CS 401

Dynamic Programming

Xiaorui Sun

Stuff

Homework 3 is due this Friday March 29 11:59pm

- Submission is now open (at Gradescope)

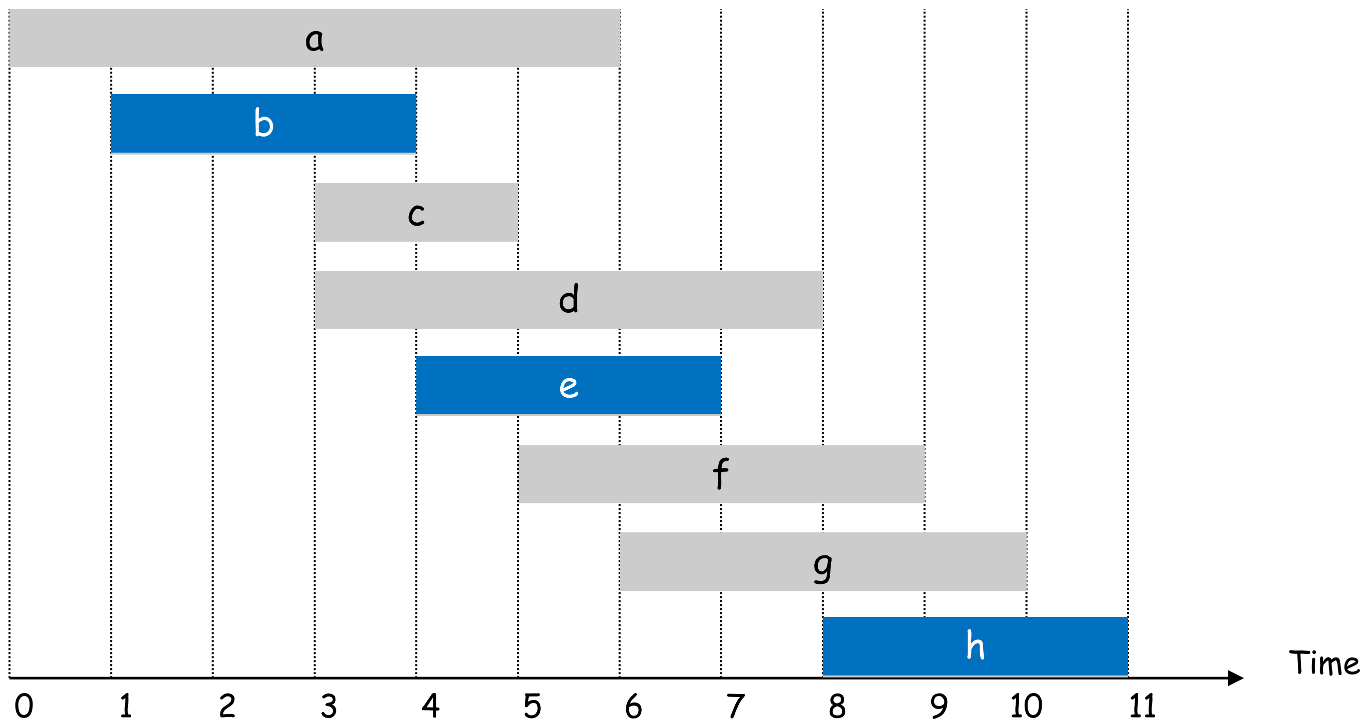
Course survey

- https://forms.gle/4gUVgQQhDGaFR2ge8
- Anonymous survey
- Collect the feedback regarding lectures/homework/midterm exam
- Is the homework/exam too easy or too hard?
- Your feedback will be used to adjust the difficulty of the rest homework and final exam

Weighted Interval Scheduling

Weighted Interval Scheduling

- Job j starts at $s(j)$ and finishes at $f(j)$ and has weight w_j
  - Two jobs compatible if they don't overlap.
  - Goal: find maximum weight subset of mutually compatible jobs.

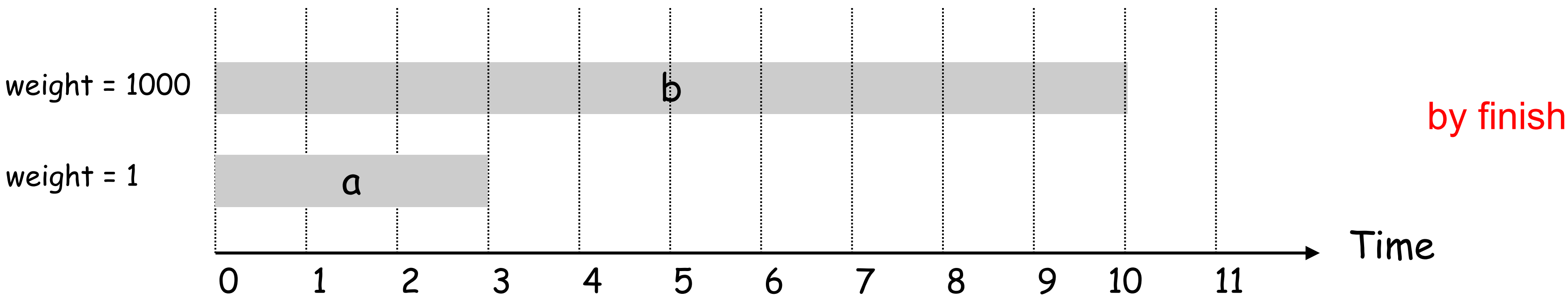


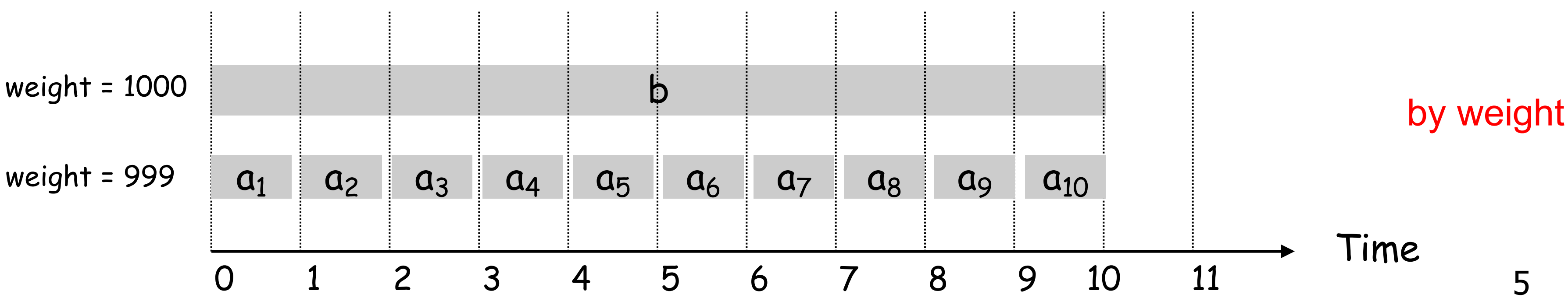
Unweighted Interval Scheduling: Review

Recall: Greedy algorithm works if all weights are 1:

- Consider jobs in ascending order of finishing time
- Add job to a subset if it is compatible with prev added jobs.

Observation: Greedy ALG fails spectacularly if arbitrary weights are allowed:





Weighted Job Scheduling by Induction

Suppose $1, \ldots, n$ ar

IH: Suppose ... of
jobs of size <

This idea works for any Optimization problem.

For NP-hard problems there is no ordering to reduce # subproblems

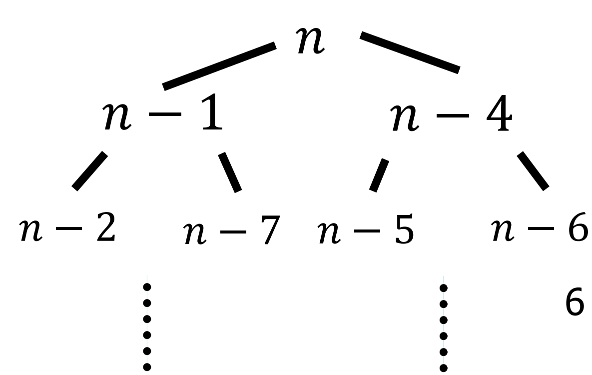
IS: Goal: For any n j

Case 1: Job n is not in OPT.
-- Then, just return OPT of $1, \ldots, n-1$.

Take best of the two

Case 2: Job n is in OPT.
-- Then, delete all jobs not compatible with n and recurse.

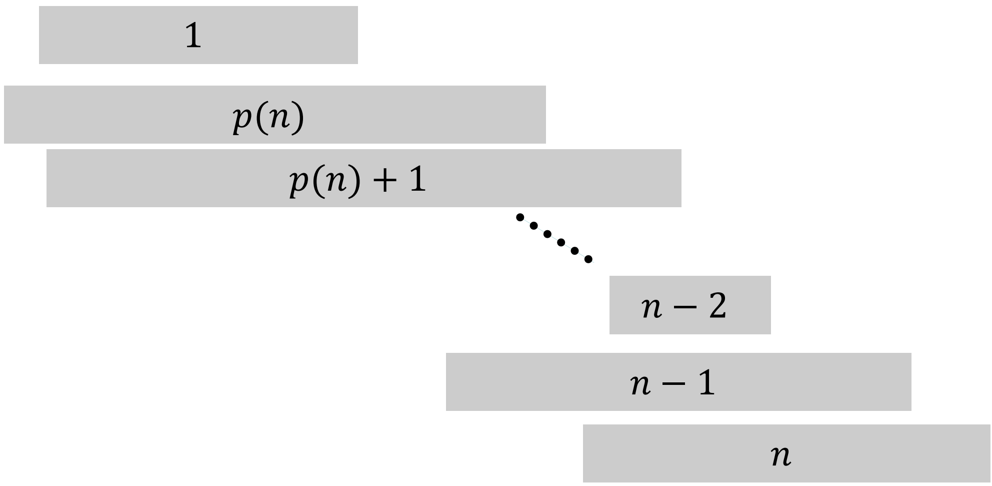
Q: Are we done?
A: No, How many subproblems are there?
Potentially 2^n all possible subsets of jobs.



Sorting to Reduce Subproblems

Sorting Idea: Label jobs by finishing time $f(1) \leq \cdots \leq f(n)$
IS: For jobs $1, \dots, n$ we want to compute OPT

Case 1: Suppose OPT has job n.
- So, all jobs i that are not compatible with n are not OPT
- Let $p(n)$ = largest index $i < n$ such that job i is compatible with n.
- Then, we just need to find OPT of $1, \dots, p(n)$



Sorting to reduce Subproblems

Sorting Idea: Label jobs by finishing time $f(1) \leq \cdots \leq f(n)$
IS: For jobs $1, \ldots, n$ we want to compute OPT

Case 1: Suppose OPT has job n.

- So, all jobs i that are not compatible with n are not OPT
- Let $p(n) =$ largest index $i < n$ such that job i is compatible with n.
- Then, we just need to find OPT of $1, \ldots, p(n)$

Take best of the two

Case 2: OPT does not select job n.

- Then, OPT is just the OPT of $1, \ldots, n-1$

Q: Have we made any progress?
A: Yes! This time every subproblem is of the form $1, \ldots, i$ for some i
So, at most n possible subproblems.

Weighted Job Scheduling by Induction

Sorting Idea: Label jobs by finishing time $f(1) \leq \cdots \leq f(n)$

Def $OPT(j)$ denote the weight of OPT solution of $1, \ldots, j$

The most important part of a correct DP; It fixes IH

To solve $OPT(j)$:

Case 1: $OPT(j)$ has job j.

- So, all jobs i that are not compatible with j are not $OPT(j)$.
- Let $p(j) =$ largest index $i < j$ such that job i is compatible with j.
- So $OPT(j) = OPT\big(p(j)\big) + w_j$.

Case 2: $OPT(j)$ does not select job j.

- Then, $OPT(j) = OPT(j-1)$.

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

Algorithm

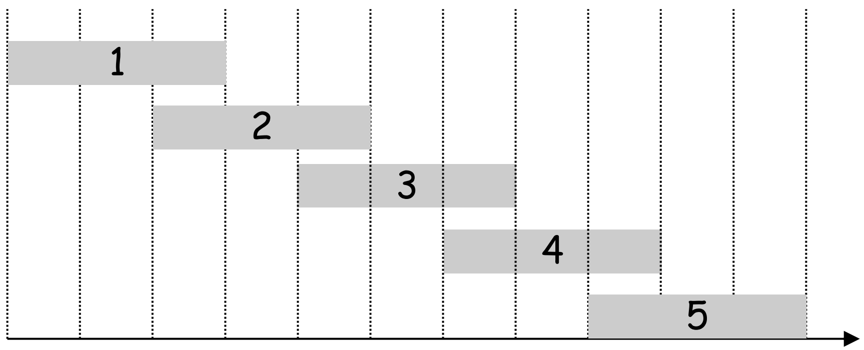
```
Input: n, s(1),...,s(n) and f(1),...,f(n) and w_1,...,w_n.

Sort jobs by finish times so that f(1) ≤ f(2) ≤ ⋯ f(n).

Compute p(1),p(2),...,p(n)

OPT(j) {
   if ( j = 0 )
      return 0
   else
      return max (w_j + OPT(p(j)), OPT(j − 1)).
}
```

Recursive Algorithm Fails

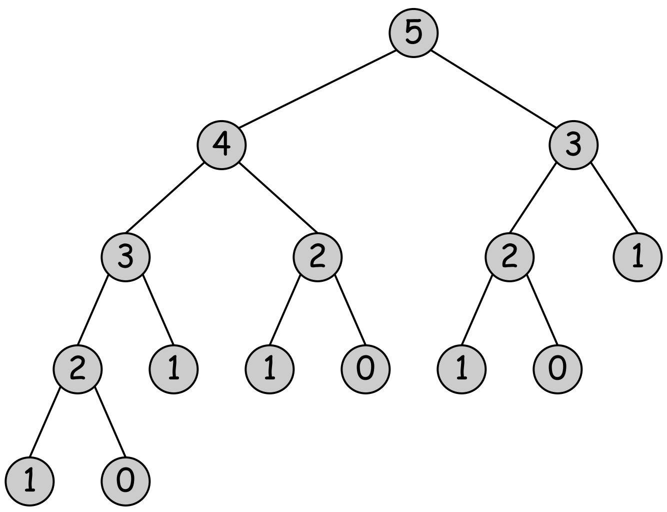
Even though we have only n subproblems, we do not store the solution to the subproblems

➢So, we may re-solve the same problem many many times.

Ex. Number of recursive calls for family of "layered" instances grows exponentially



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



Algorithm with Memoization

Memorization. Compute and Store the solution of each sub-problem in a cache the first time that you face it. lookup as needed.

```
Input: n, s(1),…,s(n)  and f(1),…,f(n) and w_1,…,w_n.

Sort jobs by finish times so that f(1) ≤ f(2) ≤ ⋯ f(n).

Compute p(1),p(2),…,p(n)

for j = 1 to n
   M[j] = empty
M[0] = 0

OPT(j) {
   if (M[j] is empty)
      M[j] = max(w_j + OPT(p(j)), OPT(j − 1)).
   return M[j]
}
```

In practice, you may get stack overflow if $n \gg 10^6$ (depends on the language).

Bottom up Dynamic Programming

You can also avoid recursion

- recursion may be easier conceptually when you use induction

```
Input: n, s(1),...,s(n)  and f(1),...,f(n) and w_1,...,w_n.

Sort jobs by finish times so that f(1) ≤ f(2) ≤ ⋯ f(n).          O(n log n)

Compute p(1),p(2),...,p(n) <------------ Binary search            O(n log n)

M[0] = 0
for j = 1 to n
   M[j] = max(w_j + M[p(j)], M[j − 1]).                            O(n)


Output M[n]
```

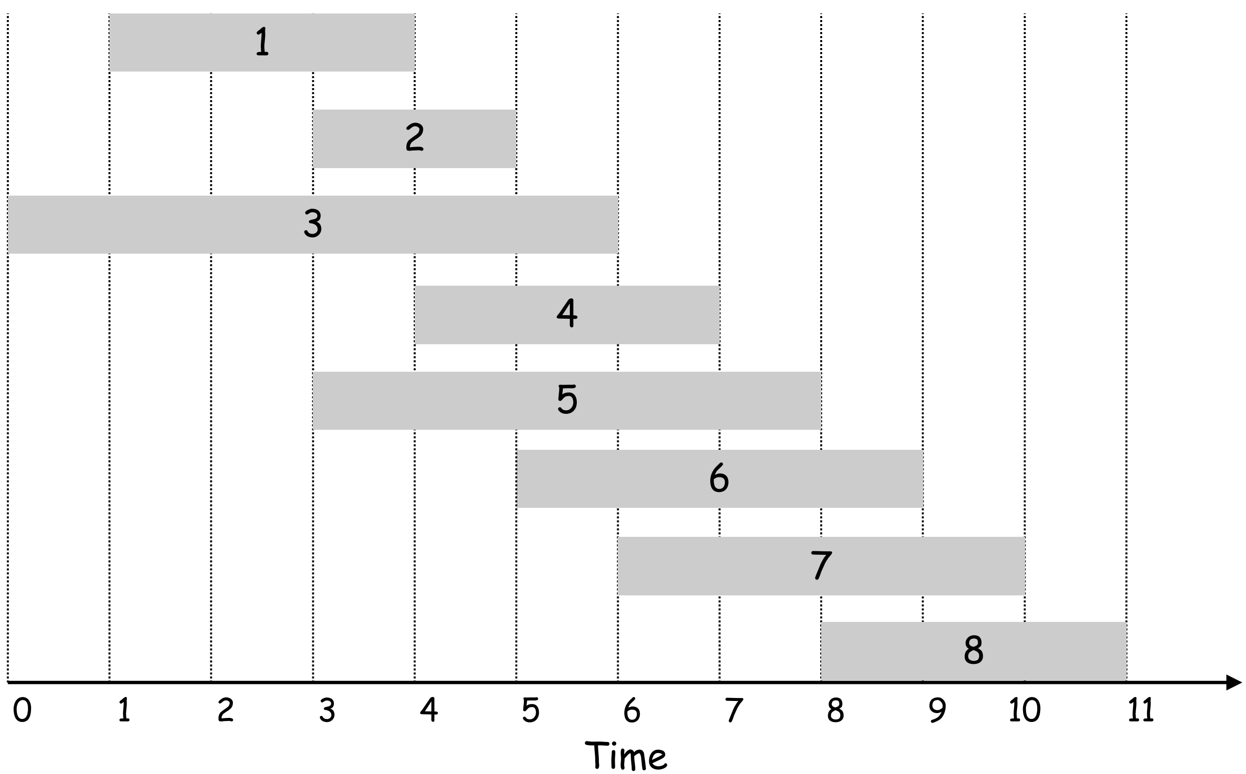
Claim: $M[j]$ is value of $OPT(j)$

Example

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

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

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



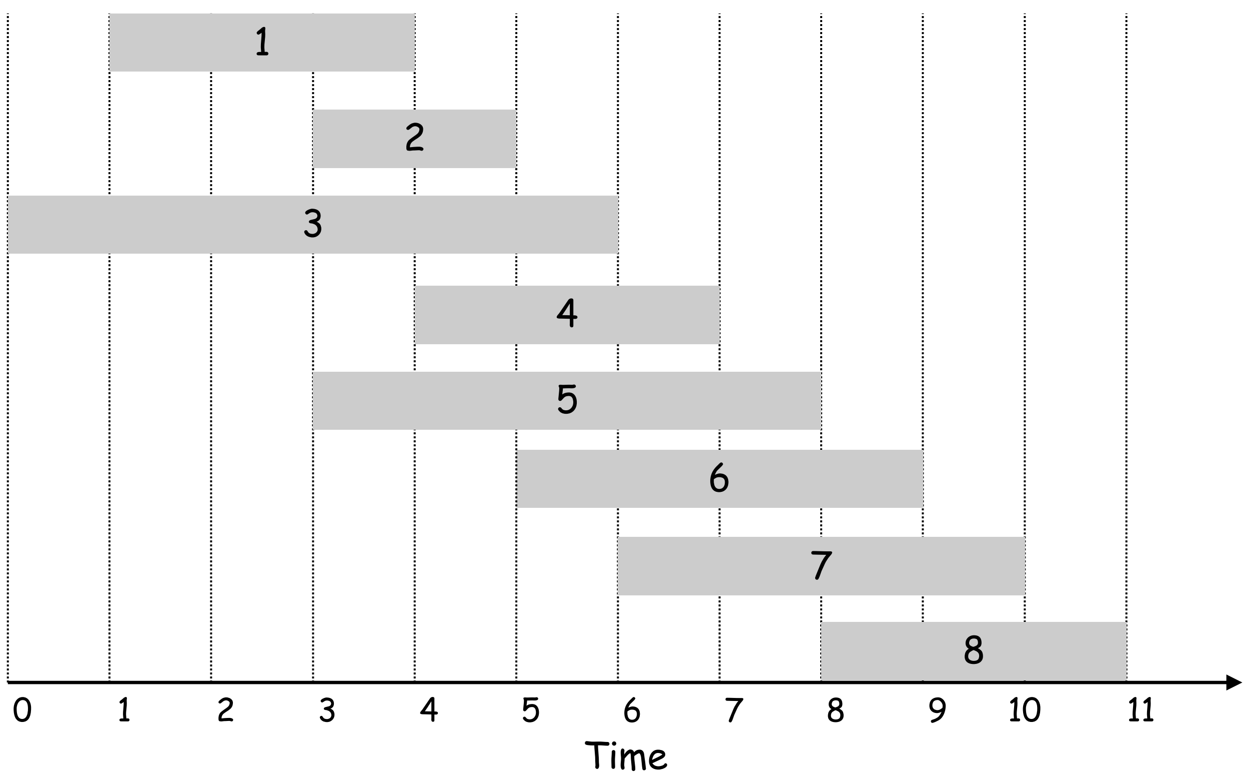
j	w_j	p(j)	OPT(j)
0			0
1	3	0	
2	4	0	
3	1	0	
4	3	1	
5	4	0	
6	3	2	
7	2	3	
8	4	5	

Example

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

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

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



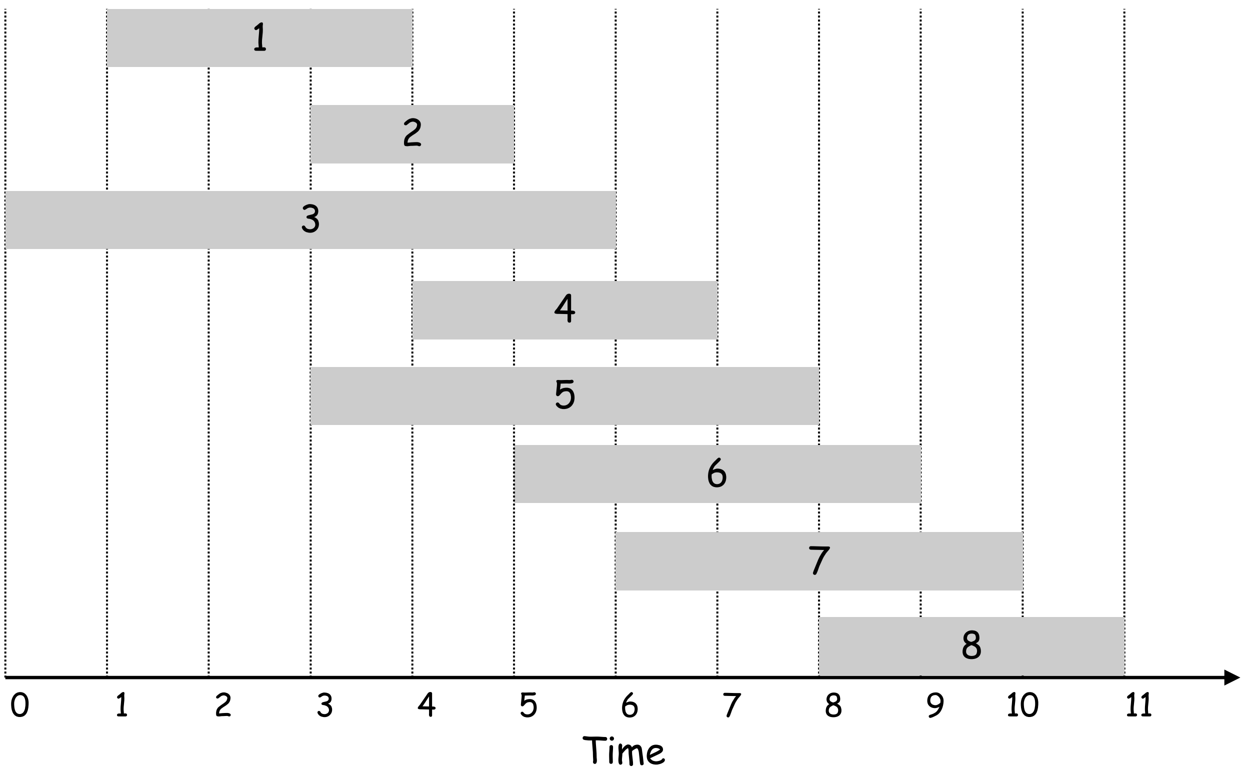
j	w_j	p(j)	OPT(j)
0			0
1	3	0	3
2	4	0	
3	1	0	
4	3	1	
5	4	0	
6	3	2	
7	2	3	
8	4	5	

Example

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

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

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



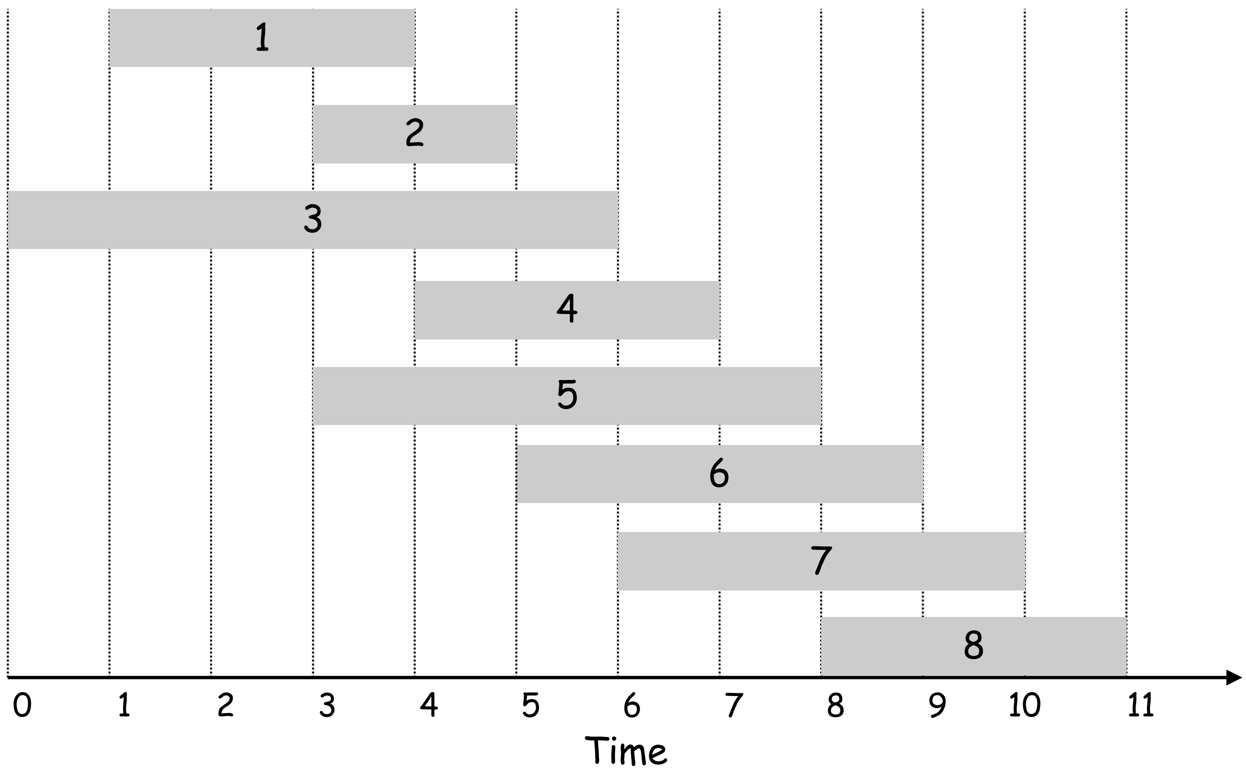
j	w_j	p(j)	OPT(j)
0			0
1	3	0	3
2	4	0	4
3	1	0	
4	3	1	
5	4	0	
6	3	2	
7	2	3	
8	4	5	

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

Example

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

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



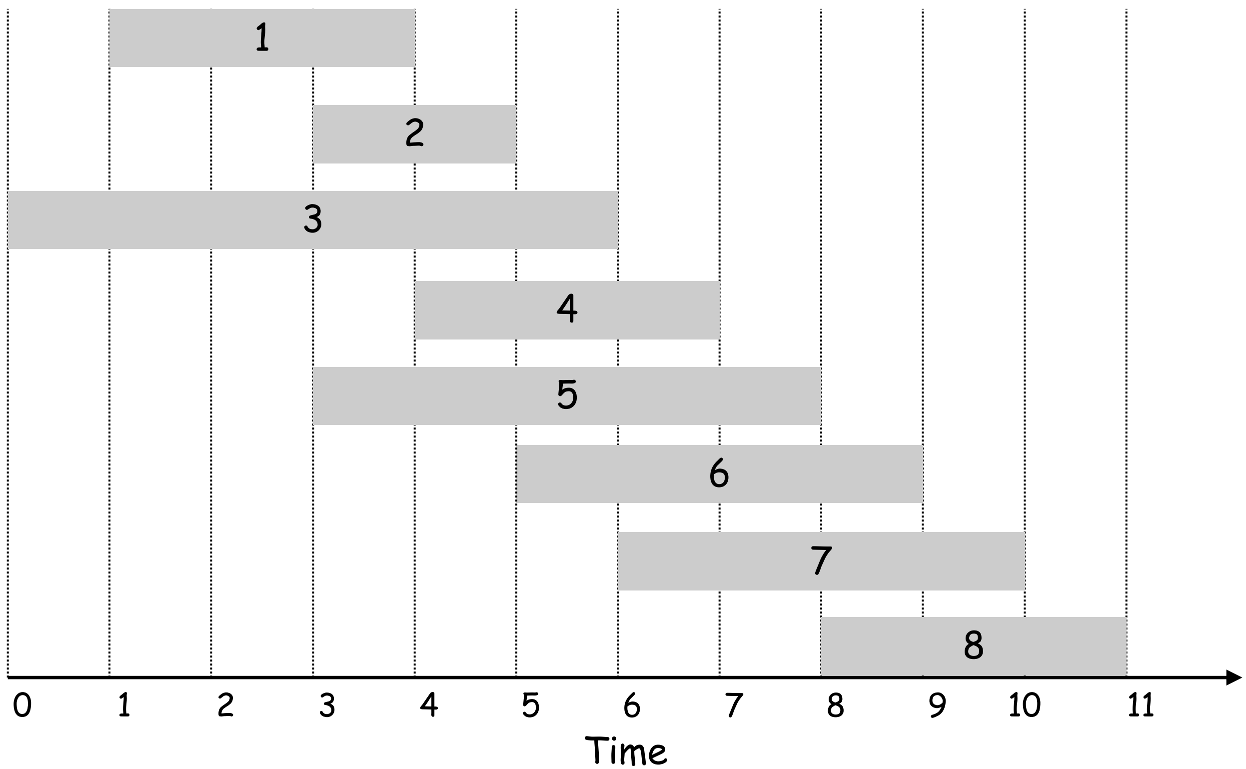
j	w_j	p(j)	OPT(j)
0			0
1	3	0	3
2	4	0	4
3	1	0	4
4	3	1	
5	4	0	
6	3	2	
7	2	3	
8	4	5	

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

Example

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

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



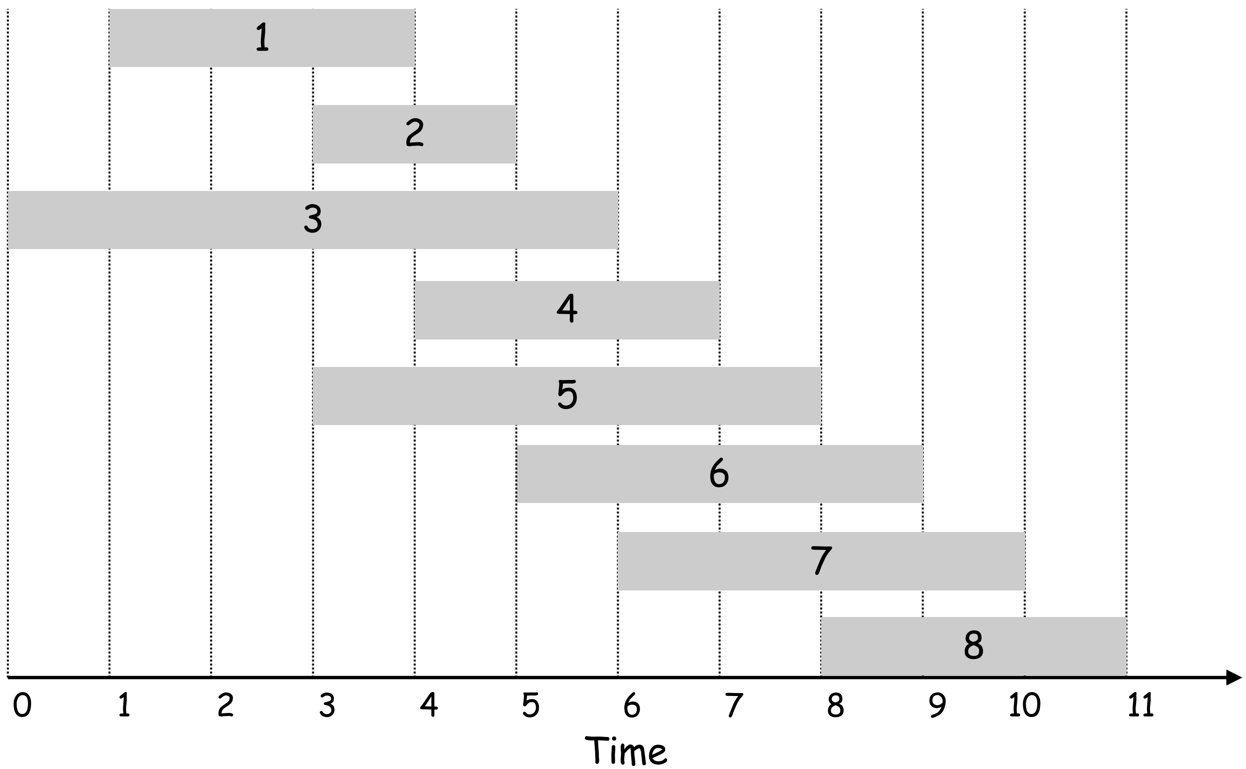
j	w_j	p(j)	OPT(j)
0			0
1	3	0	3
2	4	0	4
3	1	0	4
4	3	1	6
5	4	0	
6	3	2	
7	2	3	
8	4	5	

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

Example

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

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



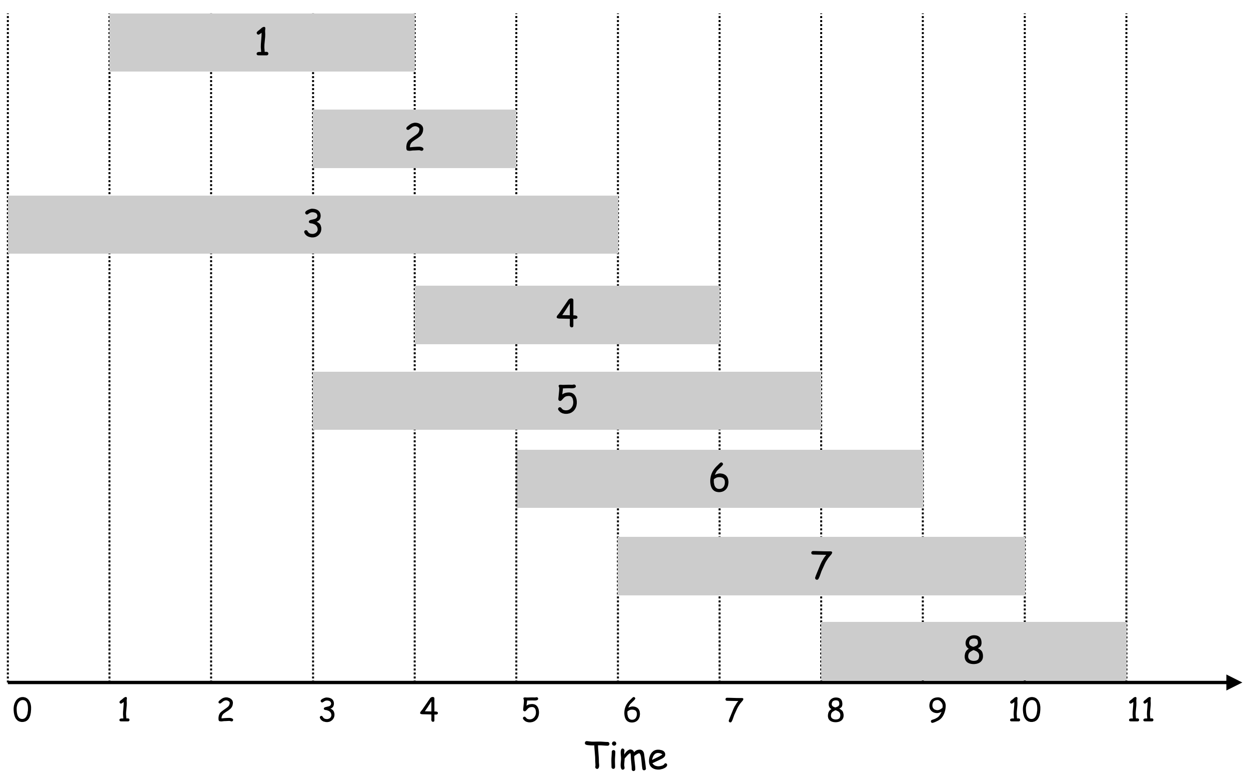
j	w_j	p(j)	OPT(j)
0			0
1	3	0	3
2	4	0	4
3	1	0	4
4	3	1	6
5	4	0	6
6	3	2	
7	2	3	
8	4	5	

Example

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

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

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



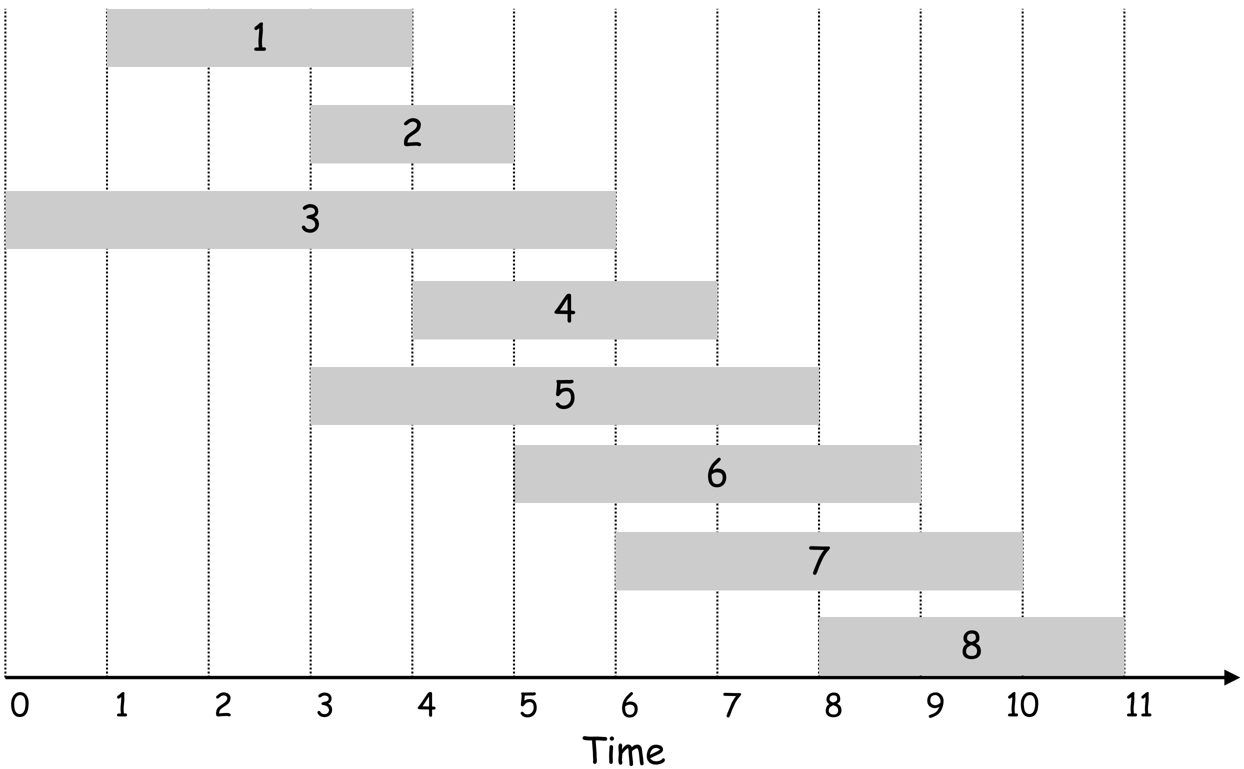
j	w_j	p(j)	OPT(j)
0			0
1	3	0	3
2	4	0	4
3	1	0	4
4	3	1	6
5	4	0	6
6	3	2	7
7	2	3	
8	4	5	

Example

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

Label jobs by finishing time: $f(1) \le \cdots \le f(n)$.

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



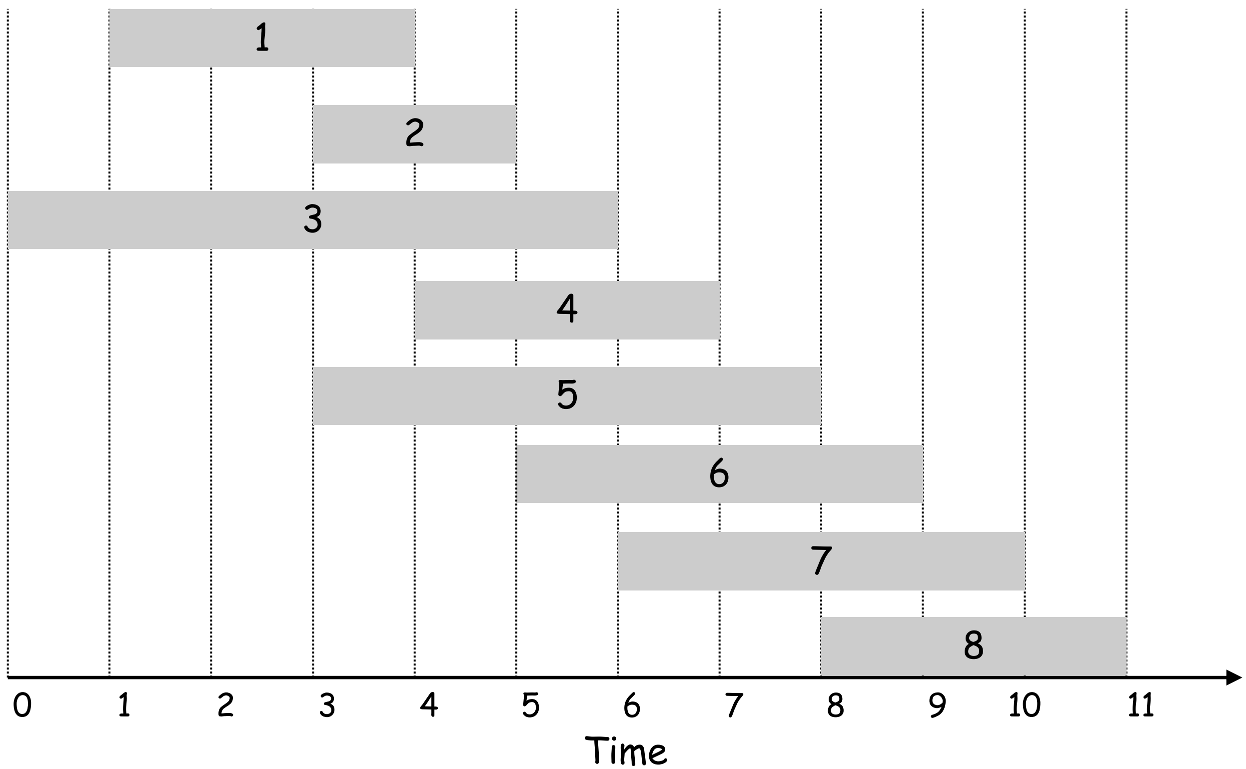
j	w_j	p(j)	OPT(j)
0			0
1	3	0	3
2	4	0	4
3	1	0	4
4	3	1	6
5	4	0	6
6	3	2	7
7	2	3	7
8	4	5	

Example

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

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

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



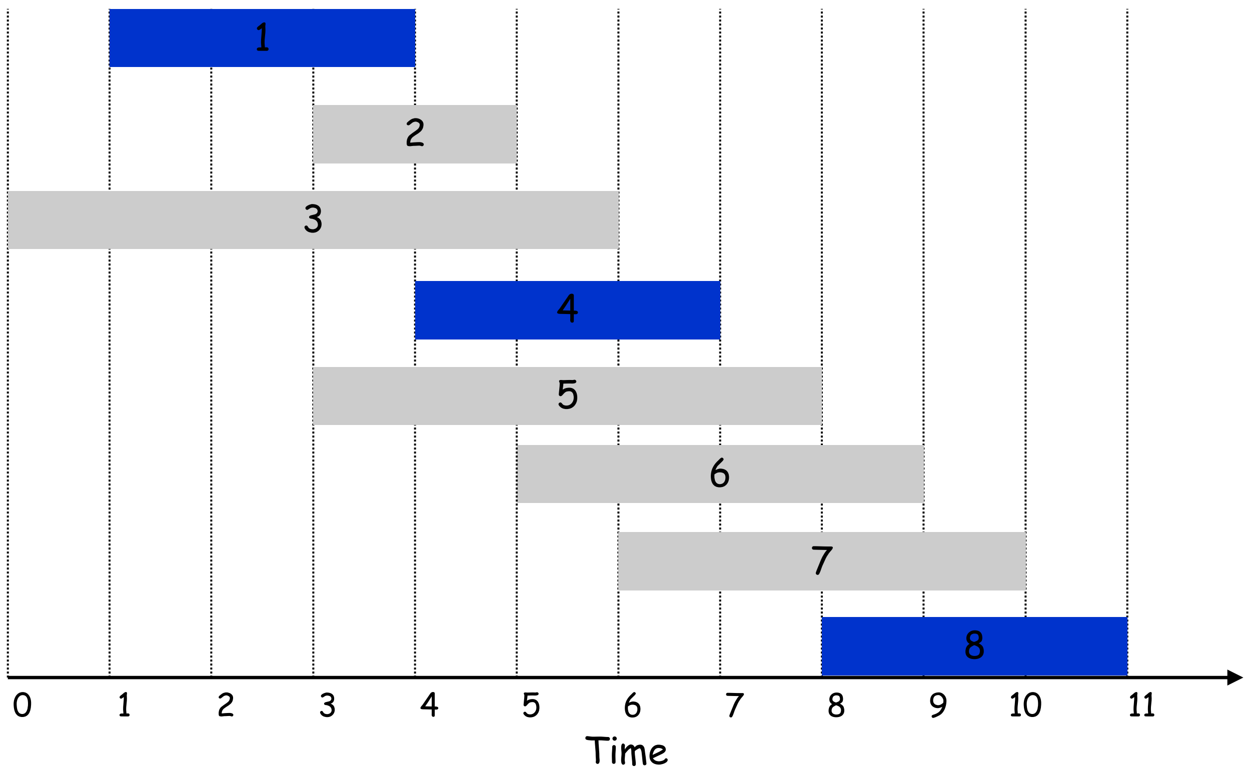
j	w_j	p(j)	OPT(j)
0			0
1	3	0	3
2	4	0	4
3	1	0	4
4	3	1	6
5	4	0	6
6	3	2	7
7	2	3	7
8	4	5	**10**

Example

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

Label jobs by finishing time: $f(1) \leq \cdots \leq f(n)$.

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



j	w_j	p(j)	OPT(j)
0			0
1	3	0	3
2	4	0	4
3	1	0	4
4	3	1	6
5	4	0	6
6	3	2	7
7	2	3	7
8	4	5	**10**

Dynamic Programming

- Optimal substructure: Optimal solution of a problem can be obtained from optimal solutions of smaller (overlapping) sub-problems
- Useful when the same subproblems show up again and again in the solution.