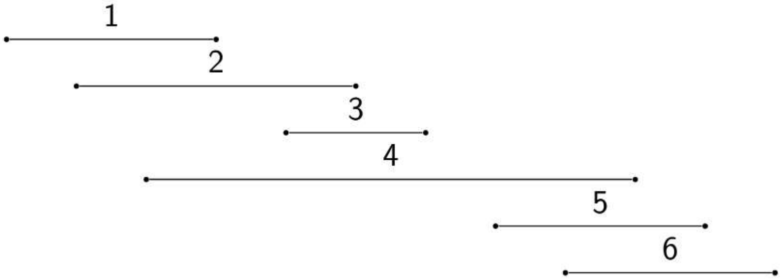
Computer Science
Spring 2021 Lecture 13:
Dynamic Programming:
Interval Scheduling

Warm-Up

- Given n intervals, $1 \ldots n$,
  - each has start time s_i and finish time f_i.
- For each interval, compute a value $p[i]$
  - $p[i] = j$ means j is the *latest* f_j such that $f_j \leq s_i$
  - If no intervals end before s_i, then $f[i] = 0$.
- Intervals are already sorted by finish time.

Example:



3

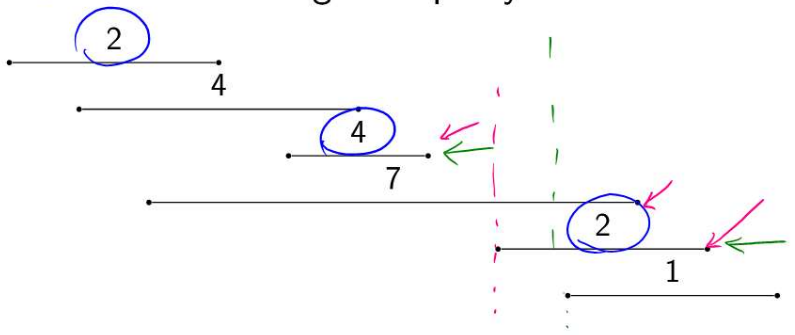
Warm-Up

```
Warm-up(int $n$, intervals $[s_1, f_1]$, $[s_2, f_2]$, $\dots$ $[s_n, f_n]$)
  Sort intervals by finish time (if not already)
  for each interval i   // $[S_i, f_i]$
      // find largest $f_j \le S_i$ in $O(\log n)$
      $p[i]$ = binary search for that
```

4

Interval Scheduling Problem Statement

- Which classes should take next quarter?
- The classes all meet once a day,
  - at different times and lengths
  - are worth different amounts of credits.
- Maximize amount of credits earned in quarter
- Without having to skip any classes



5

Interval Scheduling: Recursive Solution

- ▶ Key : your friend will take class i xor won't

```
OPT(i) // opt # of credits, intervals 1...i
   // Base Case:
        if  i< 1    return 0
   // If my friend doesn't take class i:
   value_if_not_taken = OPT (i-1)

   // If my friend takes class i:
   value_if_taken = v[i] + OPT (p[i])

   //return something:
   return   max(value_if_not_taken,
                          value_if_taken)
```

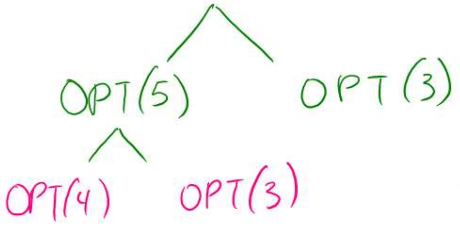
6

Interval Scheduling: Recursive Implementation

```
OPT(i)
   if i is 0 then
      return 0
   // value_if_not_taken = OPT(i − 1)
   // value_if_taken = $v_i$ + OPT(p[i])
   return  max( OPT(i − 1), $v_i$ + OPT(p[i]) )
```

recursive substructure

- ▶ To solve: call OPT(6) for this input.



Overlapping subproblems

7

Interval Scheduling: Memoization

Declare Memo[0...n], ∀i Memo[i] = -1
memo[0] = 0

- Many overlapping subproblems in rec solution.

```
OPT(i)
  ~~if i is 0 then~~
    ~~return 0~~
  if (-1 == memo[i])



    memo[i]: = max(OPT($\underline{i-1}$), $v_i$ + OPT($\underline{p[i]}$))
return memo[i] // relies on smaller values
```

8

Interval Scheduling: Iterative Solution

- Observation: once Memo[0 . . . $i-1$] filled in,

  I can fill in Memo[i]

- We can write an iterative solution.

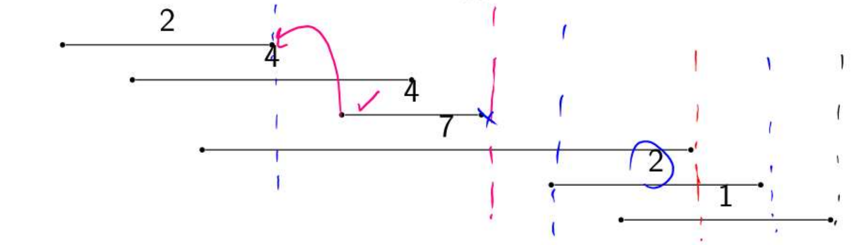
```
Declare Memo[0...n] // θ(n)
Compute p[i] values // θ(n lg n)
Memo[0] = 0
for i=1 to n
    Memo[i] = max(memo[i-1],
                  memo[p[i]] + v[i])
*
```

9

Interval Scheduling: Table

memo array

i	$p[i]$	OPT(p(i)) + v_i	OPT($i-1$)	OPT(i)
0	N/A	N/A	N/A	0
1	0	0 +2	0	2
2	0	0 +4	2	4
3	1	2 +4	4	6
4	0	0 +7	6	7
5	3	6 +2	7	8
6	3	6 +1	8	8



10

What classes to take?

- Now we have `Memo[...]` filled in.
- Instead of `return Memo[n]`, output courses.
- Hint: take course *n* or no? ← our tautology <3

```
i ← n
while i > 0:
    //do I take course i?
    if memo[i] > memo[i-1]:
        Output i^th
        i ← p(i)
    else:
        i ← i-1
```