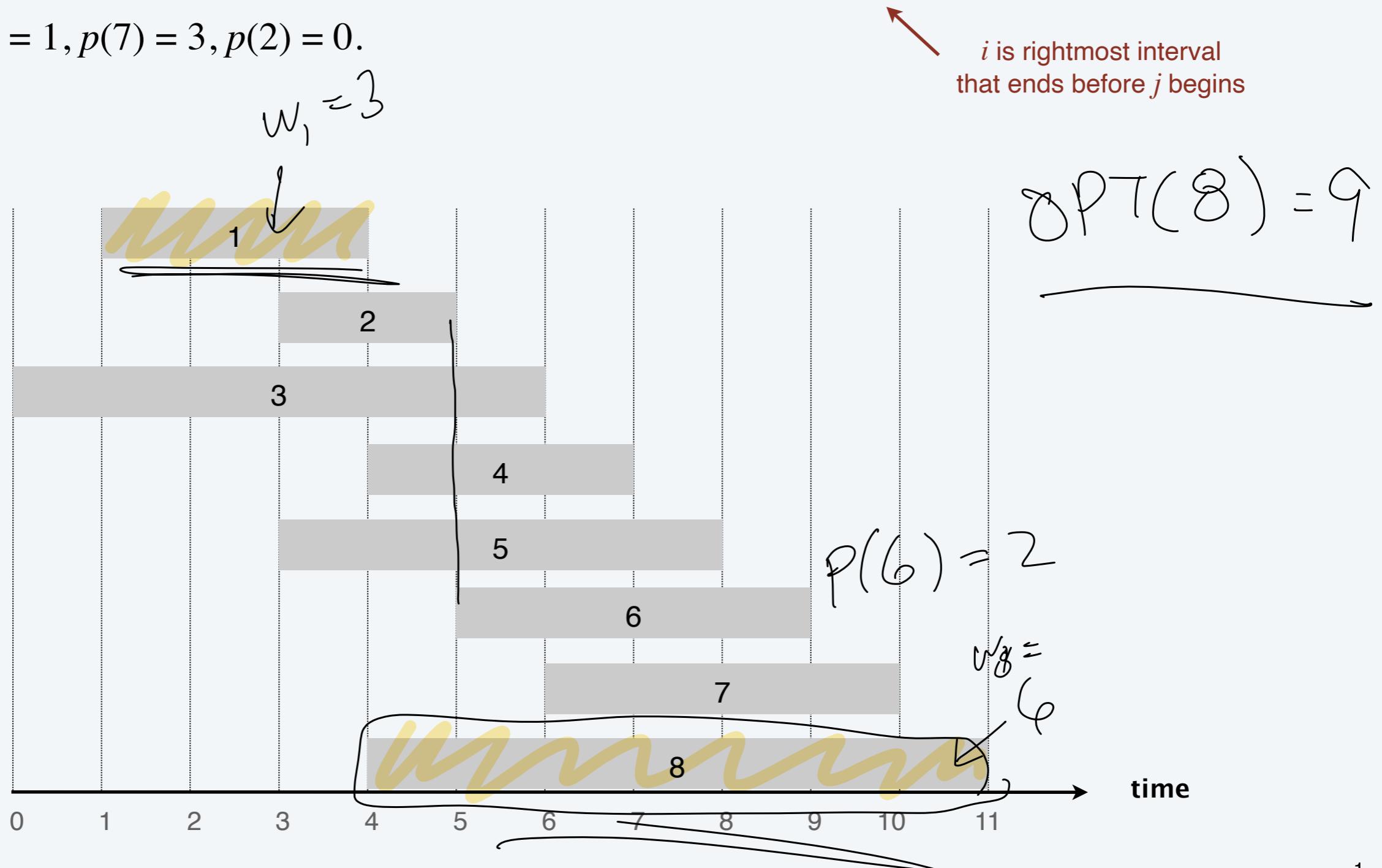


Weighted interval scheduling

Convention. Jobs are in ascending order of finish time: $f_1 \leq f_2 \leq \dots \leq f_n$.

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

Ex. $p(8) = 1, p(7) = 3, p(2) = 0$.



Dynamic programming: binary choice

Def. $OPT(j)$ = max weight of any subset of mutually compatible jobs for subproblem consisting only of jobs $1, 2, \dots, j$.

Goal. $OPT(n)$ = max weight of any subset of mutually compatible jobs.

Case 1. $OPT(j)$ does not select job j .

- Must be an optimal solution to problem consisting of remaining jobs $1, 2, \dots, j - 1$.

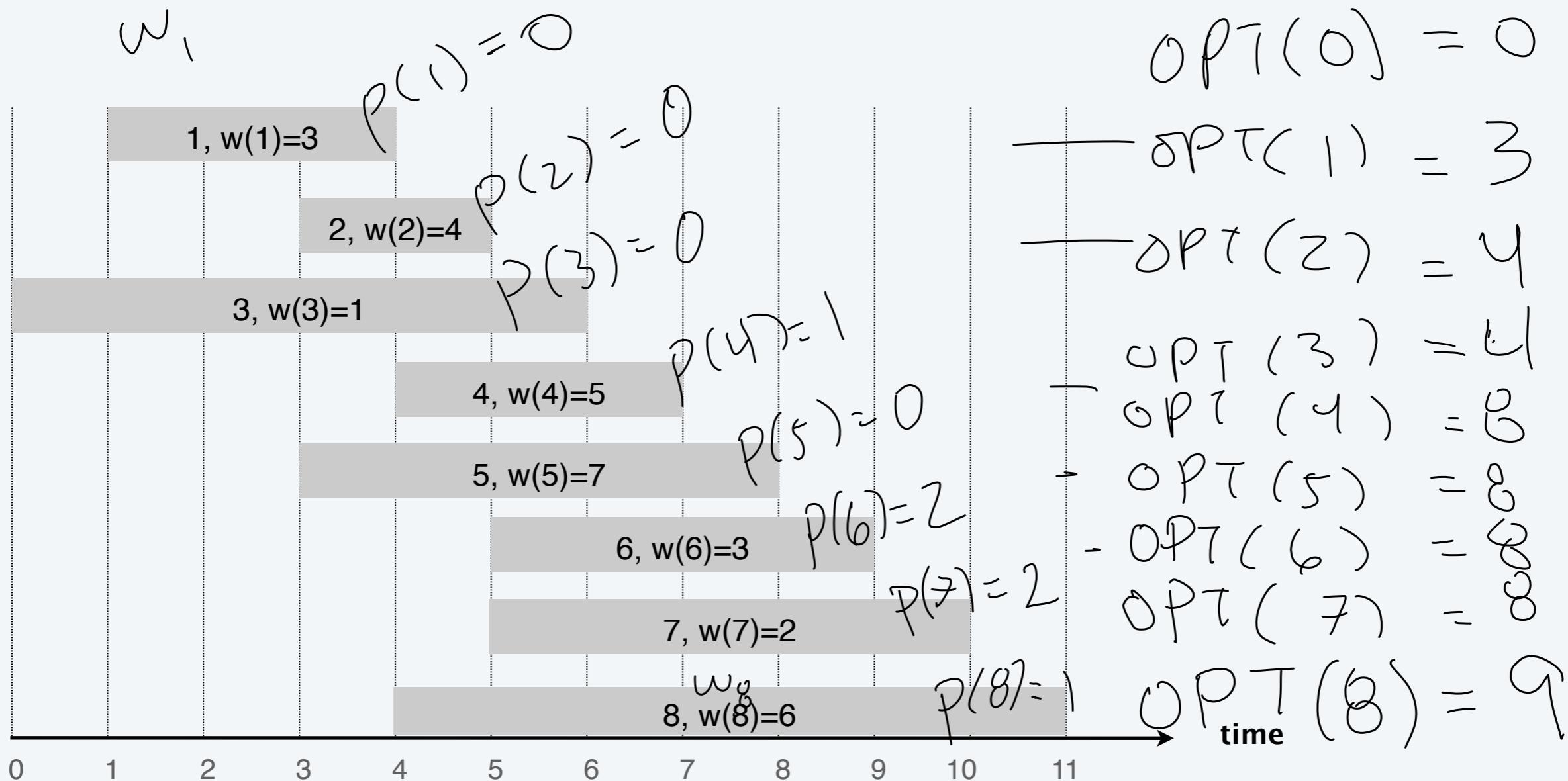
$$OPT(j - 1)$$

Case 2. $OPT(j)$ selects job j .

- Collect profit w_j .
- Can't use incompatible jobs $\{ p(j) + 1, p(j) + 2, \dots, j - 1 \}$.
- Must include optimal solution to problem consisting of remaining compatible jobs $1, 2, \dots, p(j)$.

$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max \{ OPT(j - 1), w_j + OPT(p(j)) \} & \text{if } j > 0 \end{cases}$$

Weighted interval scheduling: what is max weight subset?



$OPT(j)$ = optimal weight using jobs 1 to j

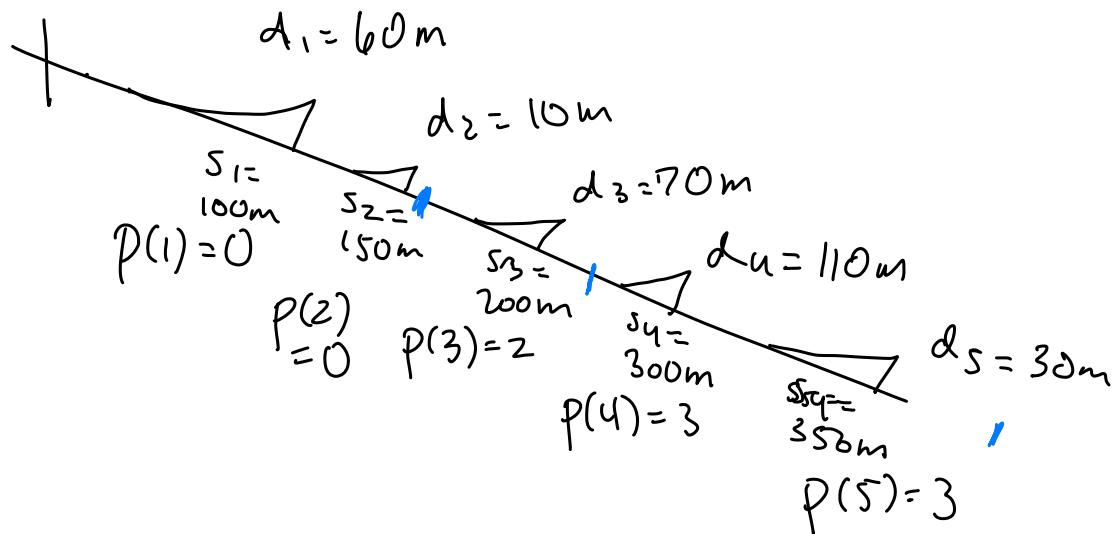
$$OPT(j) = \begin{cases} 0 & j = 0 \\ \max \{ OPT(j-1), w_j + OPT(p(j)) \} & j > 0 \end{cases}$$

$$OPT(8) = \max \{ OPT(7), 6 + OPT(1) \}$$

Activity



an example



$$\text{OPT}(j) = \begin{cases} 0 & j = 0 \\ \max(\text{OPT}(j-1), d_j + \text{OPT}(p(j))) & j > 0 \end{cases}$$

Weighted interval scheduling

SCHEDULE ($n, s_1, \dots, s_n, f_1, \dots, f_n, w_1, \dots, w_n$)

Sort jobs by finish time and renumber so that $f_1 \leq f_2 \leq \dots \leq f_n$.

Compute $p[1], p[2], \dots, p[n]$ via binary search.

RETURN COMPUTE-OPT(n).

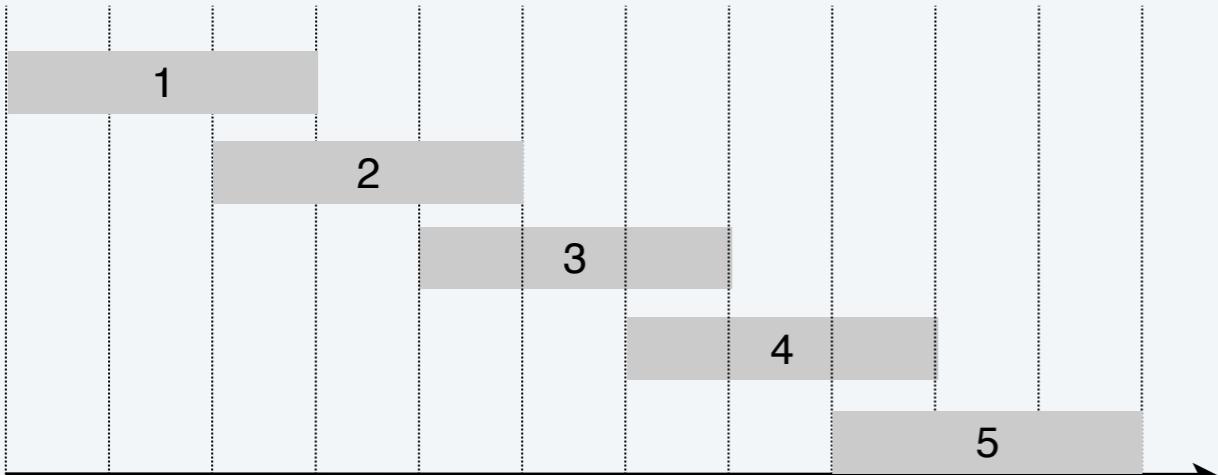
COMPUTE-OPT(j)

IF ($j = 0$)

 RETURN 0.

ELSE

 RETURN max {COMPUTE-OPT($j-1$), $w_j + \text{COMPUTE-OPT}(p[j])$ }.



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

With your table. Hint: draw the recursion tree!

What is running time of COMPUTE-OPT(n) in the worst case?

- A. $\Theta(n \log n)$
- B. $\Theta(n^2)$
- C. $\Theta(1.618^n)$
- D. $\Theta(2^n)$

COMPUTE-OPT(j)

IF ($j = 0$)

RETURN 0.

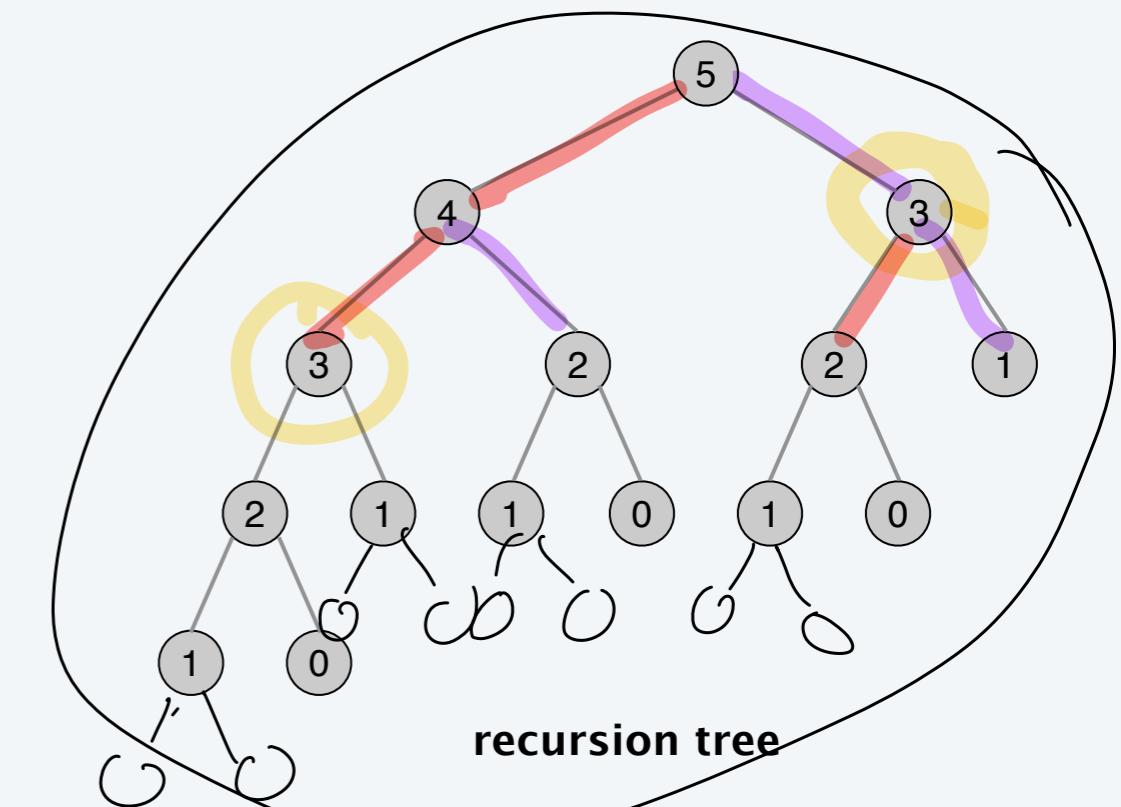
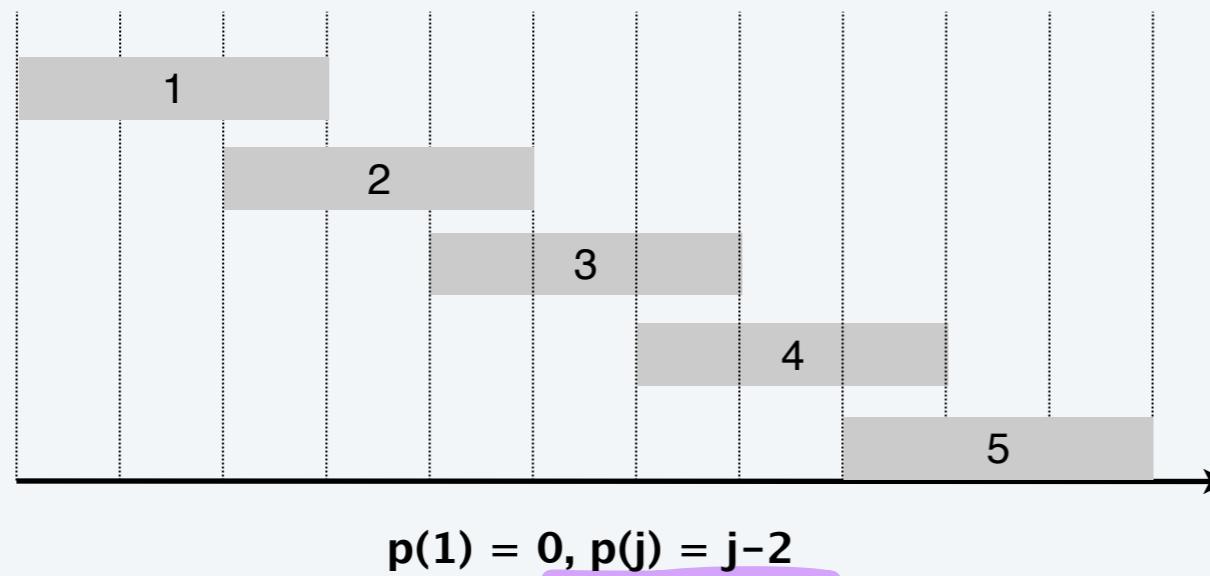
ELSE

RETURN max {COMPUTE-OPT($j - 1$), $w_j + \text{COMPUTE-OPT}(p[j])$ }.

Weighted interval scheduling: brute force

Observation. Recursive algorithm is spectacularly slow because of overlapping subproblems \Rightarrow exponential-time algorithm.

$$\text{opt}(j) = \begin{cases} \max(\text{opt}(j-1), w_j + \text{opt}(p(j))) & j > 0 \\ 0 & j = 0 \end{cases}$$



$\text{opt}(5)$

$\text{opt}(4)$

$\text{opt}(3)$

how many recursive calls? $\sim 2^n$

Weighted interval scheduling: memoization

Top-down dynamic programming (memoization).

- Cache result of subproblem j in $M[j]$.
- Use $M[j]$ to avoid solving subproblem j more than once.

TOP-DOWN($n, s_1, \dots, s_n, f_1, \dots, f_n, w_1, \dots, w_n$)

Sort jobs by finish time and renumber so that $f_1 \leq f_2 \leq \dots \leq f_n$.

Compute $p[1], p[2], \dots, p[n]$ via binary search.

$M[0] \leftarrow 0$. global array

RETURN M-COMPUTE-OPT(n).

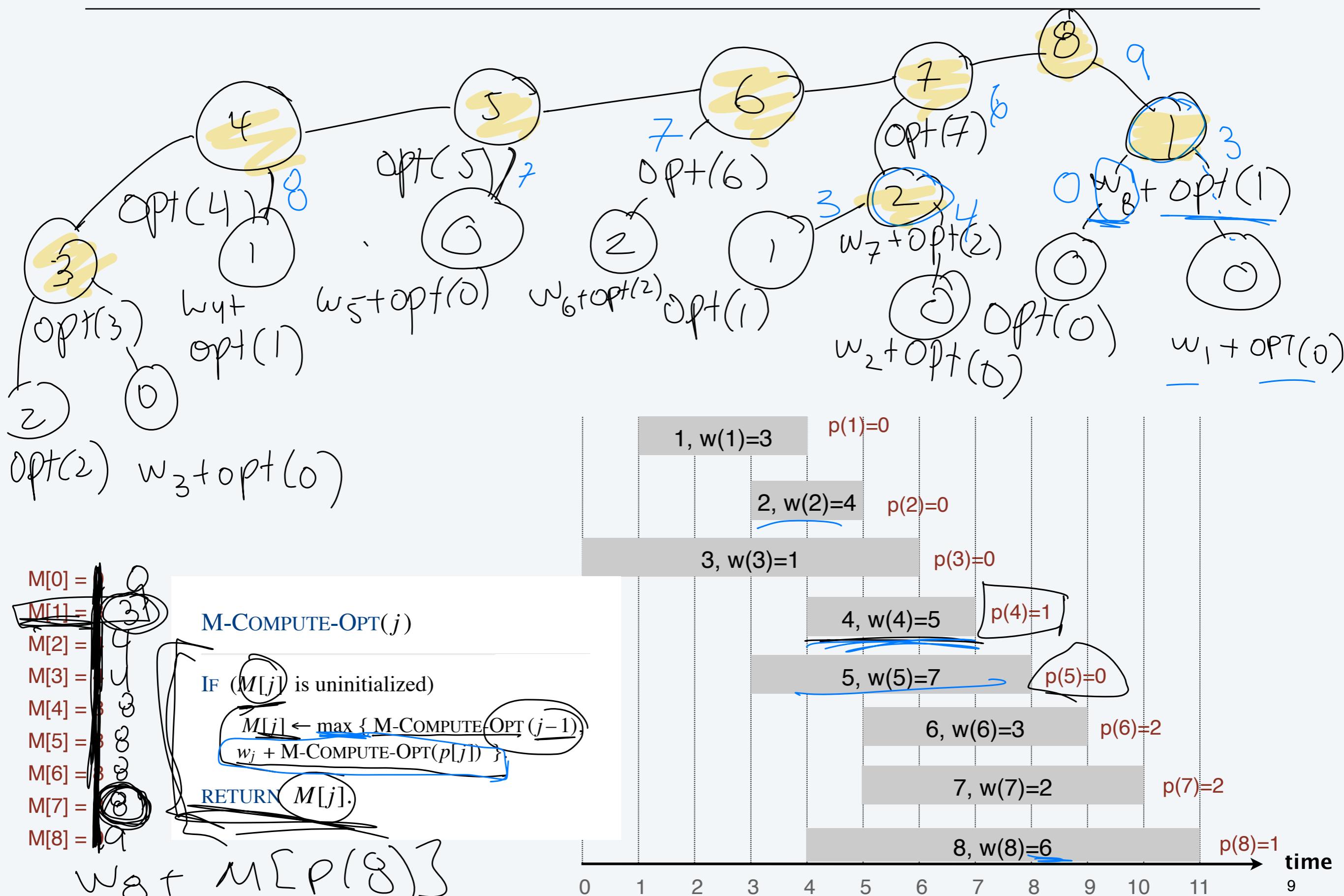
M-COMPUTE-OPT(j)

IF ($M[j]$ is uninitialized)

$M[j] \leftarrow \max \{ M\text{-COMPUTE-OPT}(j-1), w_j + M\text{-COMPUTE-OPT}(p[j]) \}$.

RETURN $M[j]$.

Trace through memoized version of Compute-OPT



Weighted interval scheduling: running time

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

Pf.

- Sort by finish time: $O(n \log n)$ via mergesort.
- Compute $p[j]$ for each j : $O(n \log n)$ via binary search.
- M-COMPUTE-OPT(j): each invocation takes $O(1)$ time and either
 - (1) returns an initialized value $M[j]$
 - (2) initializes $M[j]$ and makes two recursive calls
- Progress measure $\Phi = \# \text{ initialized entries among } M[1..n]$.
 - initially $\Phi = 0$; throughout $\Phi \leq n$.
 - (2) increases Φ by 1 $\Rightarrow \leq 2n$ recursive calls.
- Overall running time of M-COMPUTE-OPT(n) is $O(n)$. ■

Those who cannot remember the past are condemned to repeat it.

- Dynamic Programming

Weighted interval scheduling: finding a solution

Q. DP algorithm computes optimal value. How to find optimal solution?

A. Make a second pass by calling FIND-SOLUTION(n).

FIND-SOLUTION(j)

IF ($j = 0$)

RETURN \emptyset .

ELSE IF $(w_j + M[p[j]] > M[j-1])$

RETURN $\{j\} \cup \text{FIND-SOLUTION}(p[j]).$

ELSE

RETURN FIND-SOLUTION($j-1$).

$$M[j] = \max \{ M[j-1], w_j + M[p[j]] \}.$$

Analysis. # of recursive calls $\leq n \Rightarrow O(n)$.

Weighted interval scheduling: bottom-up dynamic programming

BOTTOM-UP($n, s_1, \dots, s_n, f_1, \dots, f_n, w_1, \dots, w_n$)

Sort jobs by finish time and renumber so that $f_1 \leq f_2 \leq \dots \leq f_n$.

Compute $p[1], p[2], \dots, p[n]$.

$M[0] \leftarrow 0$.

FOR $j = 1$ TO n

$M[j] \leftarrow \max \{ M[j-1], w_j + M[p[j]] \}$.

$M[0] = 0$

$M[1] =$

$M[2] =$

$M[3] =$

$M[4] =$

$M[5] =$

$M[6] =$

$M[7] =$

$M[8] =$

