p(n) \)

\[
\text{Compute-Opt}(j) \{
\quad \text{if } (j = 0) \\
\quad \quad \text{return } 0 \\
\quad \text{else} \\
\quad \quad \text{return } \max(v_j + \text{Compute-Opt}(p(j)), \text{Compute-Opt}(j-1))
\}
\]

\[
T(n) = T(n-1) + T(p(n)) + O(1)
\]

\[
T(1) = 1
\]

Reminder: \( p(n) \) = largest index \( i < n \) such that job \( i \) is compatible with job \( n \).
**Weighted Interval Scheduling: Brute Force**

*Observation.* Recursive algorithm fails spectacularly because of redundant sub-problems $\Rightarrow$ exponential algorithms.

*Ex.* Number of recursive calls for family of "layered" instances grows like Fibonacci sequence ($F_n > 1.6^n$).

\[
T(n) = T(n-1) + T(n-2) + 1
\]

\[
T(1) = 1
\]

\[p(1) = 0, \ p(j) = j-2\]

**Key Insight:** Do we really need to repeat this computation?
Weighted Interval Scheduling: Memoization

**Memoization.** Store results of each sub-problem in a cache; lookup as needed.

**Input:** \( n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n \)

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

Compute \( p(1), p(2), \ldots, p(n) \)

```
for j = 1 to n
    M[j] = empty
M[0] = 0  # global array

M-Compute-Opt(j) {
    if (M[j] is empty)
        M[j] = max(v_j + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
    return M[j]
}
```
Claim. Memoized version of algorithm takes $O(n \log n)$ time.
- Sort by finish time: $O(n \log n)$.
- Computing $p(.)$: $O(n \log n)$ via binary search.

- $M$-Compute-Opt$(j)$: each invocation takes $O(1)$ time and either
  - (i) returns an existing value $M[j]$
  - (ii) fills in one new entry $M[j]$ and makes two recursive calls

- Progress measure $\Phi = \# \text{ nonempty entries of } M[]$.
  - initially $\Phi = 0$, throughout $\Phi \leq n$.
  - (ii) increases $\Phi$ by 1 $\Rightarrow$ at most $2n$ recursive calls.

- Overall running time of $M$-Compute-Opt$(n)$ is $O(n)$.

Remark. $O(n)$ if jobs are pre-sorted by start and finish times.
Q. Dynamic programming algorithms computes optimal value. What if we want the solution itself?
A. Do some post-processing.

```
Run M-Compute-Opt(n)
Run Find-Solution(n)

Find-Solution(j) {
    if (j = 0)
        output nothing
    else if (v_j + M[p(j)] > M[j-1])
        print j
        Find-Solution(p(j))
    else
        Find-Solution(j-1)
}
```

- # of recursive calls ≤ n ⇒ O(n).
Bottom-up dynamic programming. Unwind recursion.

**Input:** $n, s_1, ..., s_n, f_1, ..., f_n, v_1, ..., v_n$

Sort jobs by finish times so that $f_1 \leq f_2 \leq ... \leq f_n$.

Compute $p(1), p(2), ..., p(n)$

Iterative-Compute-Opt {
    $M[0] = 0$
    for $j = 1$ to $n$
        $M[j] = \max(v_j + M[p(j)], M[j-1])$
}
Recall the weighted interval scheduling problem in which we are given a list of n meeting requests $J_1 = [s_1, f_1], \ldots, J_n = [s_n, f_n]$ with positive weights $w_1, \ldots, w_n > 0$ where meeting request $J_i$ starts at time $s_i$ and finishes at time $f_i > s_i$. We are given a single conference room and our goal is to find the maximum weight schedule with no conflicts (recall that requests $J_i$ and $J_k$ conflict if they overlap).

Part A: Does the greedy strategy (sort by weight) work?

![Diagram showing the scheduling of meetings with weights and time intervals.](image)
Recall the weighted interval scheduling problem in which we are given a list of $n$ meeting requests $J_1 = [s_1, f_1], \ldots, J_n = [s_n, f_n]$ with positive weights $w_1, \ldots, w_n > 0$ where meeting request $J_i$ starts at time $s_i$ and finishes at time $f_i > s_i$. We are given a single conference room and our goal is to find the maximum weight schedule with no conflicts (recall that requests $J_i$ and $J_k$ conflict if they overlap).

Part B: Suppose $f_1 < f_2 < \ldots < f_n$ and $w_1 > \ldots > w_n$?

**Greedy:**
- $i_1$, $i_2$, $\ldots$, $i_r$, $i_{r+1}$, $\ldots$

**OPT:**
- $j_1$, $j_2$, $\ldots$, $j_r$, $j_{r+1}$, $\ldots$

Why not replace job $j_{r+1}$ with job $i_{r+1}$?

Job $i_{r+1}$ finishes before $j_{r+1}$ and therefore has higher weight.
T(1), T(2), T(3), T(4) < 11 and T(n) = 7T(n/2)+T(n/5) + n^3

Use induction to prove T(n) <= 10n^3

Bases Cases? Trivially true for n< 5.

Inductive Hypothesis: T(k) <= 10k^3 whenever k < n

Inductive Step:

T(n) = 7T(n/2)+T(n/5) + n^3 <= 10*7*(n^3 /8)+ 10*n^3/125 + n^3 (IH)
<= 10n^3 (by algebra)