CS 381 – FALL 2019

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…

Midterm 1

- 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

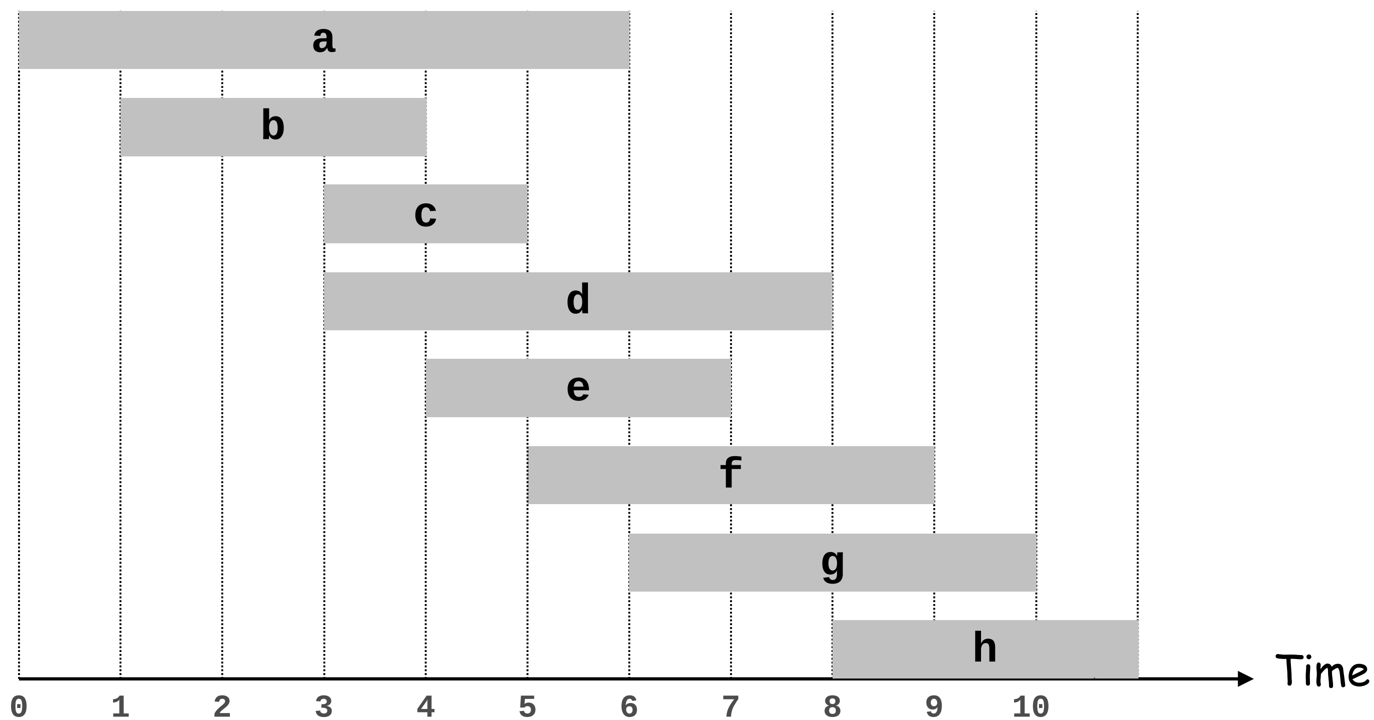
- No Dynamic Programming Required (until Midterm 2)

6.1 Weighted Interval Scheduling

Weighted Interval Scheduling

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.

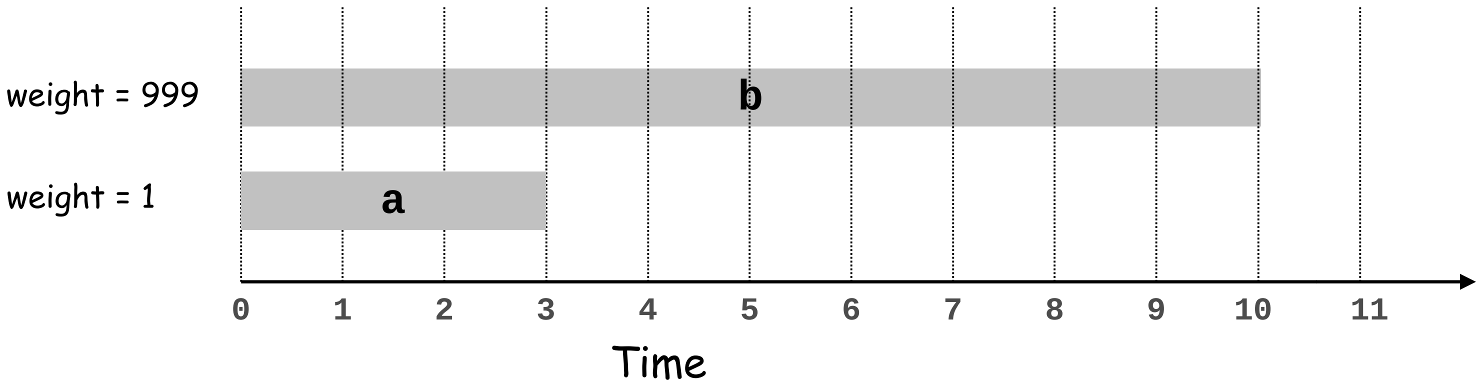


Unweighted Interval Scheduling (will cover in Greedy paradigms)

Previously Showed: Greedy algorithm works if all weights are 1.

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

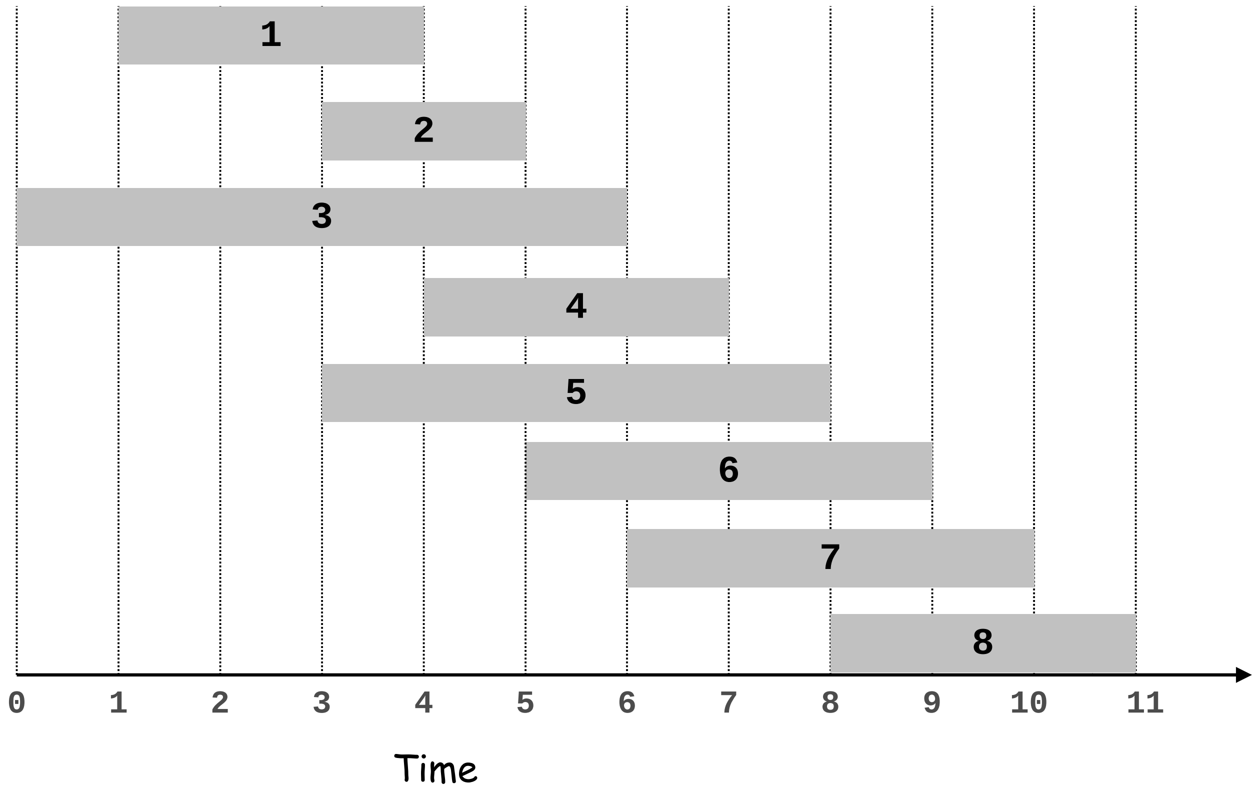
Observation. Greedy algorithm can fail spectacularly if arbitrary weights are allowed.



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)

optimal substructure

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

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max \left\{ v_j + OPT(p(j)),\ OPT(j-1) \right\} & \text{otherwise} \end{cases}$$

Weighted Interval Scheduling: Brute Force

Brute force algorithm.

```
Input: n, s1,…,sn , f1,…,fn , v1,…,vn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

Compute p(1), p(2), …, p(n)

Compute-Opt(j) {
   if (j = 0)
      return 0
   else
      return max(vj + Compute-Opt(p(j)), Compute-Opt(j-1))
}
```

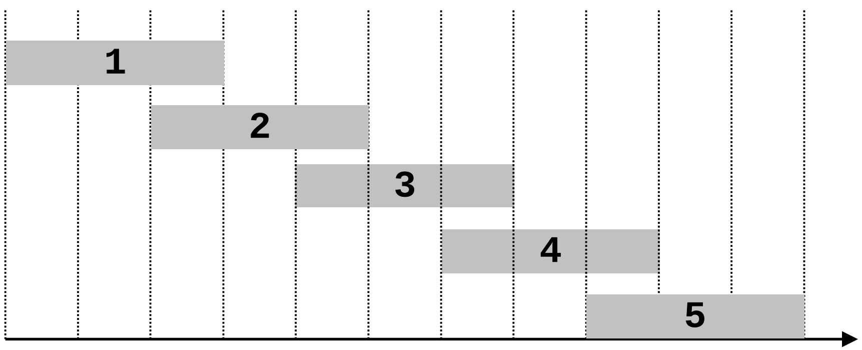
$$T(n) = T(n-1)+T(p(n))+O(1)$$
$$T(1) = 1$$

Reminder: $p(n)$ = largest index $i < n$ such that job i is compatible with job n.

Weighted Interval Scheduling: Brute Force

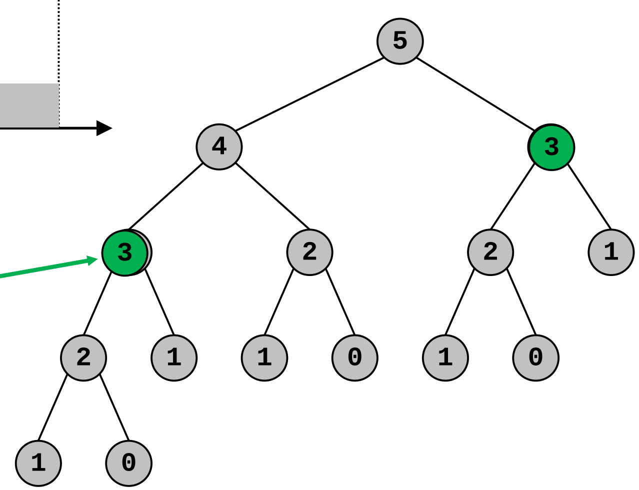
Observation. Recursive algorithm fails spectacularly because of redundant sub-problems $\Rightarrow$ exponential algorithms.

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



p(1) = 0, p(j) = j-2

$T(n) = T(n-1)+T(n-2)+1$
$T(1) = 1$



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, s1,…,sn , f1,…,fn , v1,…,vn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.
Compute p(1), p(2), …, 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(vj + M-Compute-Opt(p(j)), M-Compute-Opt(j-1))
   return M[j]
}
```

Weighted Interval Scheduling: Running Time

Claim. Memoized version of algorithm takes $O(n \log n)$ time.

- Sort by finish time: $O(n \log n)$.
- Computing $p(\cdot)$: $O(n \log n)$ 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 Φ = # nonempty entries of `M[]`.
  - initially $\Phi = 0$, throughout $\Phi \leq n$.
  - (ii) increases Φ 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.

Weighted Interval Scheduling: Finding a Solution

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 $\leq n \Rightarrow O(n)$.

Weighted Interval Scheduling: Bottom-Up

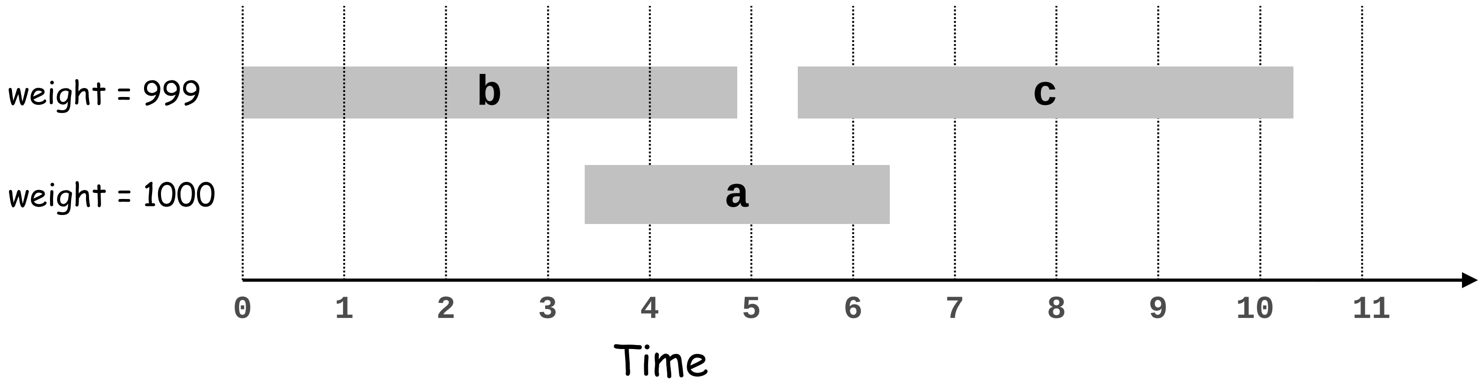
Bottom-up dynamic programming. Unwind recursion.

```
Input: n, s_1,…,s_n , f_1,…,f_n , v_1,…,v_n

Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ 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])
}
```

Practice Midterm

Recall the weighted interval scheduling problem in which we are given a list of n meeting requests J1 = [s1, f1], . . . , Jn = [sn, fn] with positive weights w1, . . . , wn > 0 where meeting request Ji starts at time si and finishes at time fi > si . We are given a single conference room and our goal is to find the maximum weight schedule with no conflicts (recall that requests Ji and Jk conflict if they overlap).

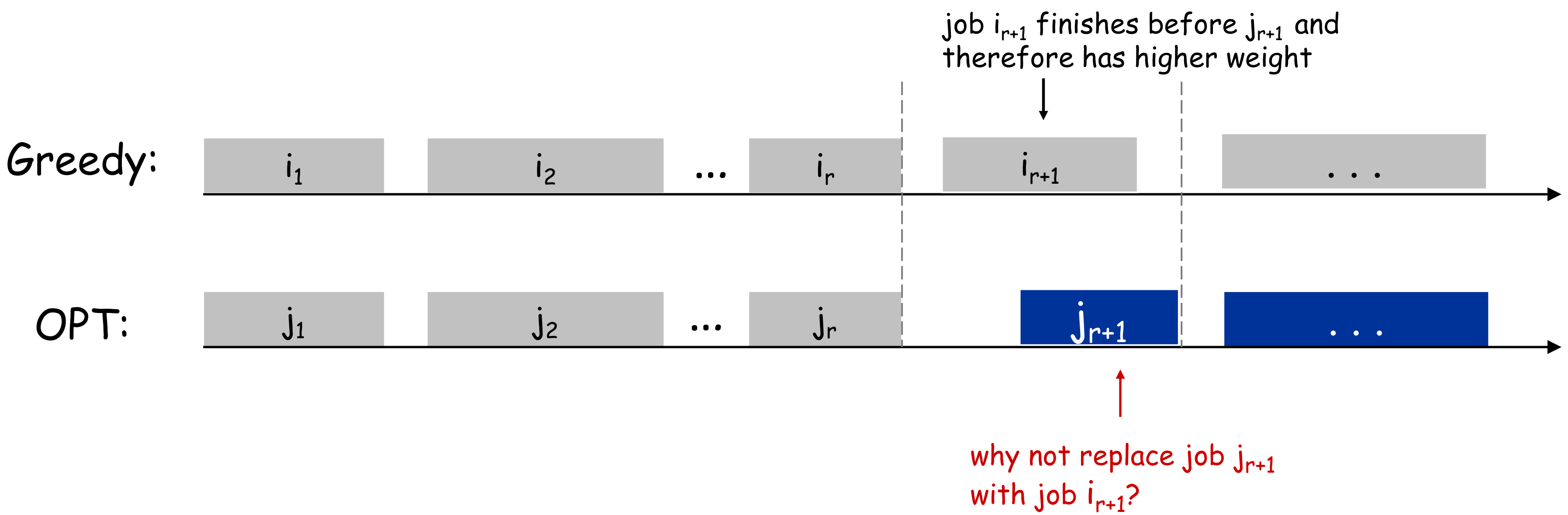
Part A: Does the greedy strategy (sort by weight) work?



Practice Midterm

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 $w_1 > \ldots > w_n$?



Practice Midterm

$T(1), T(2), T(3), T(4) < 11$ 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)