Weighted interval scheduling

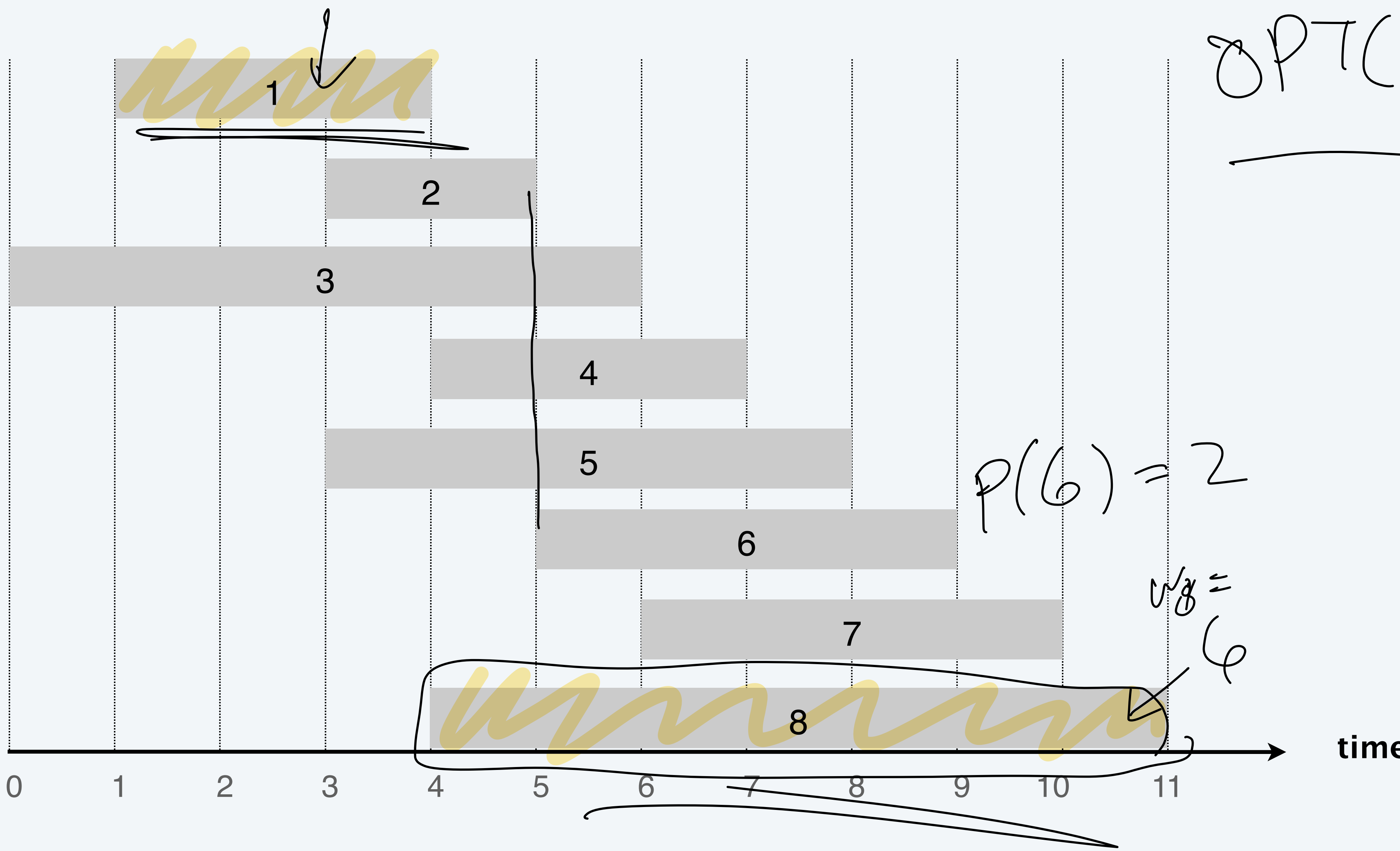
Convention. Jobs are in ascending order of finish time: $f_1 \leq f_2 \leq \ldots \leq f_n$.

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

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

i is rightmost interval that ends before j begins

$w_1 = 3$



$OPT(8) = 9$

Dynamic programming: binary choice

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

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 $1, 2, ..., j-1$.

$OPT(j-1)$

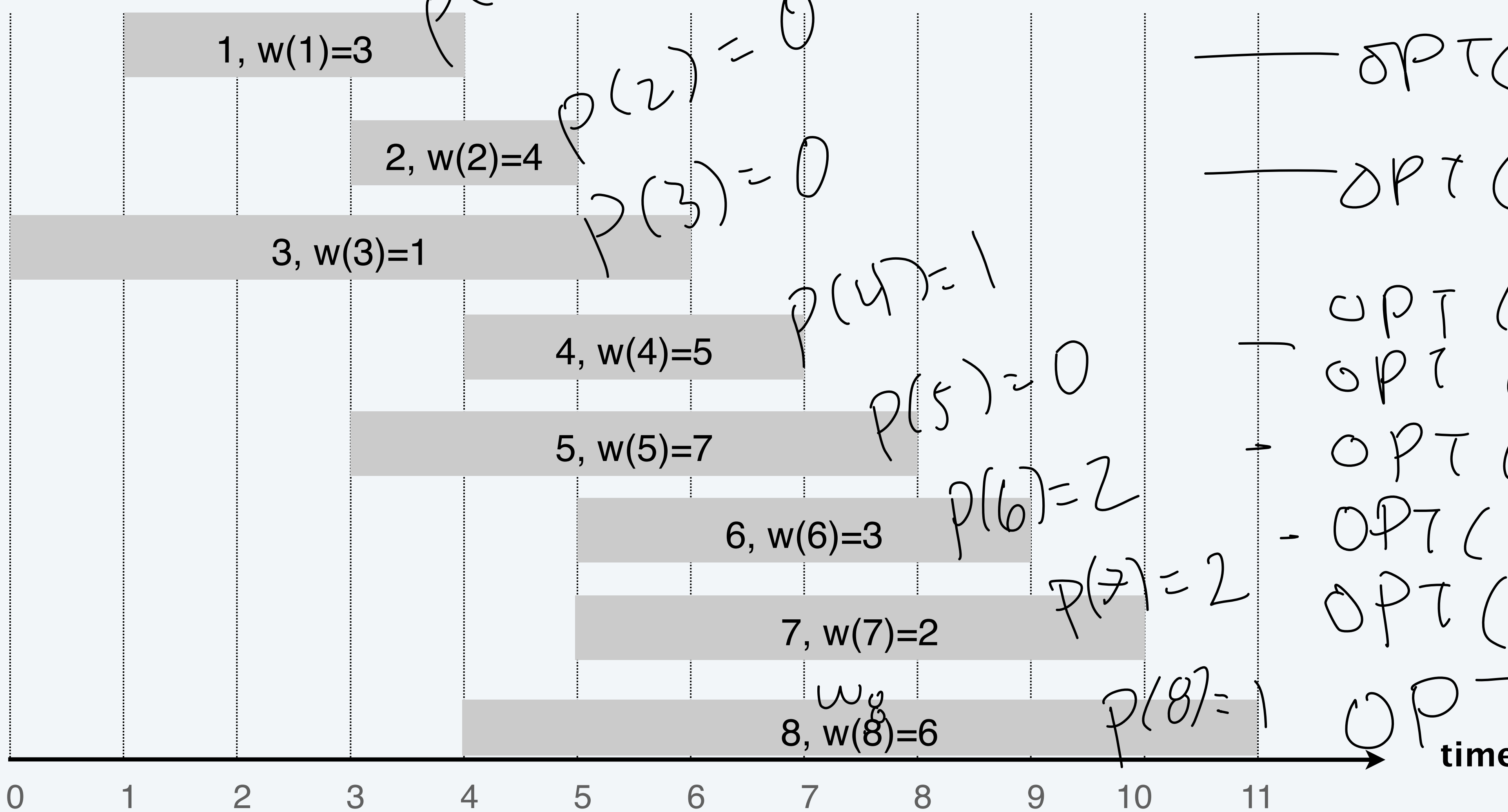
Case 2. $OPT(j)$ selects job j.

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

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

Weighted interval scheduling: what is max weight subset?

w_1



$p(1)=0$

$p(2)=0$

$p(3)=0$

$p(4)=1$

$p(5)=0$

$p(6)=2$

$p(7)=2$

w_8

$p(8)=1$

$OPT(0) = 0$

$OPT(1) = 3$

$OPT(2) = 4$

$OPT(3) = 4$

$OPT(4) = 8$

$OPT(5) = 8$

$OPT(6) = 8$

$OPT(7) = 8$

$OPT(8) = 9$

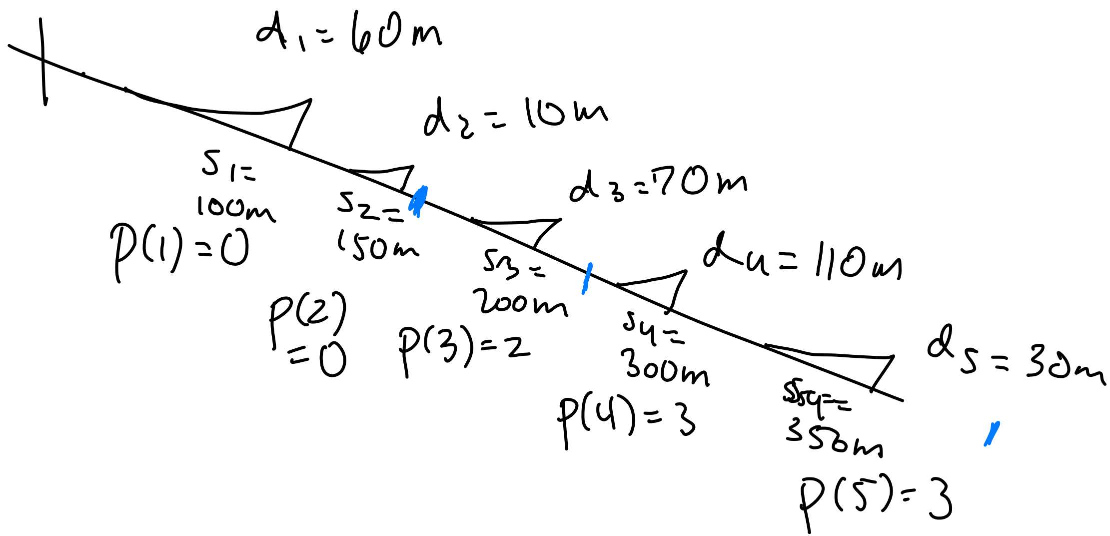
$OPT(j)$ = optimal weight using jobs 1 to j

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

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

Activity

an example



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

Weighted interval scheduling

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

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

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

RETURN COMPUTE-OPT(n).

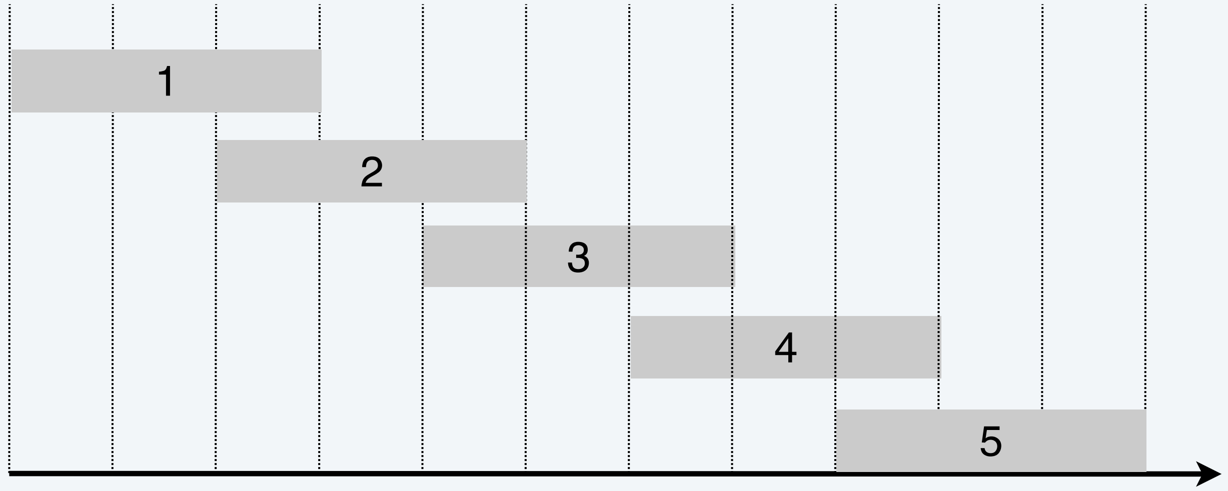
COMPUTE-OPT(j)

IF $(j = 0)$

  RETURN 0.

ELSE

  RETURN $\max\{$COMPUTE-OPT$(j-1)$, w_j + 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.

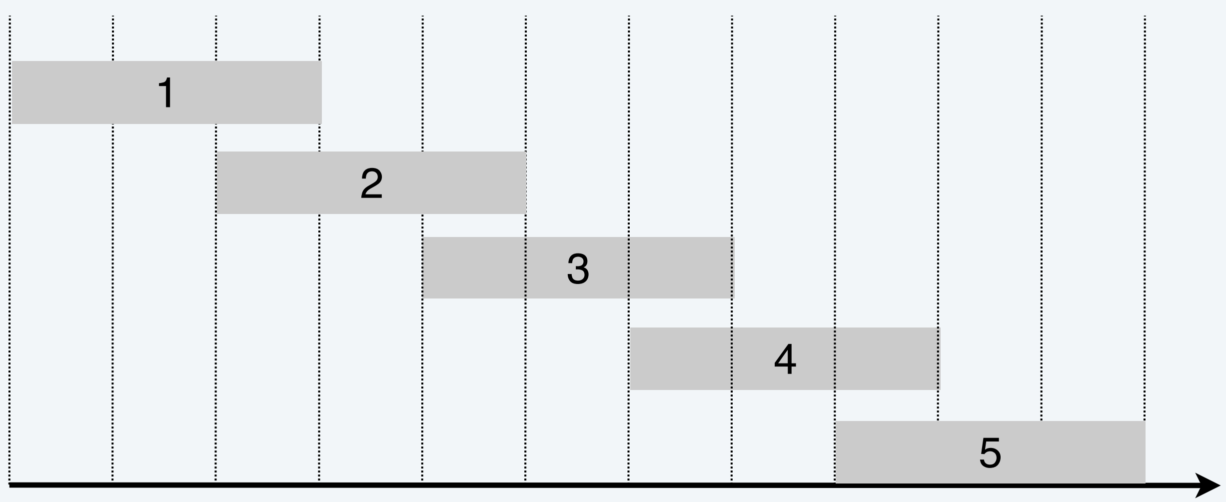
ELSE

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

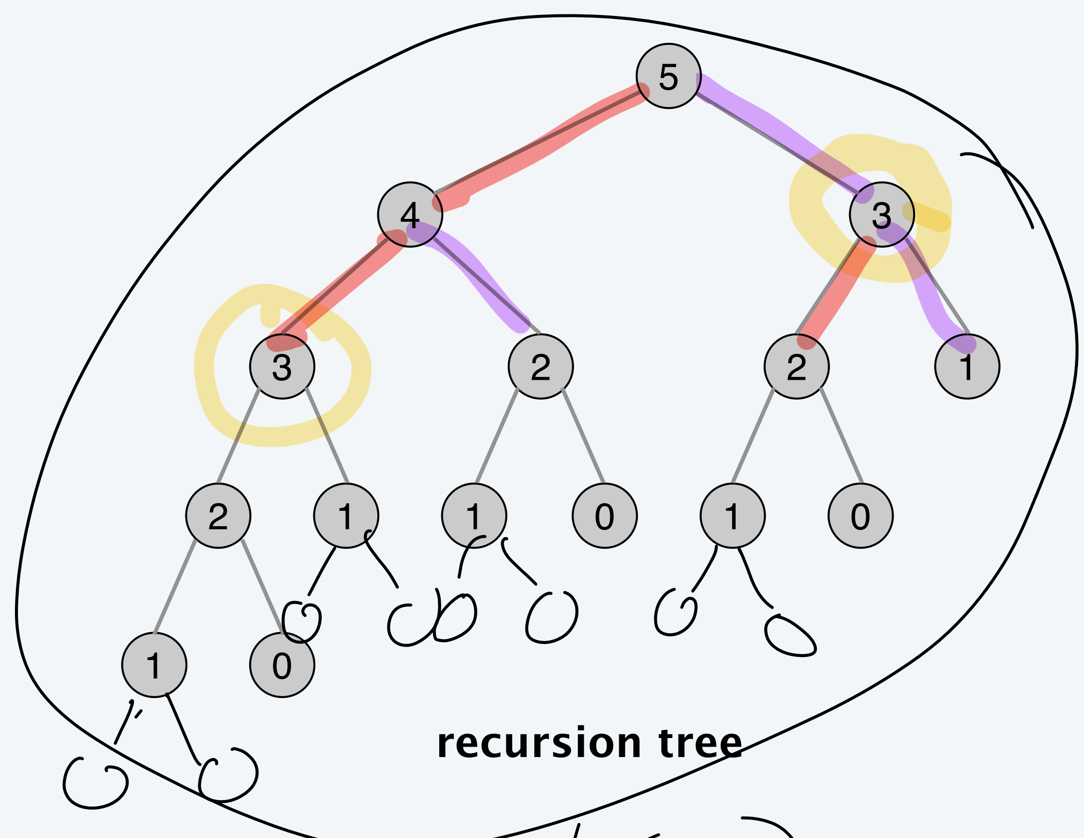
Weighted interval scheduling: brute force

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

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



p(1) = 0, p(j) = j–2



recursion tree

opt(5) opt(4) 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$(n, s_1, \ldots, s_n, f_1, \ldots, f_n, w_1, \ldots, w_n)$

Sort jobs by finish time and renumber so that $f_1 \le f_2 \le \ldots \le f_n$.

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

$M[0] \leftarrow 0.$ ← global array

RETURN M-COMPUTE-OPT(n).

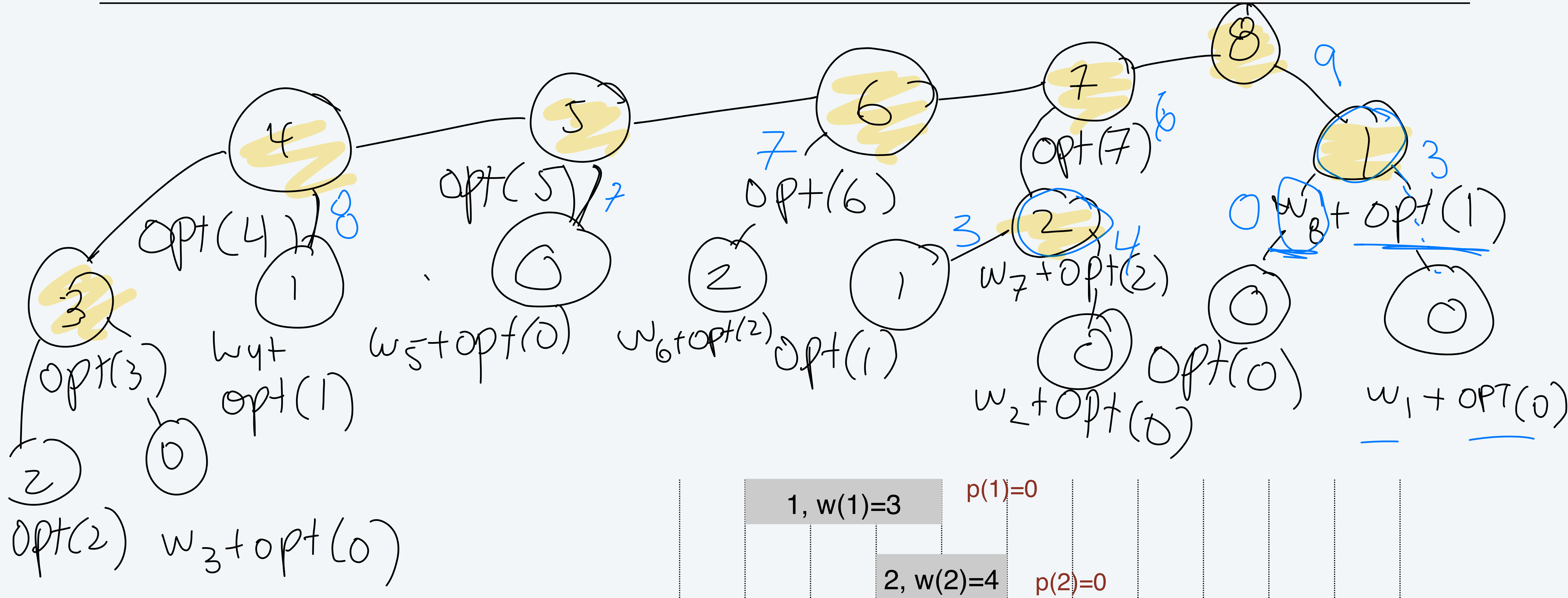
M-COMPUTE-OPT(j)

IF $(M[j]$ is uninitialized)

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

RETURN $M[j]$.

Trace through memoized version of Compute-OPT



M[0] = 0
M[1] = 3
M[2] = 4
M[3] = 4
M[4] = 8
M[5] = 8
M[6] = 8
M[7] = 8
M[8] = 9

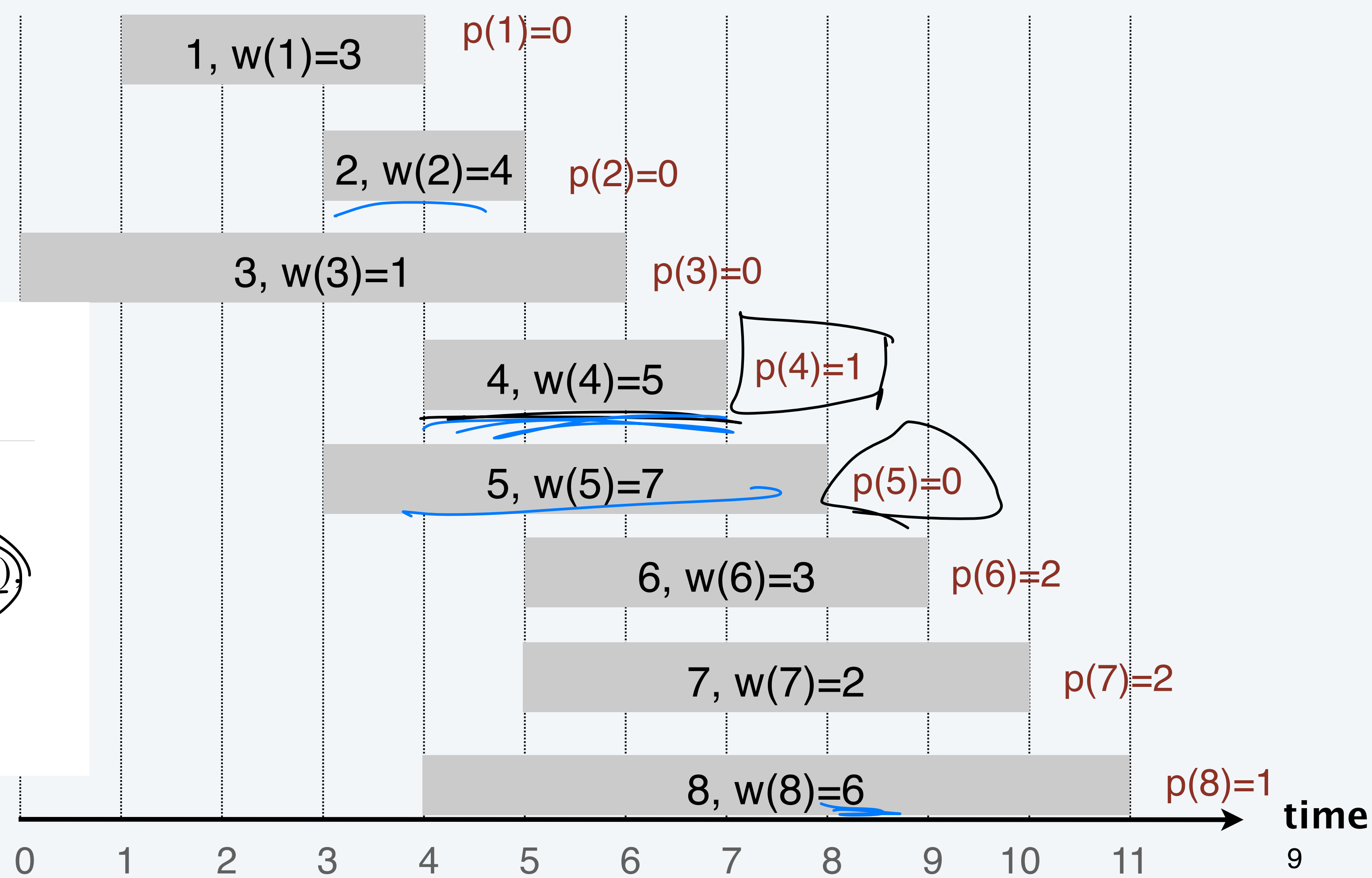
M-COMPUTE-OPT(j)

IF ($M[j]$ is uninitialized)

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

RETURN $M[j]$.

$w_8 + M[p(8)]$



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 Φ = # 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 $\varnothing$.
ELSE IF $(w_j + M[p[j]] > M[j-1])$
  RETURN $\{ j \} \cup$ 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 $\leq n \Rightarrow O(n)$.