CS 5112

Dynamic Programming

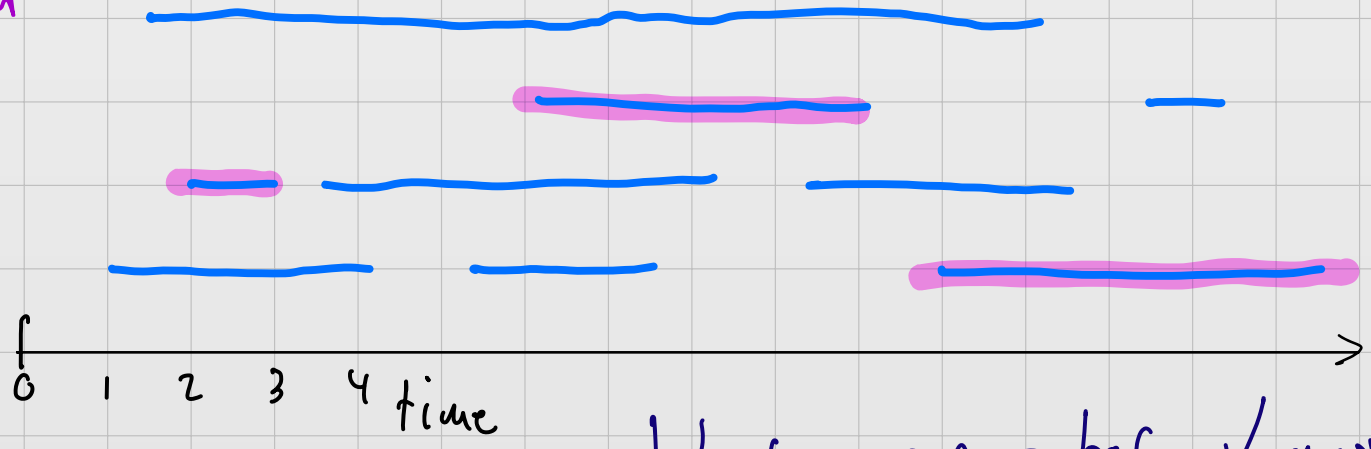
Weighted Interval Scheduling

Interval Scheduling

Recall

Input: Jobs $J_1, J_2, \ldots, J_n$ $\quad J_i = (s_i, f_i)$

A subset of jobs are compatible if no two overlap



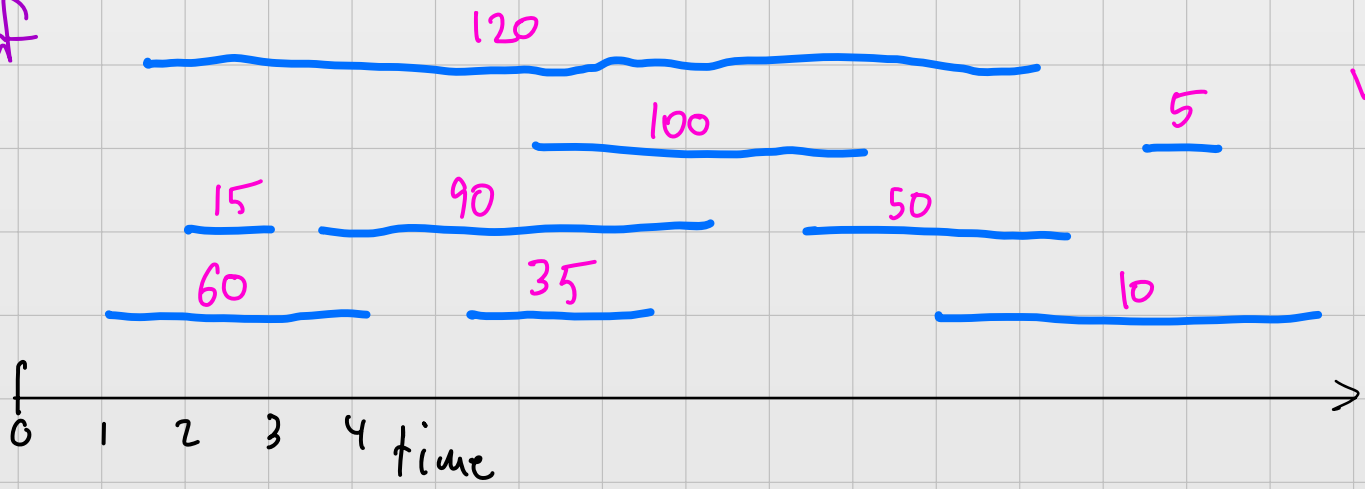
Want a comp. subset w/ most # of jobs

Weighted Interval Scheduling

Input: Jobs $J_1, J_2, \ldots, J_n$ $J_i = (s_i, f_i, v_i)$

v_i = value/weight

A subset of jobs are compatible if no two overlap



Output: The compatible subset with greatest value

Motivating Example

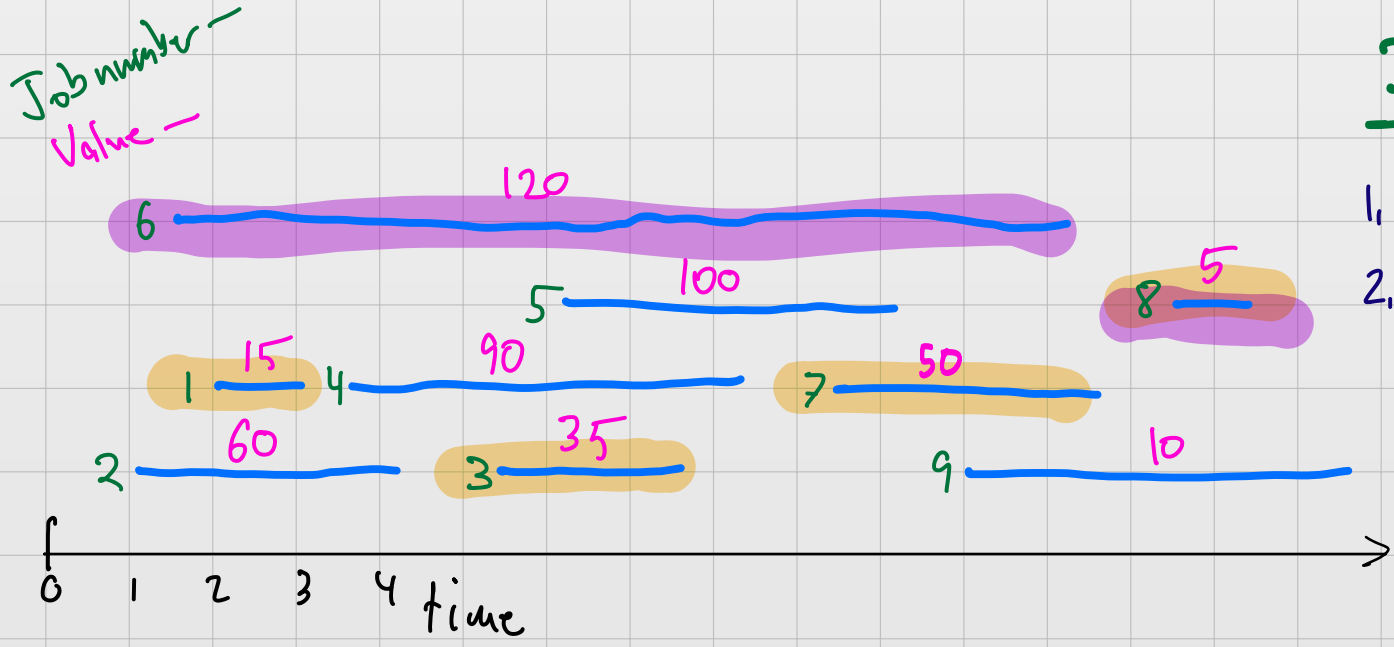
You live in NYC. NYC is expensive.
You decide to do odd jobs to make money.

Job 1:	Walk neighbor's dog:	Tue 9am - 9:30am	$20
Job 2:	Help with move:	Wed 10am - 6pm	$300
Job 3:	Housesit:	Tue 9am - Th 8pm	$100
Job 4:	IT desk:	Tue 5pm - 8pm	$120
⋮	⋮	⋮	⋮

How to maximize the money you can make?

Weighted Interval Scheduling

Assume jobs are ordered by EFT. ($f_1 \leq f_2 \leq f_3 \leq \cdots$)

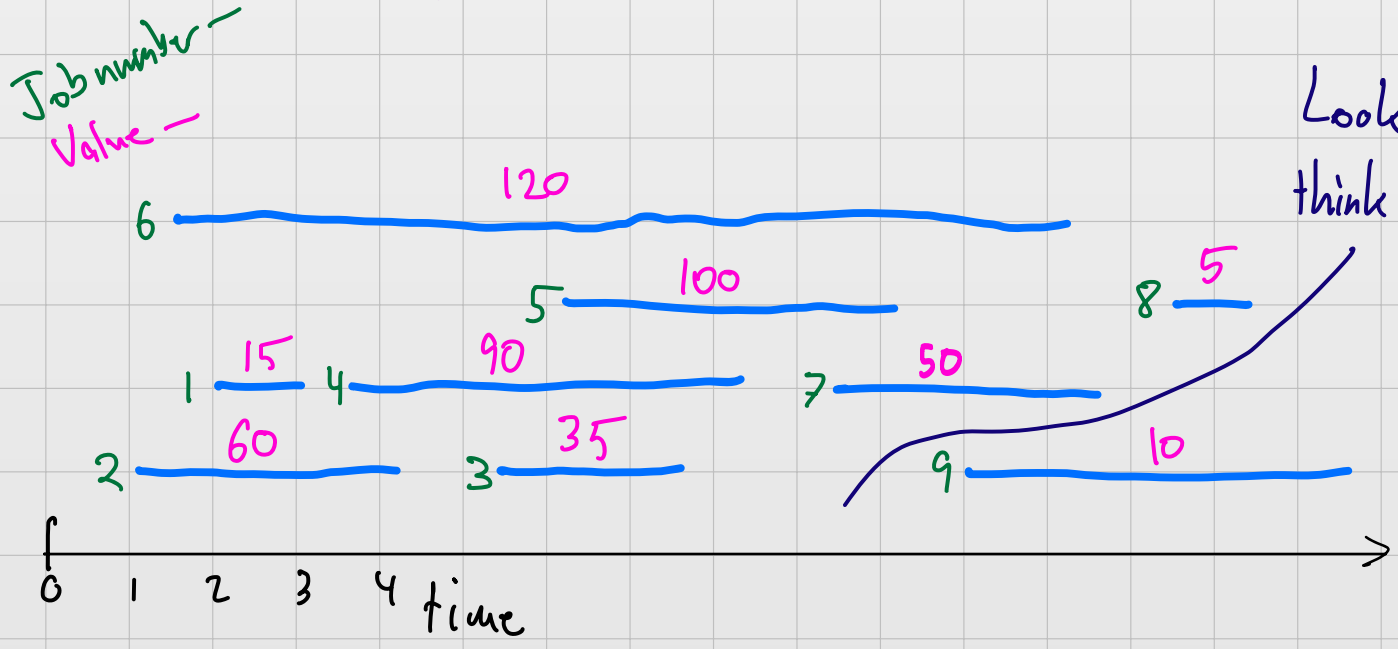


Ideas

1. Brute force - $O(2^n)$
2. Use greedy from IS
   - wrong answer
3. Other greedy
   - wrong answer

Weighted Interval Scheduling

Assume jobs are ordered by EFT. $(f_1 \leq f_2 \leq f_3 \leq \cdots)$



Look at J_9 and think about Opt

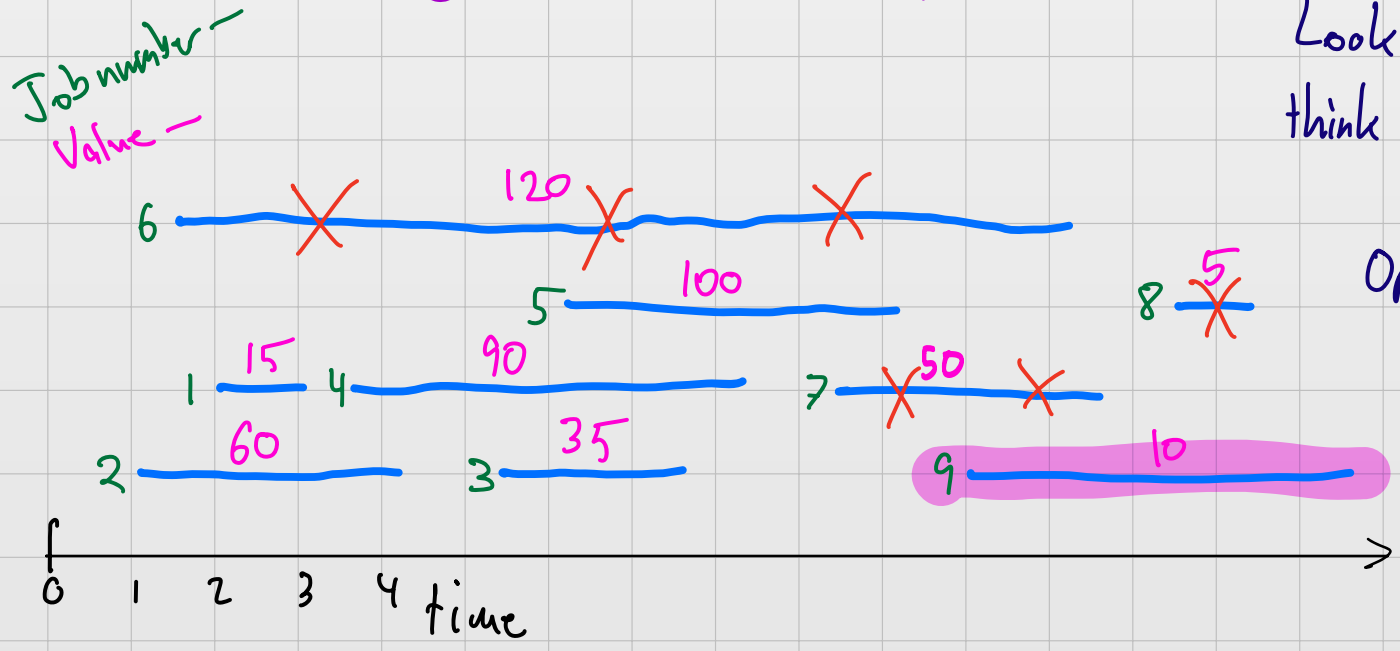
Case 1: $J_9 \in Opt$

Case 2: $J_9 \notin Opt$

Opt is also an optimal solution for $\{J_1, \ldots, J_8\}$

Weighted Interval Scheduling

Assume jobs are ordered by EFT. ($f_1 \le f_2 \le f_3 \le \cdots$)



Look at J_9 and think about Opt

Case 1: $J_9 \in$ Opt

Opt is an opt. sol. on $\{J_1, \ldots, J_8\}$ + J_9

Case 2: $J_9 \notin$ Opt

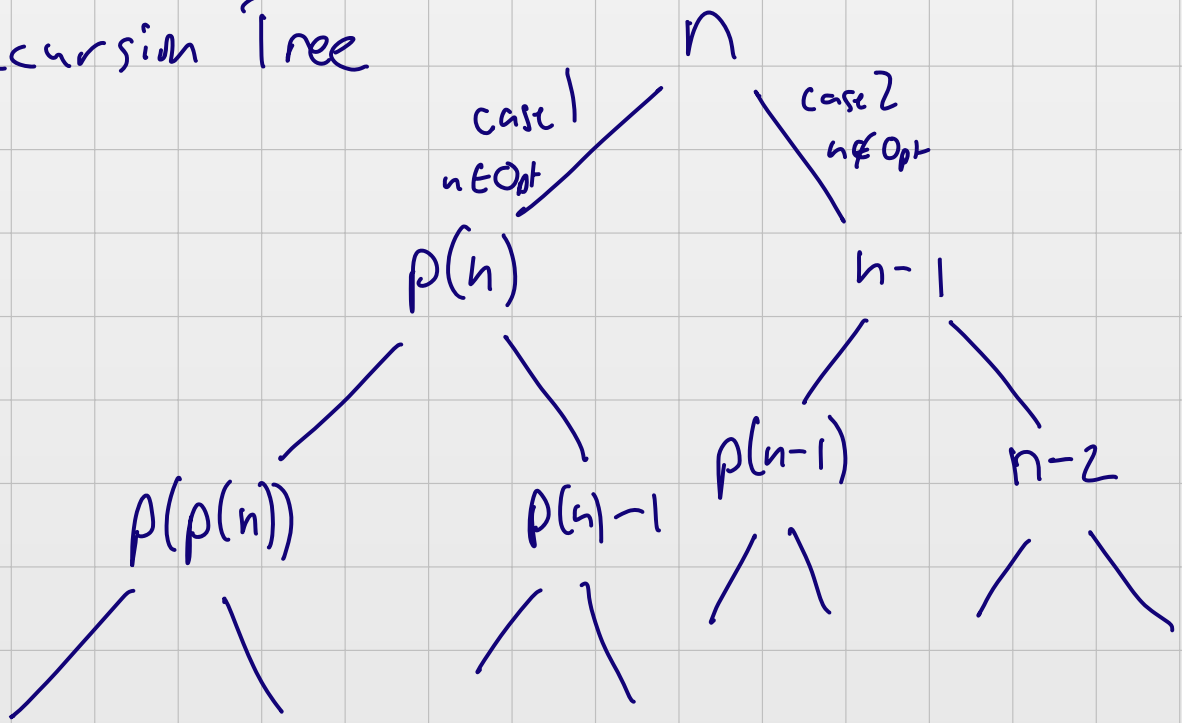
Opt is also an optimal solution for $\{J_1, \ldots, J_8\}$

Weighted Interval Scheduling

```
Opt-val(n):        // optival subset for J_1, ..., J_n
    val1 = Value(Opt-val(p(n)) + Value(J_n)
    val2 = Value(Opt-val(n-1))
    if(val1 > val2)
        return Opt-val(p(n)) ∪ {J_n}
    else
        return Opt-val(n-1)
```

Defn
p(n) to be the largest index $i < n$ s.t. J_i doesn't overlap with J_n

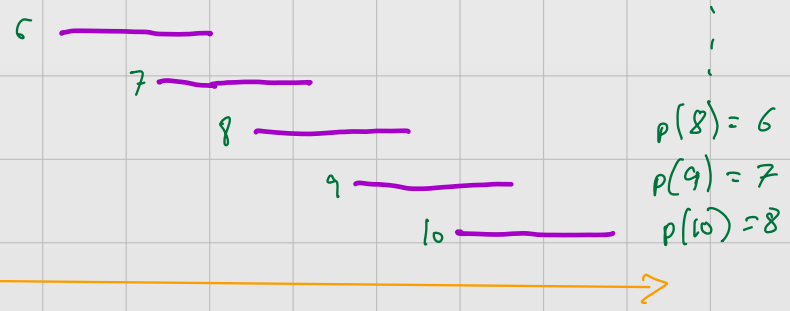
Recursion Tree



If we go down $n/2$ times down the tree, that level will be full.

$\Rightarrow 2^{n/2}$ recursive calls at that level

Still exponential time!

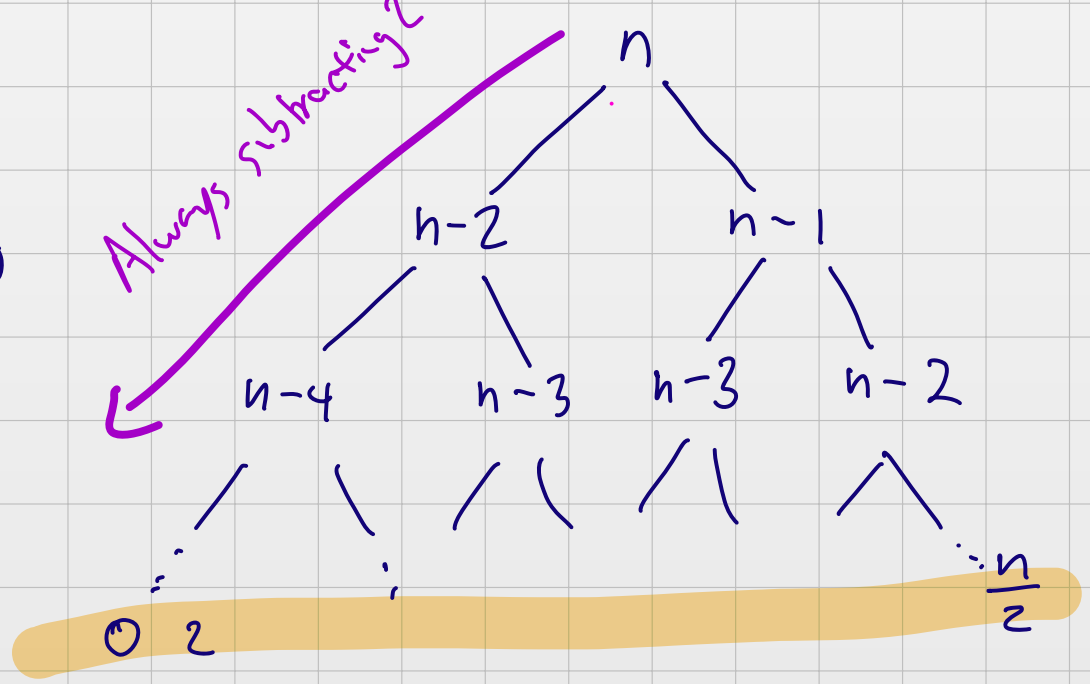


$p(n) = n-2$

$p(8) = 6$

$p(9) = 7$

$p(10) = 8$



Observation: We're duplicating a lot of computation!

Memoization

Idea: Remember the results of prior recursive calls.

Initialize an array $M[1, \ldots, n]$ $M[i] = -1$ $1 \le i \le n$

-1 ⇑ not yet computed

```
M-Opt-Val(n):
    if n = 0 : return 0
    if M[n] ≠ -1 : return M[n]      // new part : memoization
    val1 = ...
    val2 = ...
    if val1 > val2 ....
    M[n] = better of cases          // new part: store the answer
```

Only do once for each n

$\Rightarrow n$ recursive calls

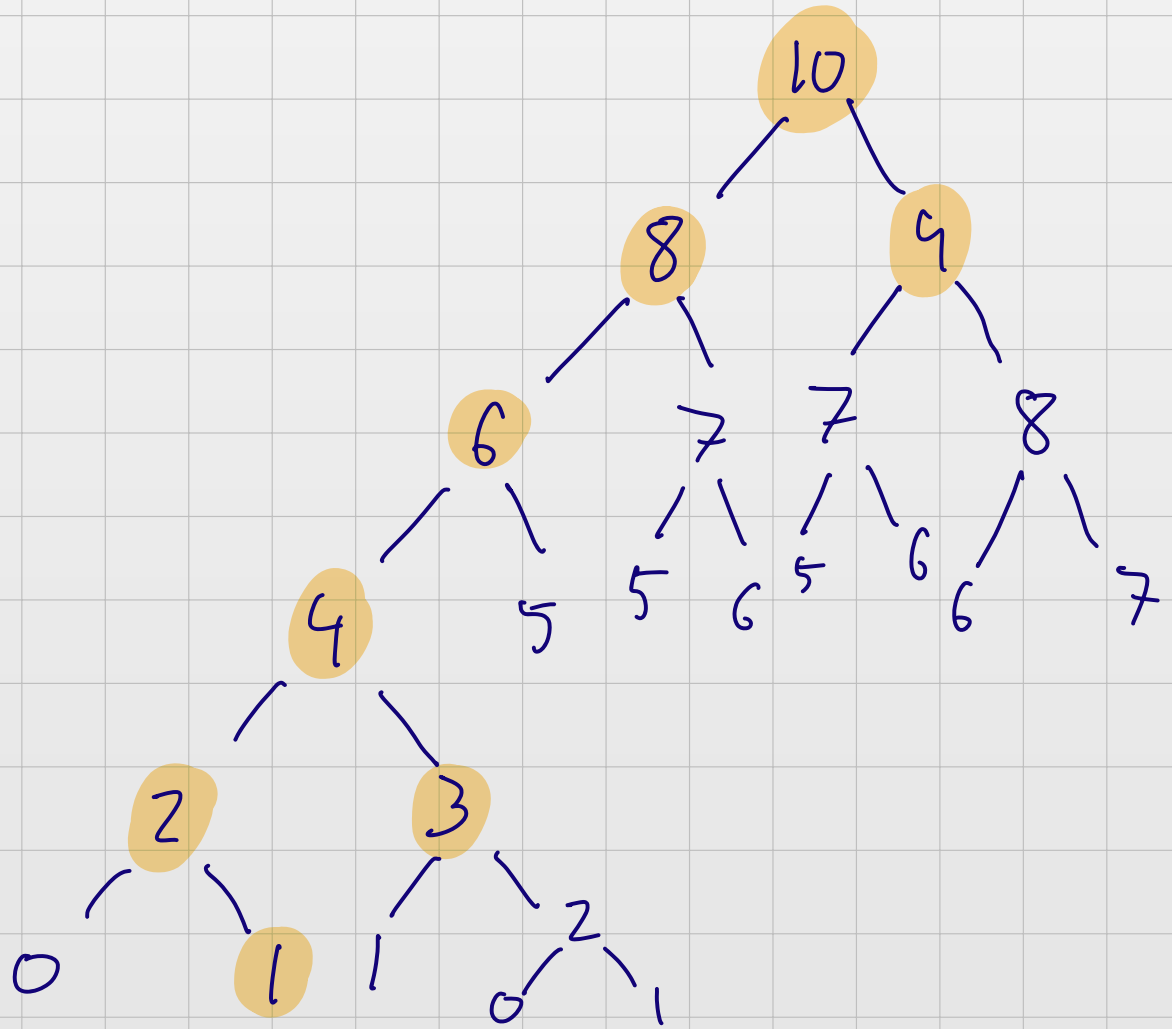
Weighted Interval Scheduling

```
Opt-val(n):        // optival subset for J_1 .., J_n
    if n = 0 : return 0
    if M[n] ≠ -1 : return M[n]
    val1 = Value(Opt-val(p(n))) + Value(J_n)
    val2 = Value(Opt-val(n-1))
    if (val1 > val2)
        M[n] = Opt-val(p(n)) ∪ {J_n}
    else
        M[n] = Opt-val(n-1)
    return M[n]
```

Memoization

Do O(n) calls, each one does O(1) work

=> O(n)

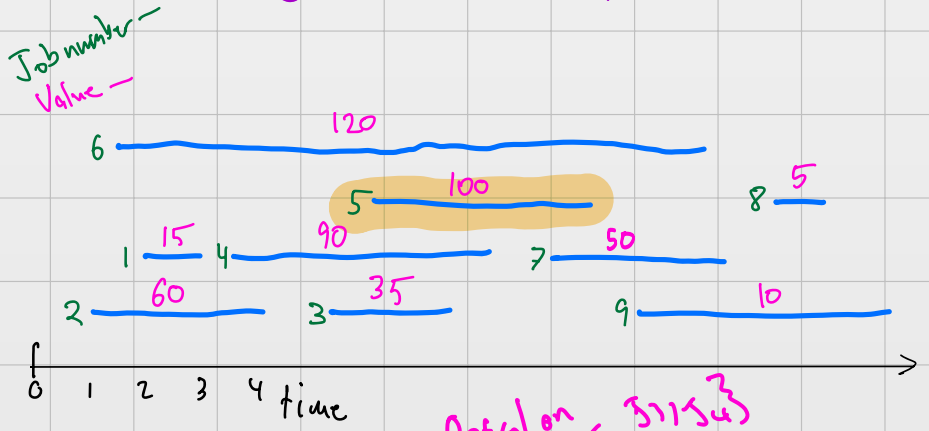


M :

1 ✓
2 ✓
3 ✓
4 ✓
5
6 ✓
7
8 ✓
9
10 ✓

Weighted Interval Scheduling

Assume jobs are ordered by EFT. $(f_1 \le f_2 \le f_3 \le \cdots)$



Opt-val on $\{3, 5\}$ $\{3, 4, 5\}$ $\{1, 3, 4, 5\}$

M: 0 1 2 3 4 5 6 7 8 9

Opt-val(9)

val1

Opt-val(5)

val1 val2

Opt-val(2) = 60 Opt-val(4) Opt-val(3)

Opt-val = 15 + 90 = 105

val1 val2

val1 = 0 + 60 Opt-val(1) = 15

val1 = 0 val2 = 0 + 15