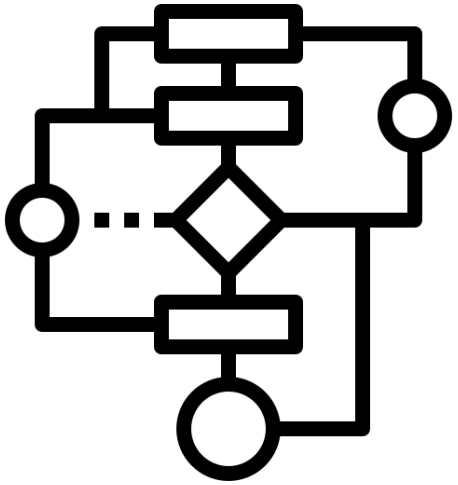
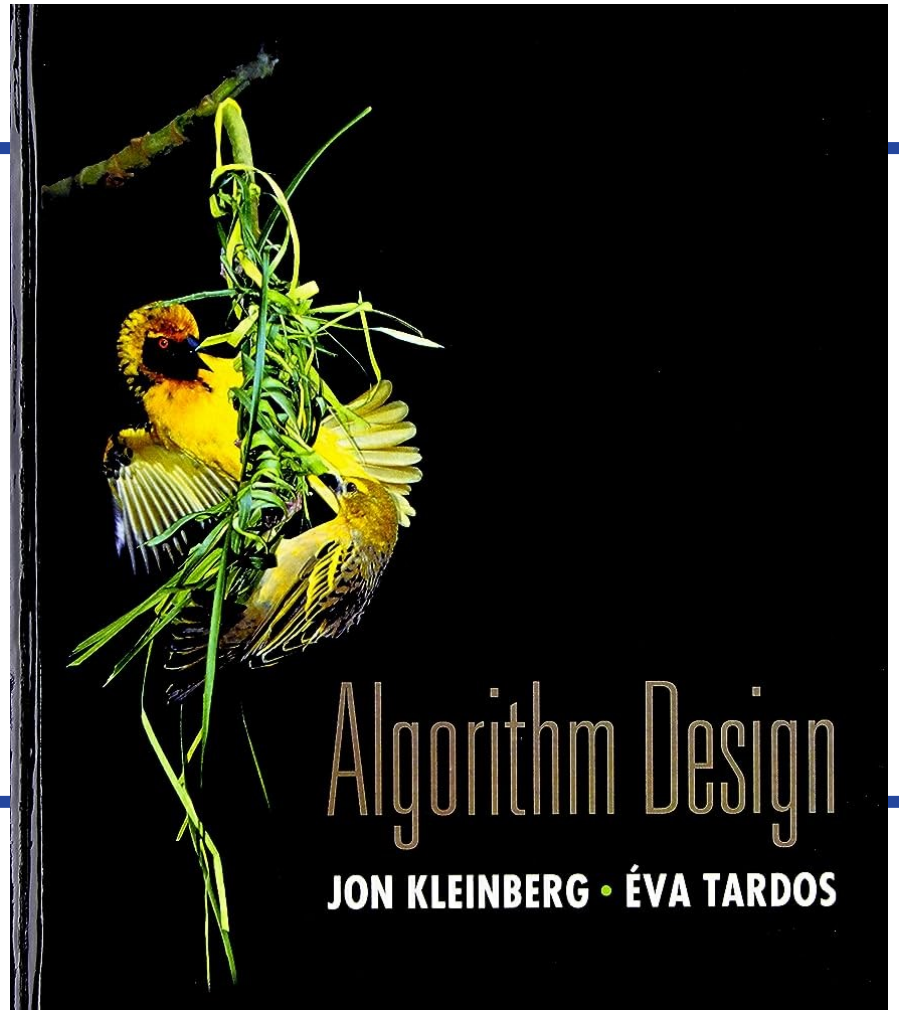


CS 310: Algorithms

Lecture 18

Instructor: Naveed Anwar Bhatti



Chapter 6:
Dynamic Programming

Section :
Weighed Interval Scheduling Problem

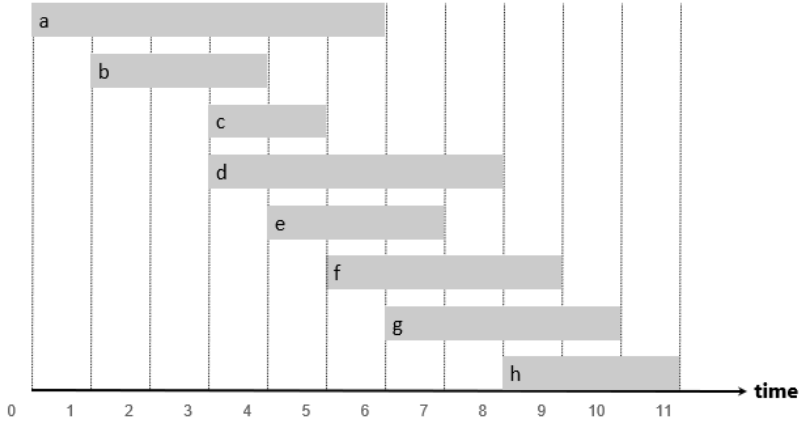
Weighted Interval Scheduling Problem

Do you guys remember "Interval Scheduling Problem"?

The goal was to accept ***maximum number of non overlapping jobs***

Interval Scheduling Problem

- Job j starts at s_j and finishes at f_j.
- Two jobs are compatible if they don't overlap.



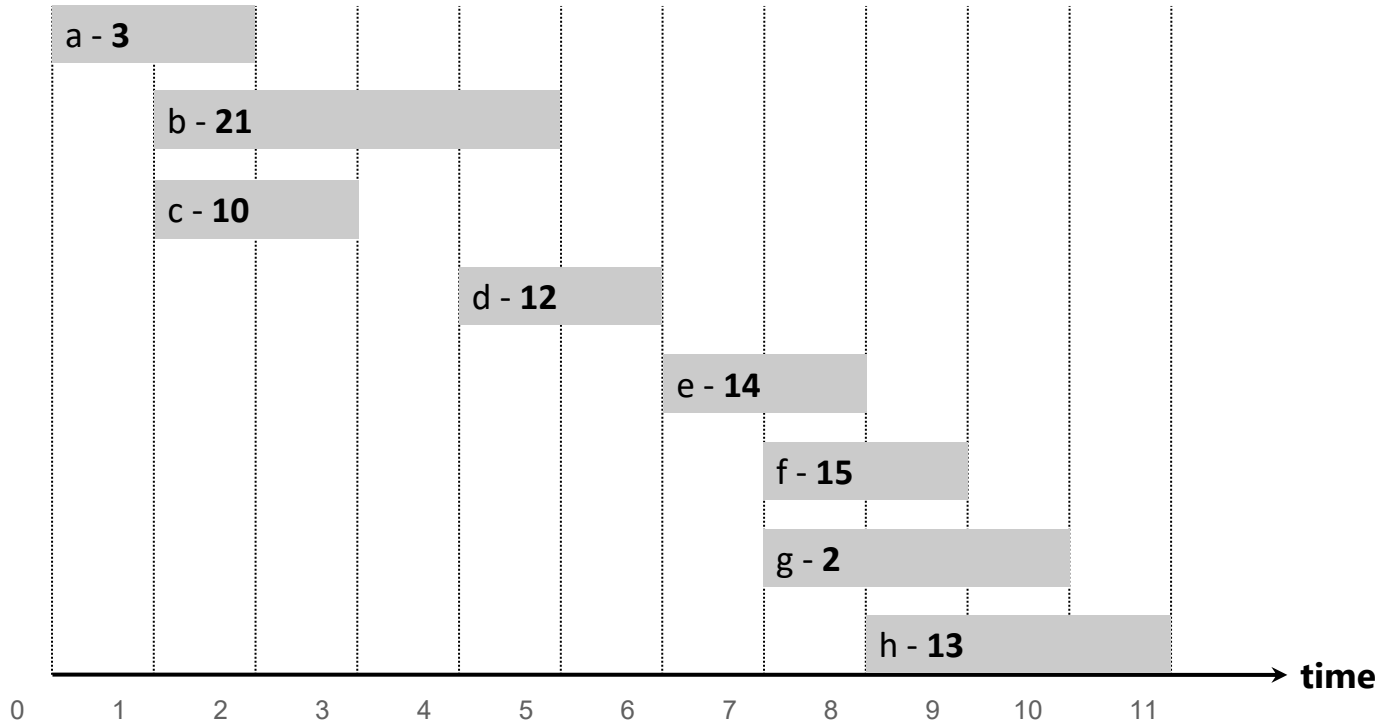
Weighted Interval Scheduling Problem

Now

- Multiple jobs to schedule
- Each job species a start time and finish time
- **Each job has a value (weight)**
- Problem is to schedule (accept/reject) the requests
- Selected requests must not overlap
- **Goal is to accept jobs with maximum total value**

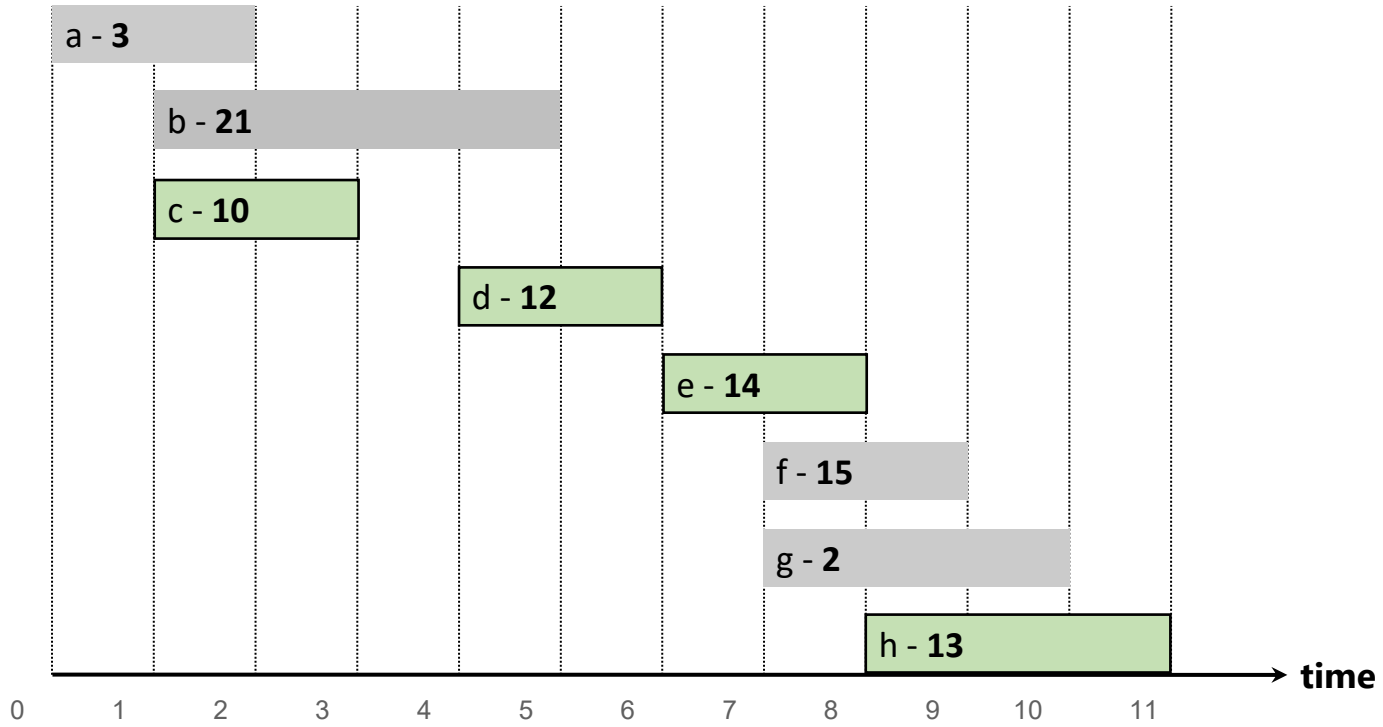
Weighted Interval Scheduling Problem

What job(s) should we select?



Weighted Interval Scheduling Problem

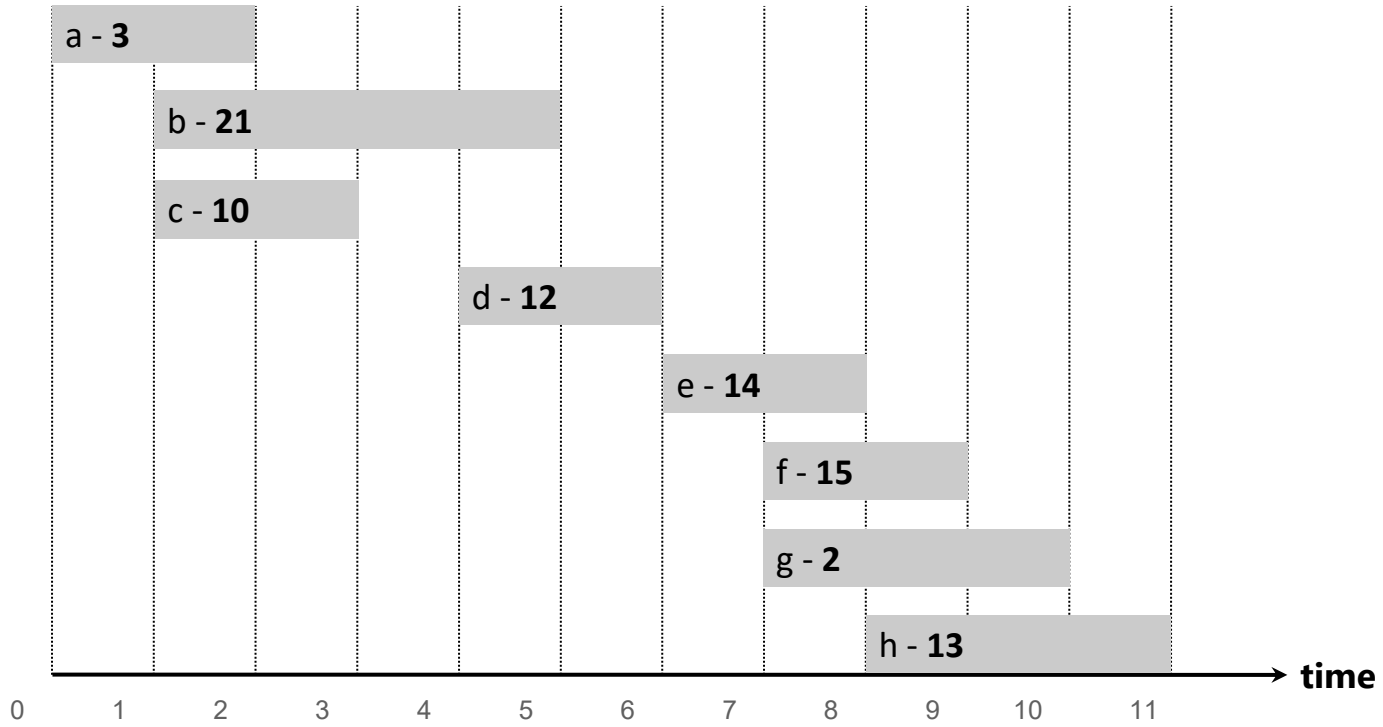
Optimal Solution



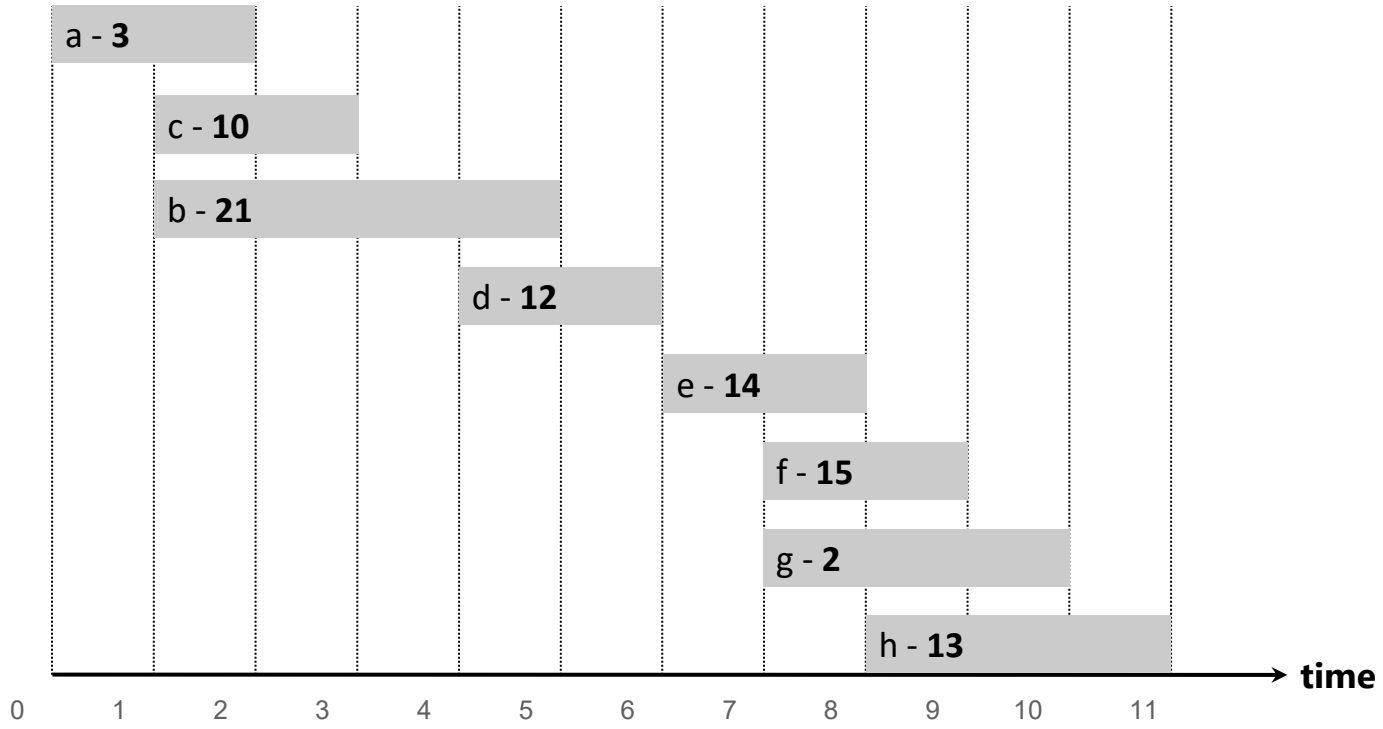
Total weight of 49

Weighted Interval Scheduling Problem

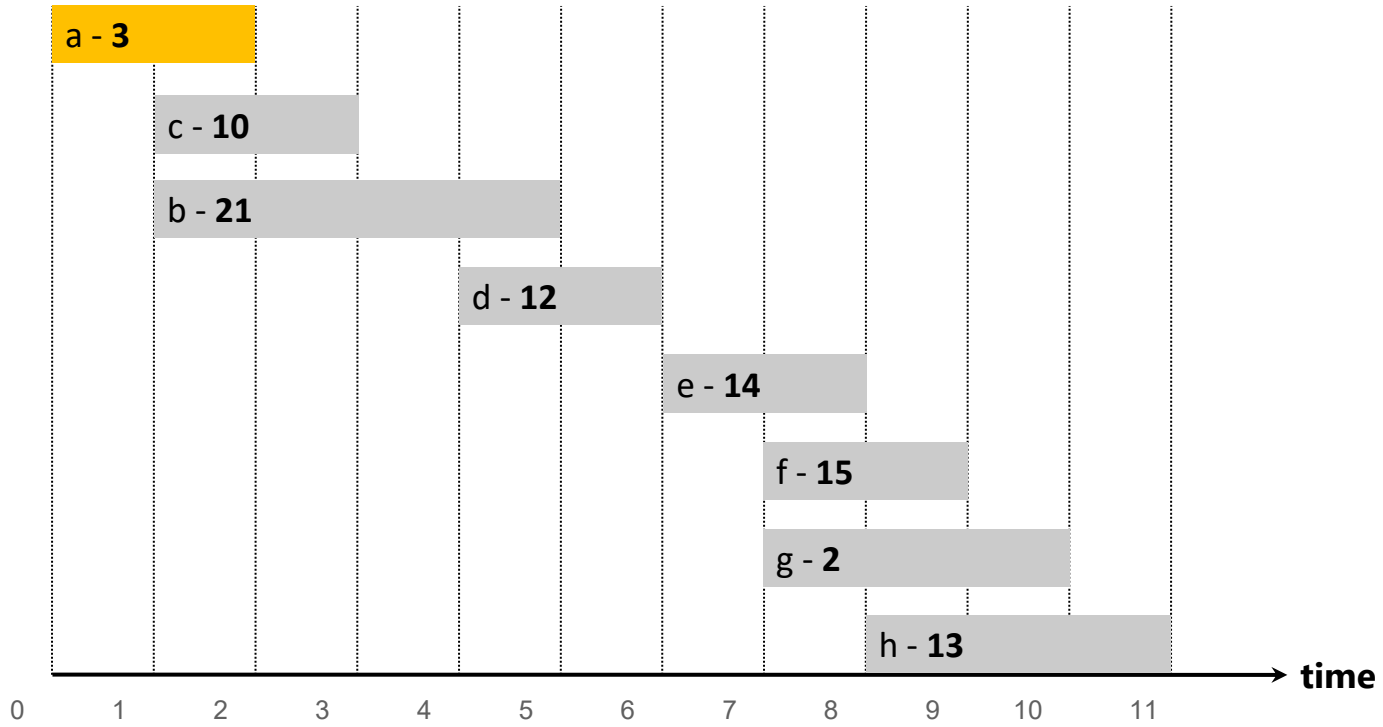
- Last time… **EFTF** (greedy approach) worked and gave optimal result



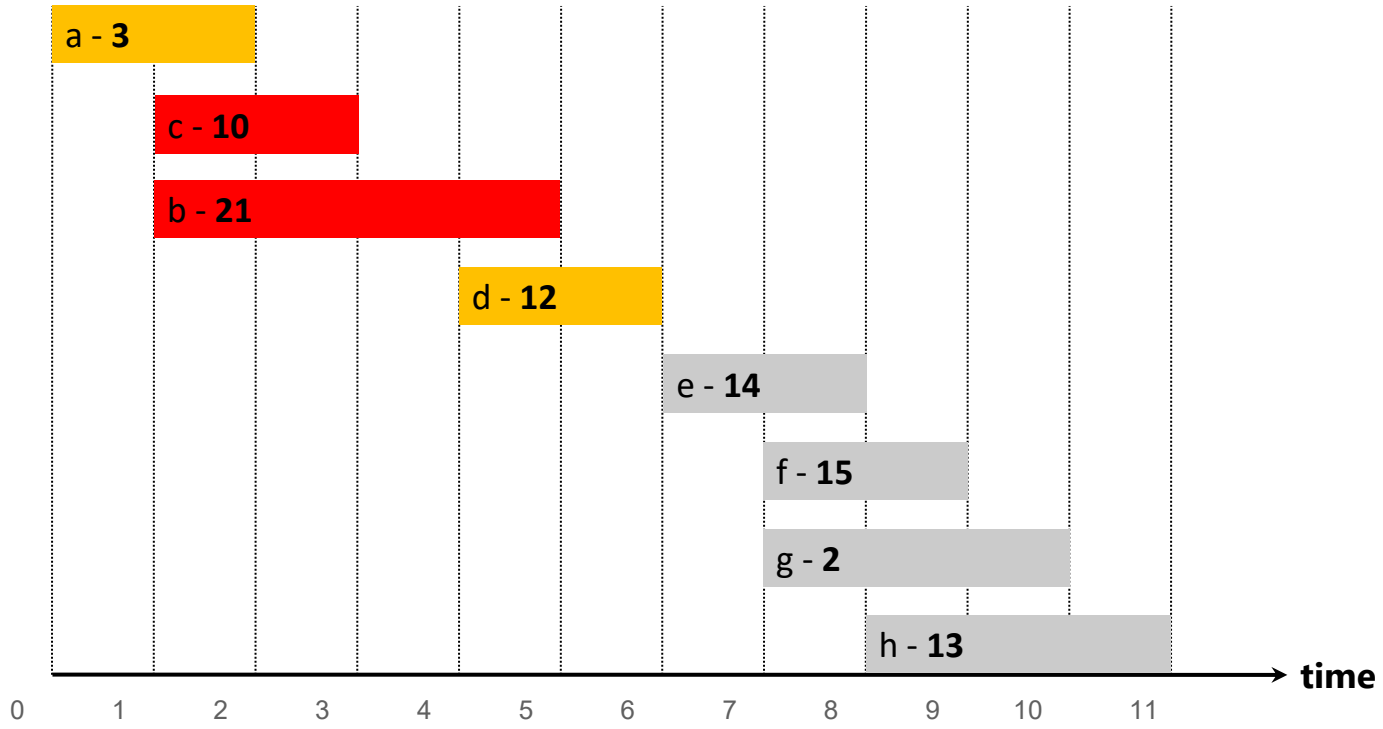
Weighted Interval Scheduling Problem



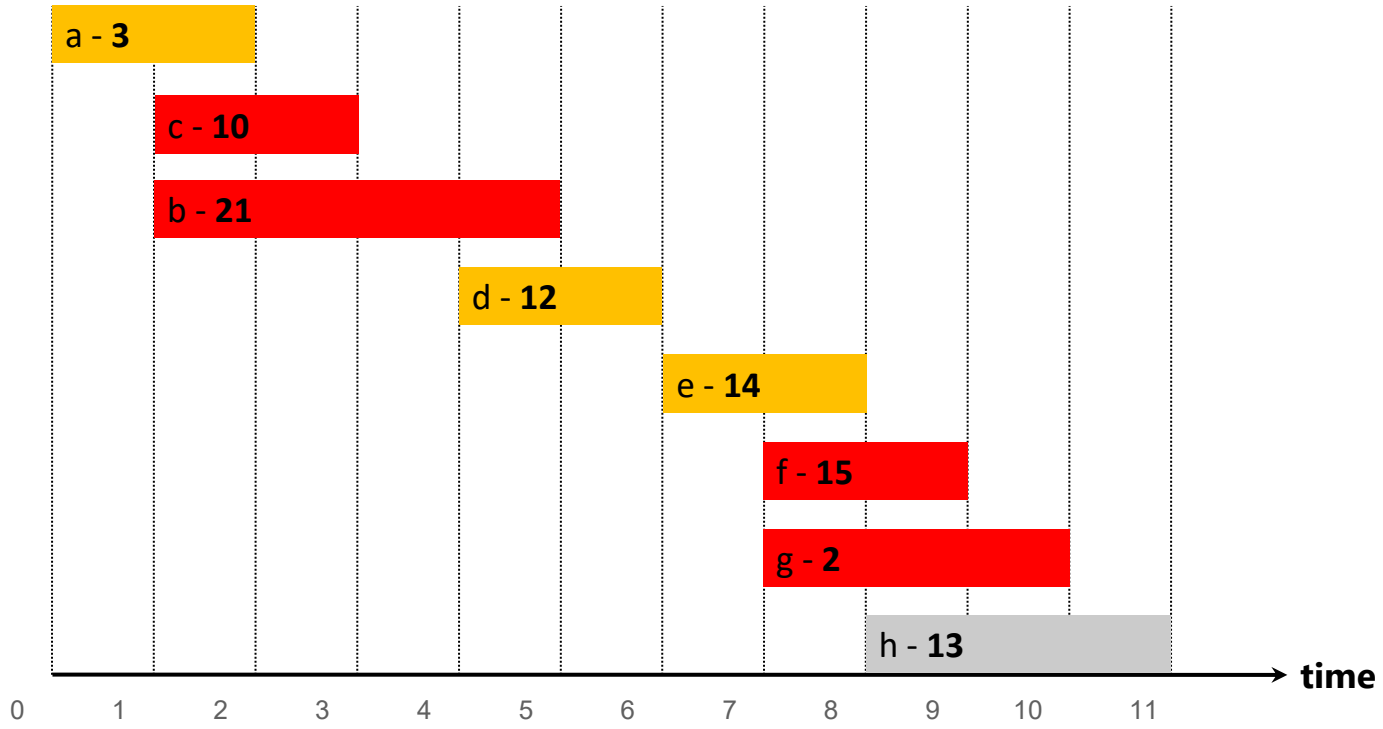
Weighted Interval Scheduling Problem



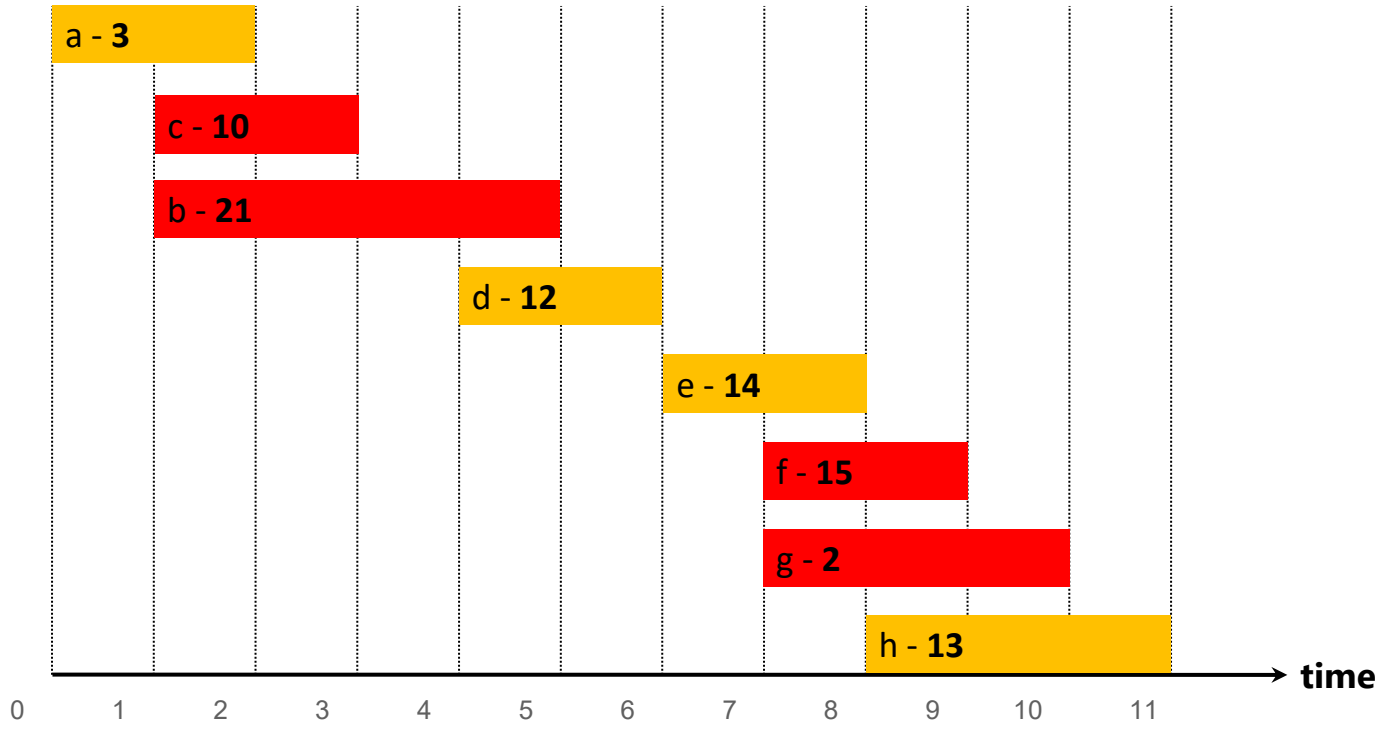
Weighted Interval Scheduling Problem



Weighted Interval Scheduling Problem

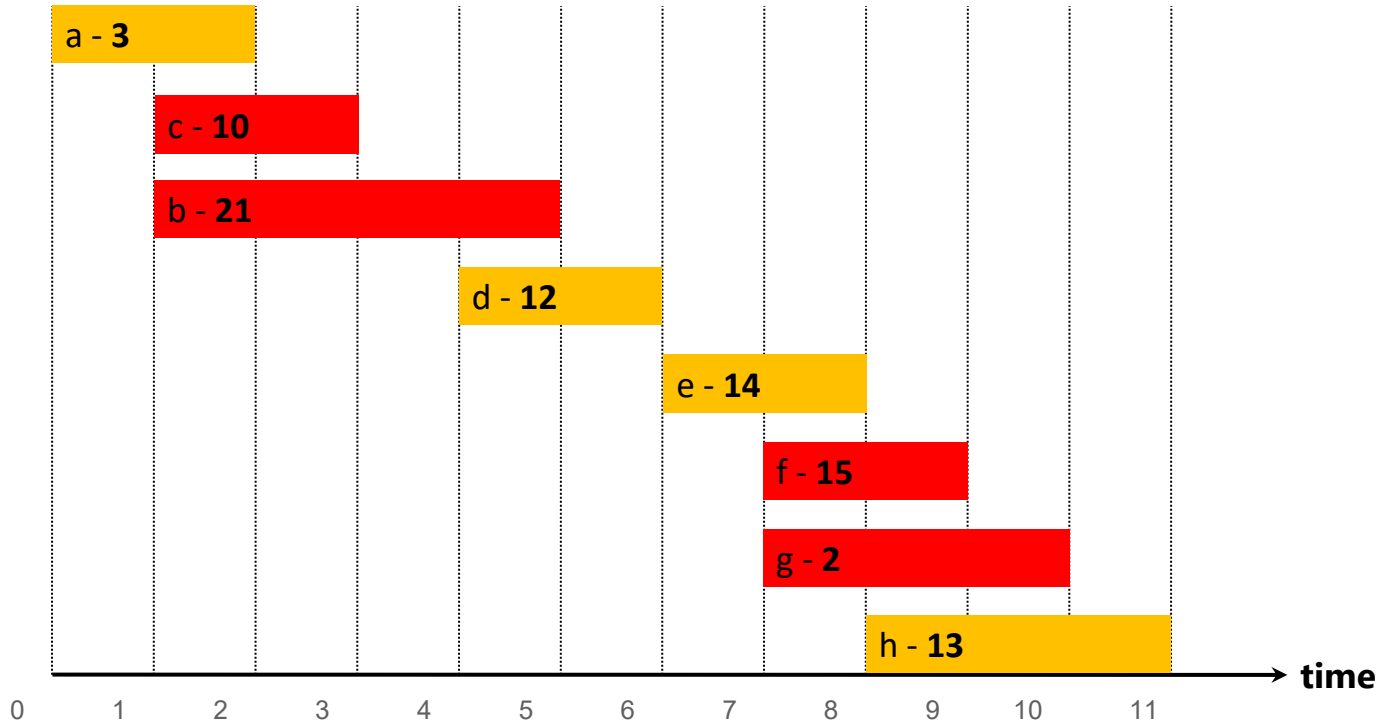


Weighted Interval Scheduling Problem



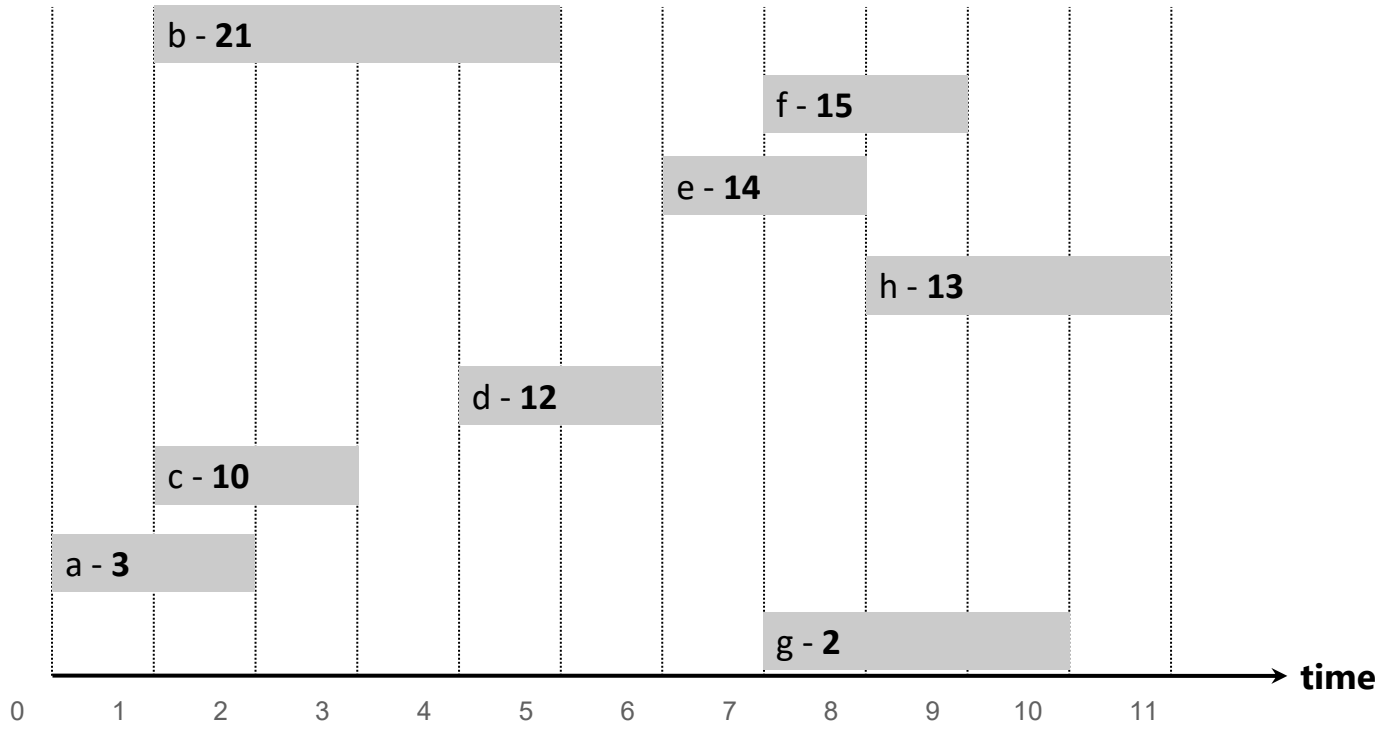
Weighted Interval Scheduling Problem

- Let try greedy approach w.r.t **Weight**

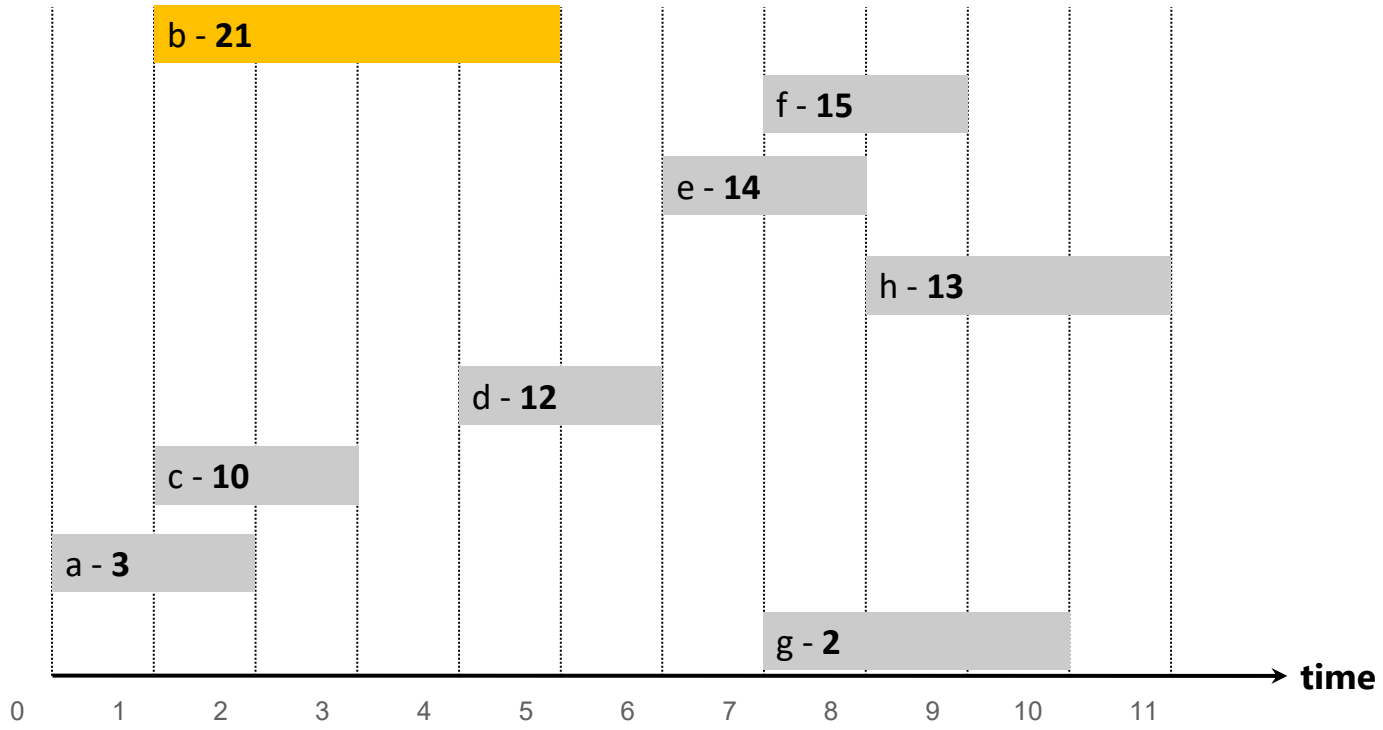


4 job with total weight of 42

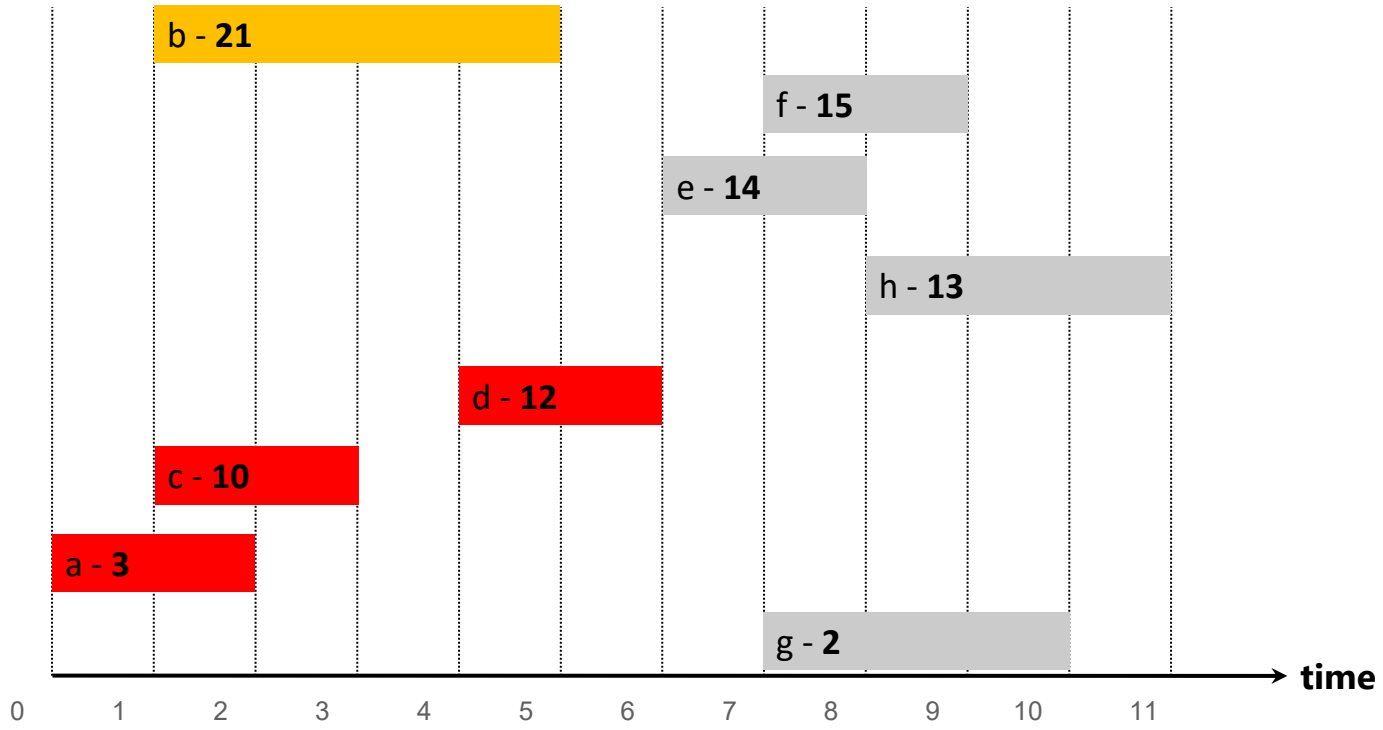
Weighted Interval Scheduling Problem



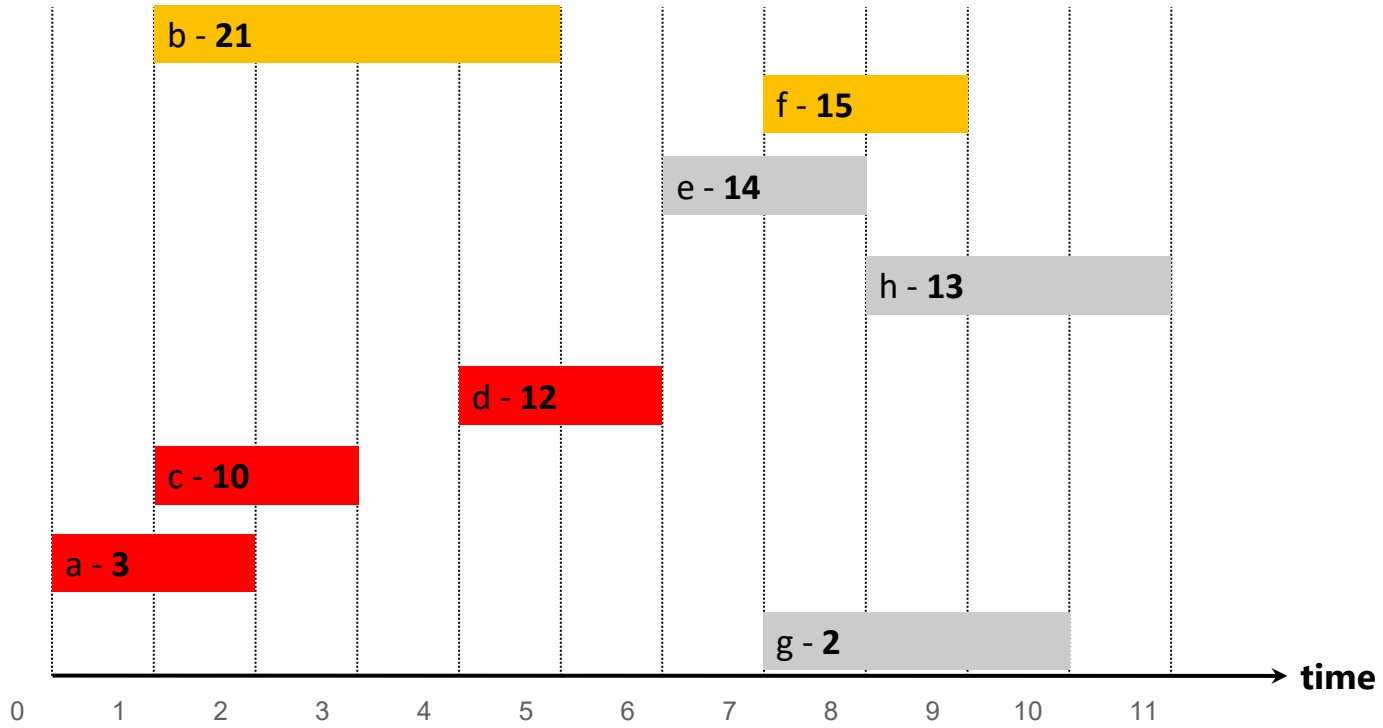
Weighted Interval Scheduling Problem



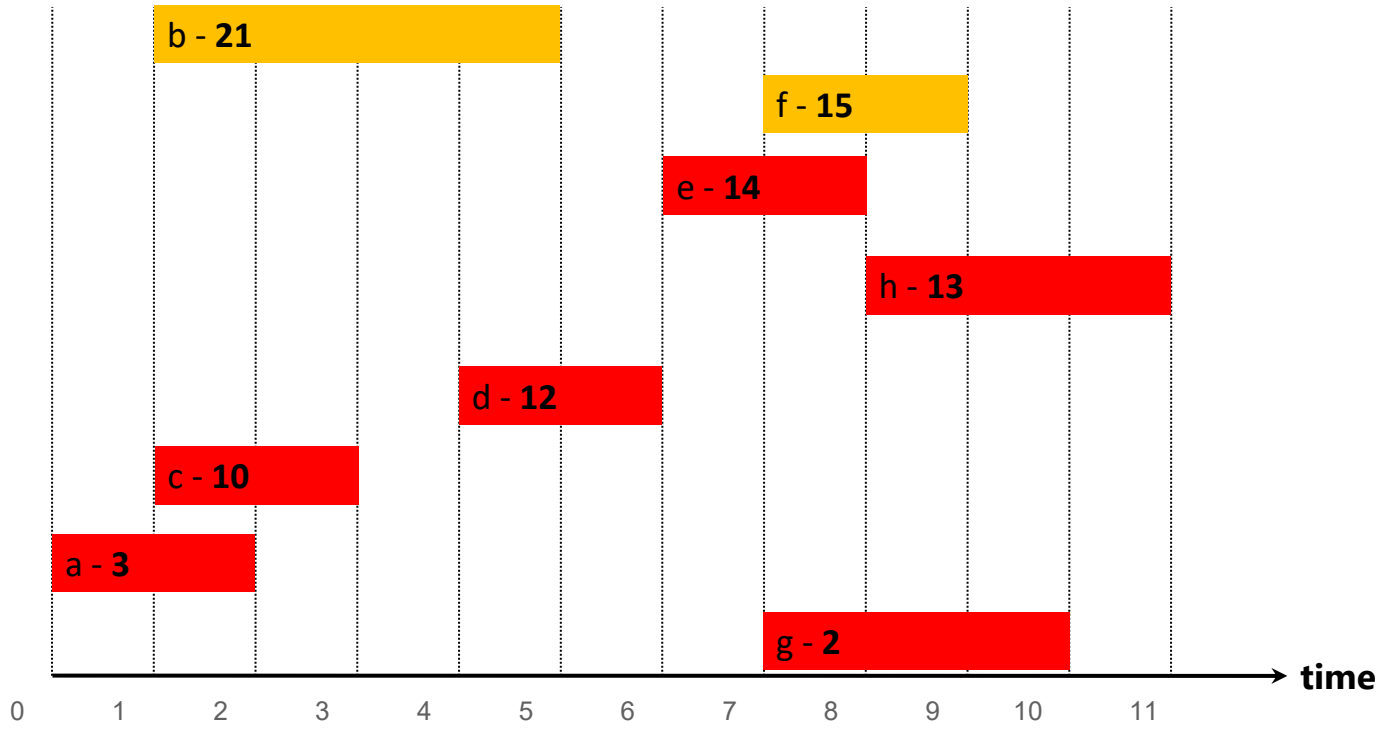
Weighted Interval Scheduling Problem



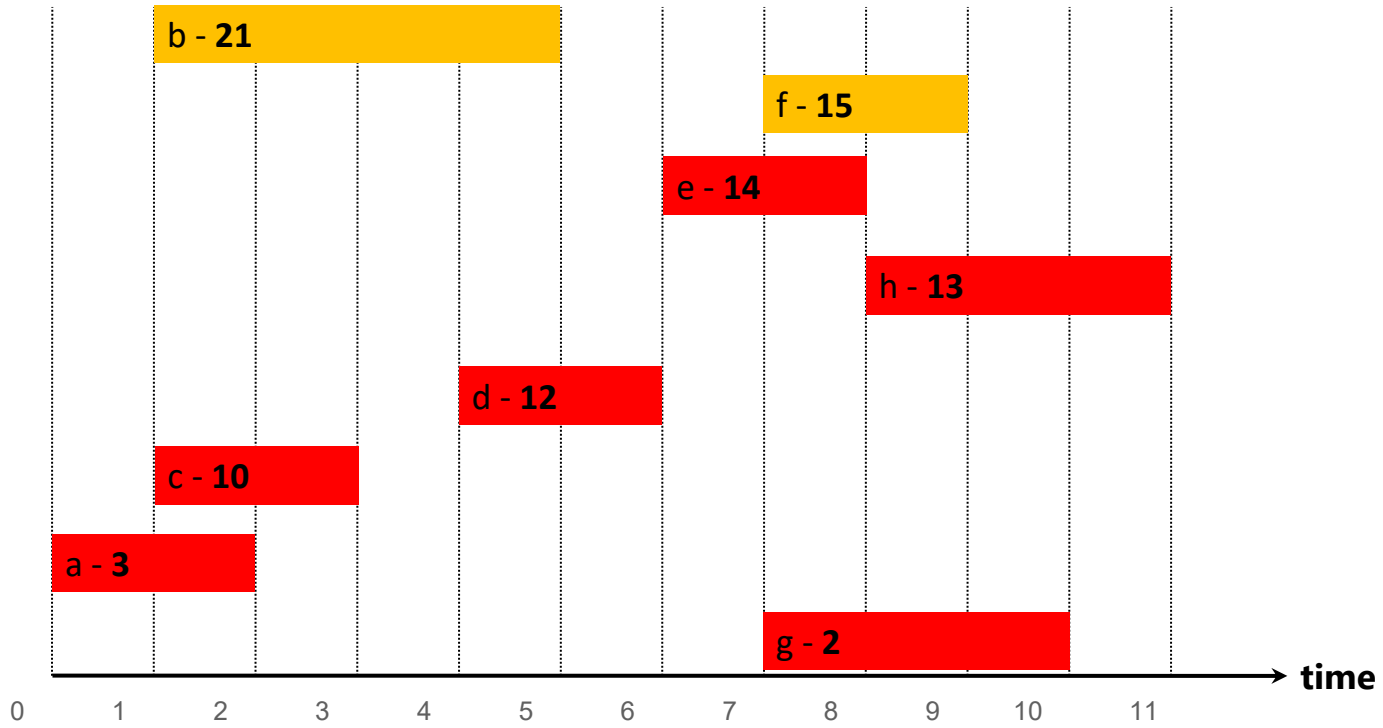
Weighted Interval Scheduling Problem



Weighted Interval Scheduling Problem

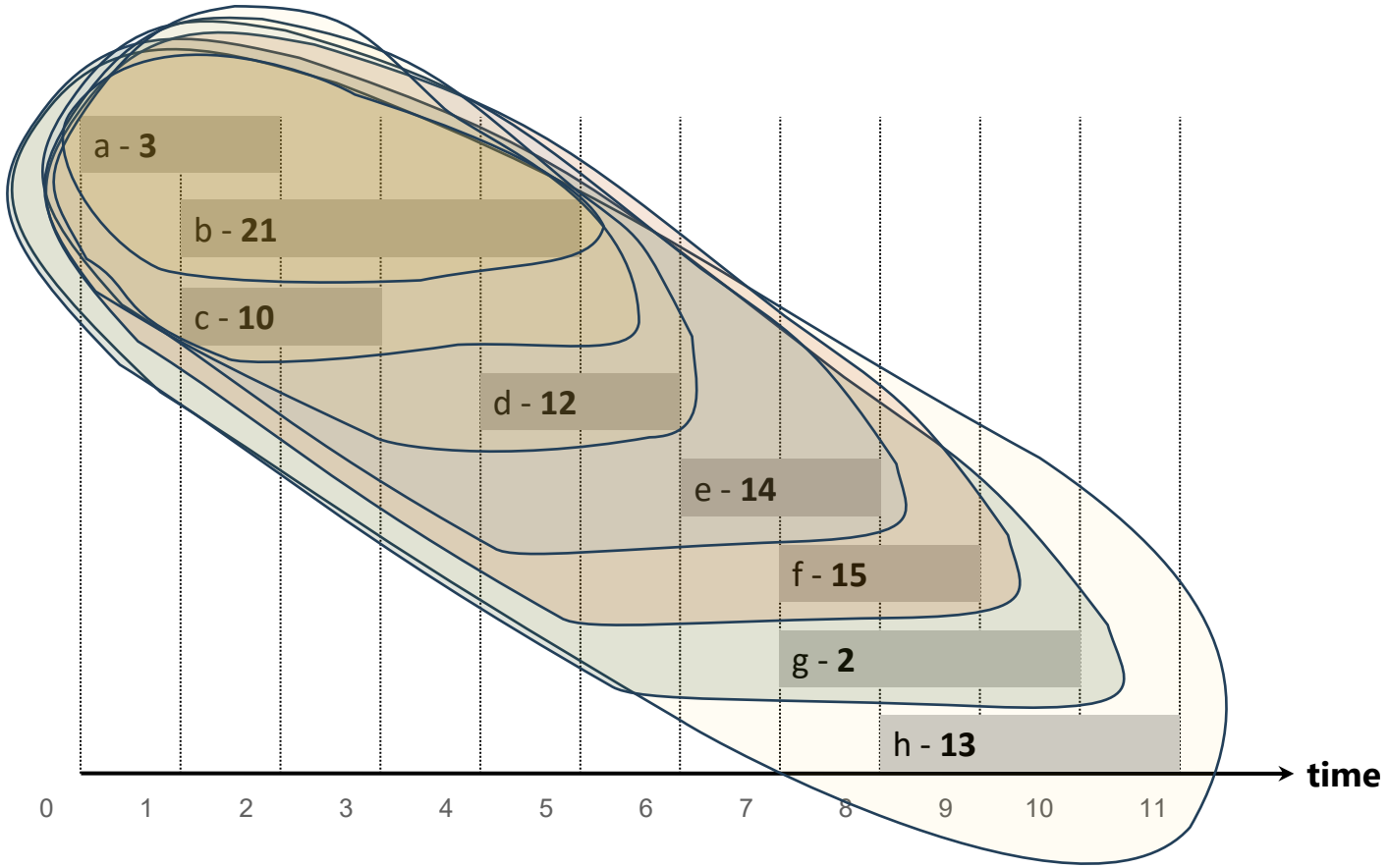


Weighted Interval Scheduling Problem



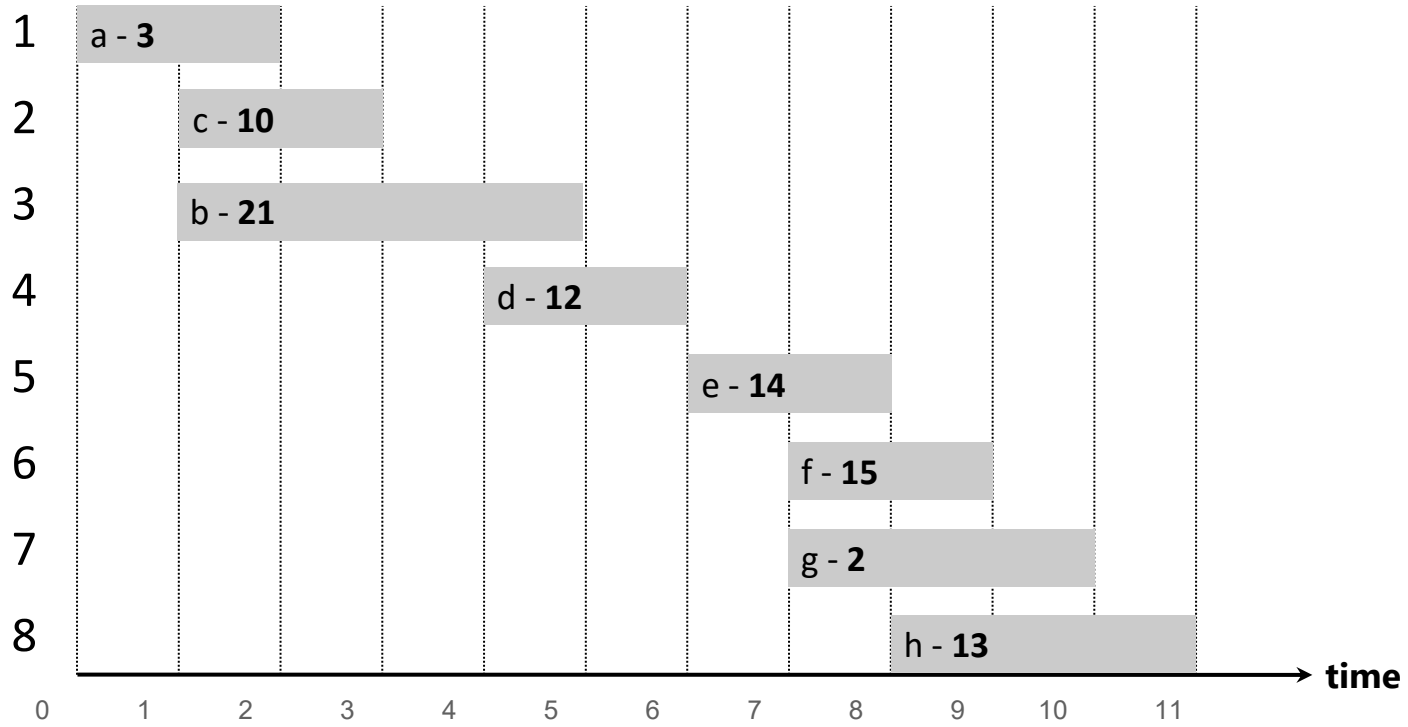
2 job with total weight of 36

Weighted Interval Scheduling Problem

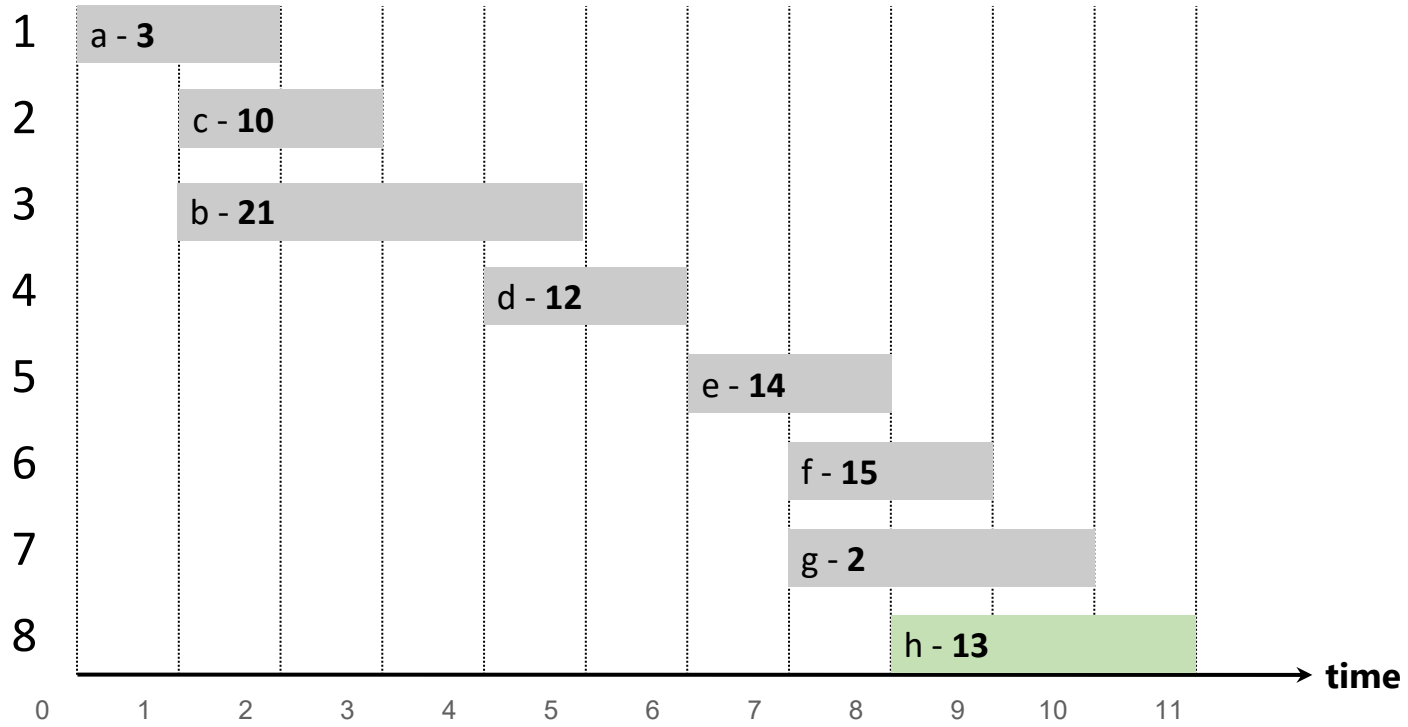


Weighted Interval Scheduling Problem

Our goal is to find **`Max_Value(n)`**

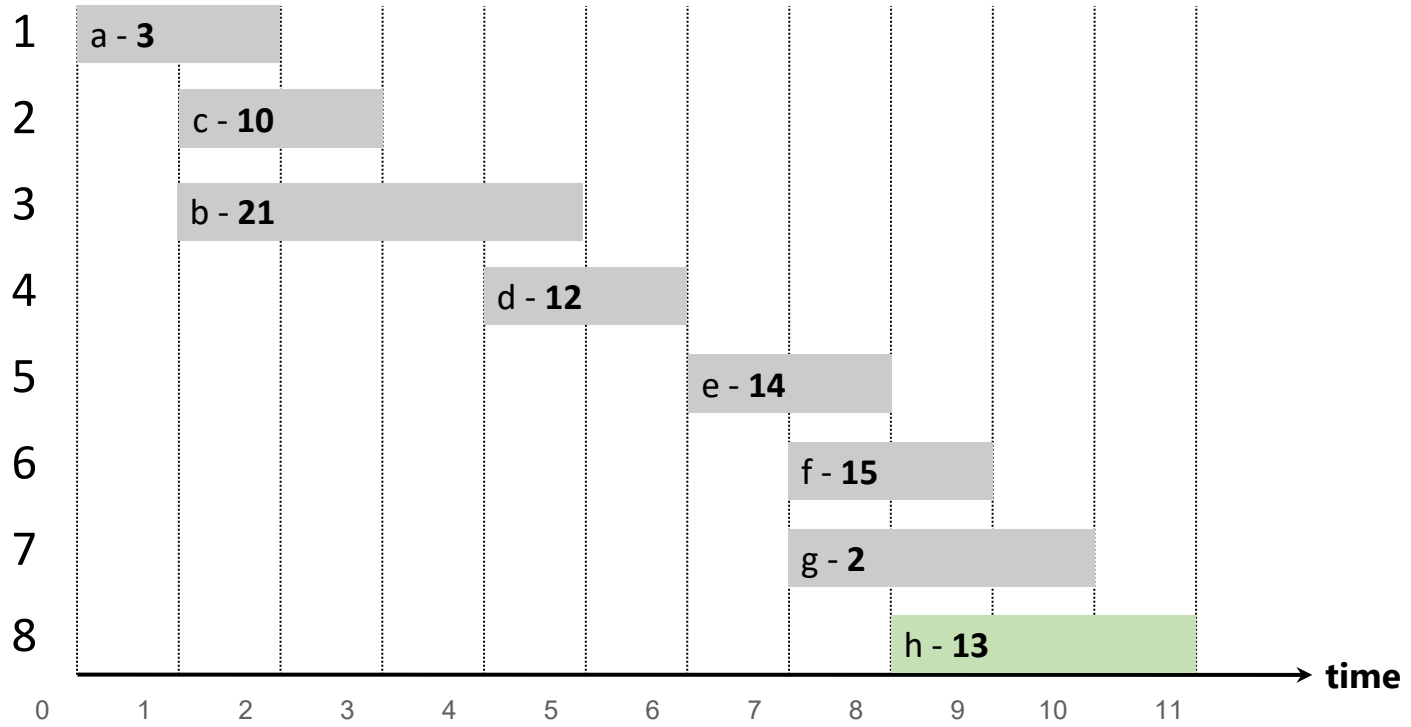


Weighted Interval Scheduling Problem



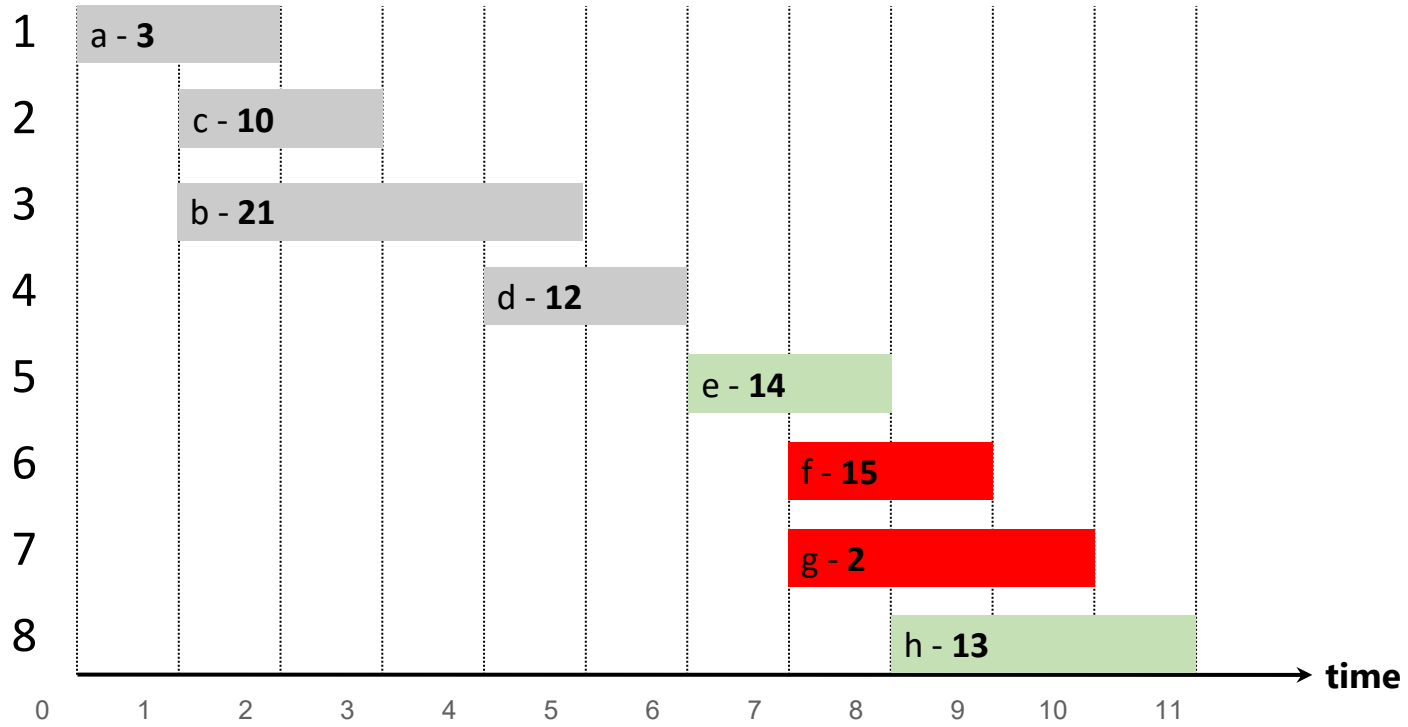
Max_Value$(8)= W_8+$

Weighted Interval Scheduling Problem



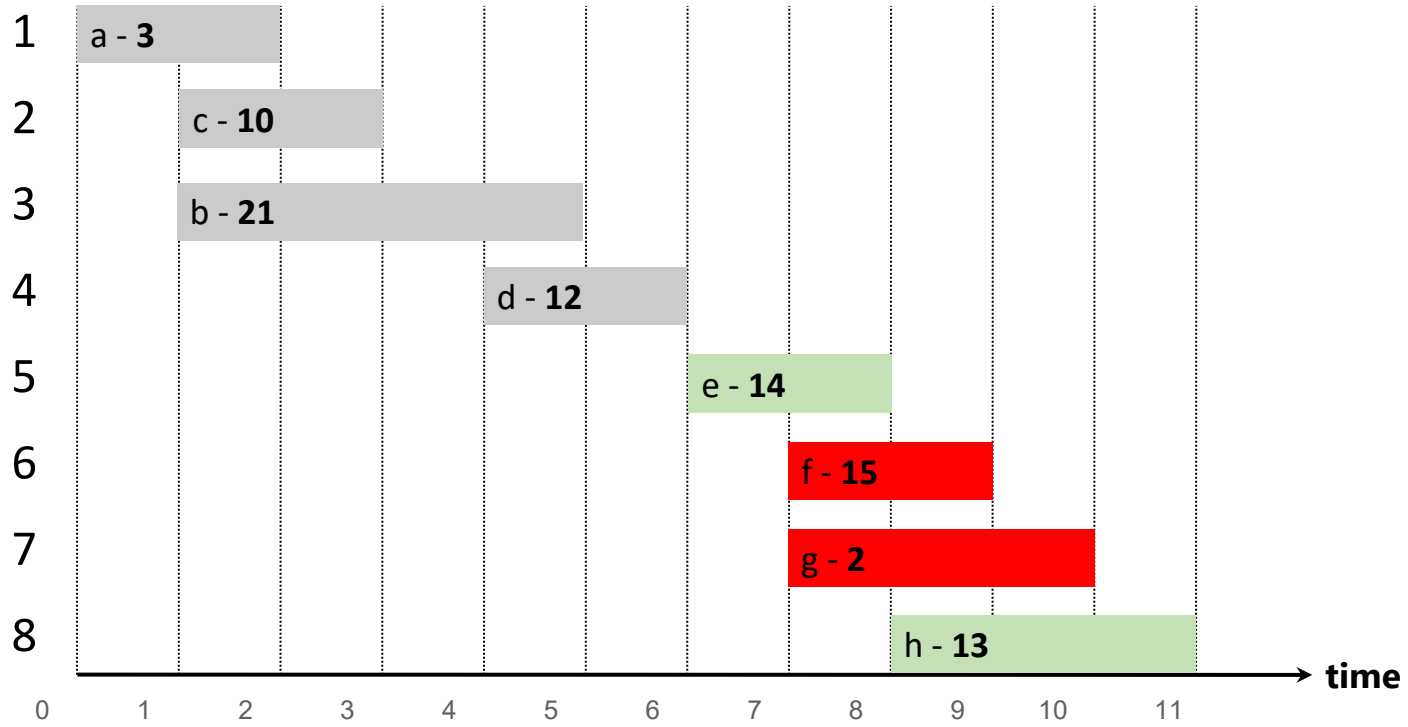
`Max_Value`(8)= W_8+ (Weight of last non overlapping job *w.r.t* 8)

Weighted Interval Scheduling Problem



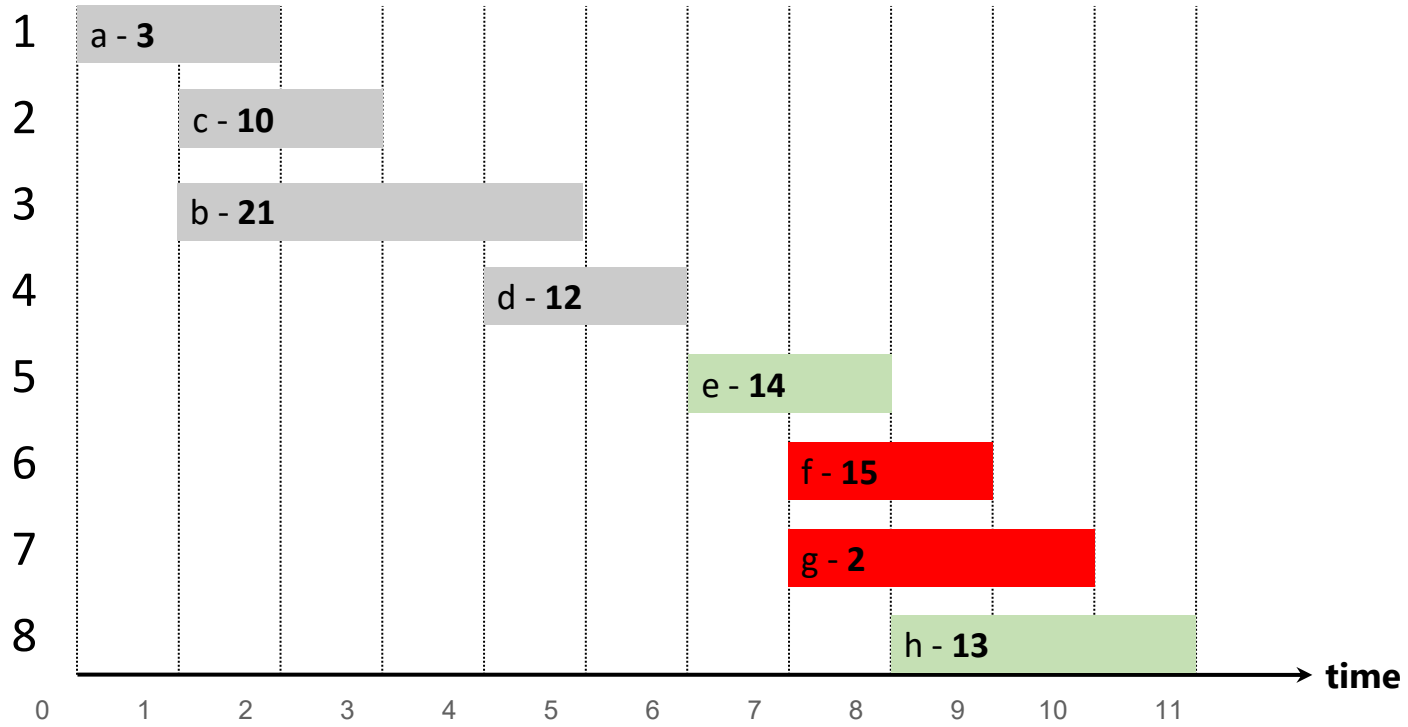
Max_Value$(8)= W_8 + (W_5 +)$

Weighted Interval Scheduling Problem



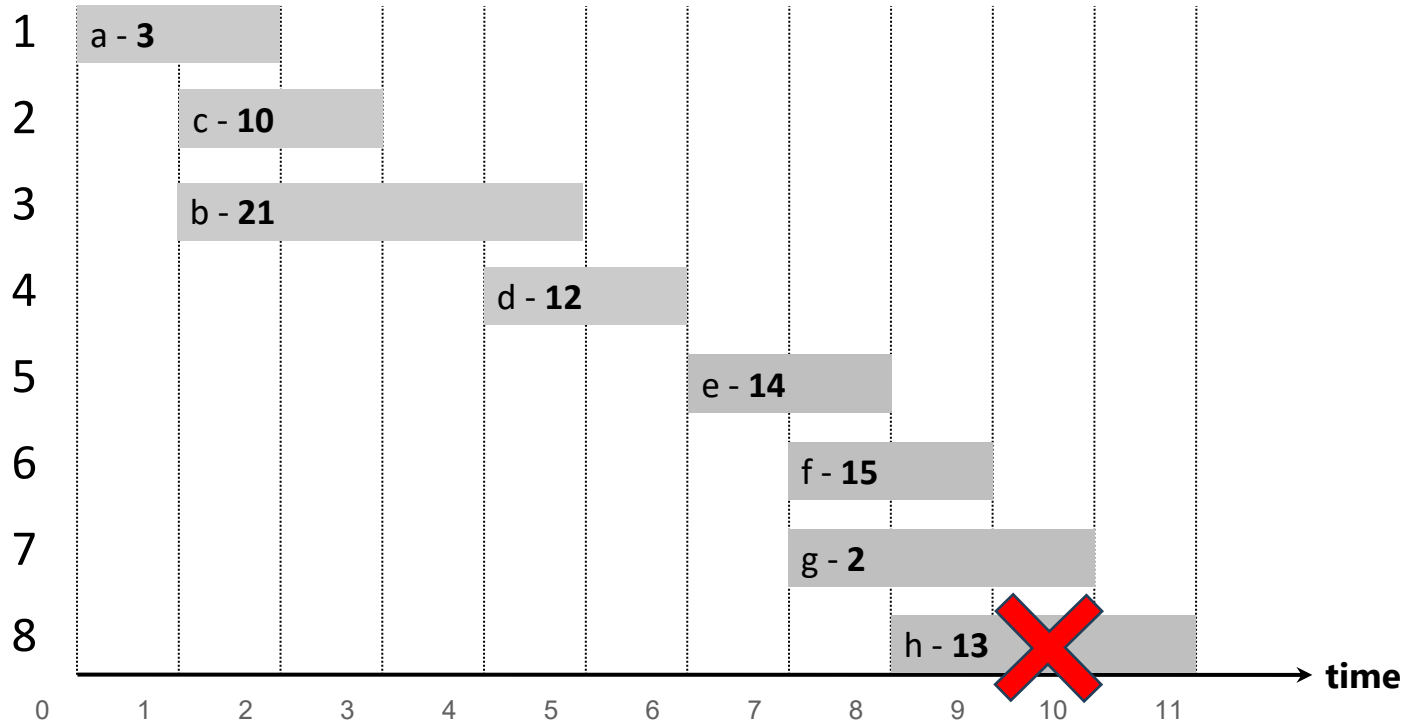
`Max_Value`(8)= W_8+ (W_5 + (Weight of last non overlapping job *w.r.t* 5))

Weighted Interval Scheduling Problem



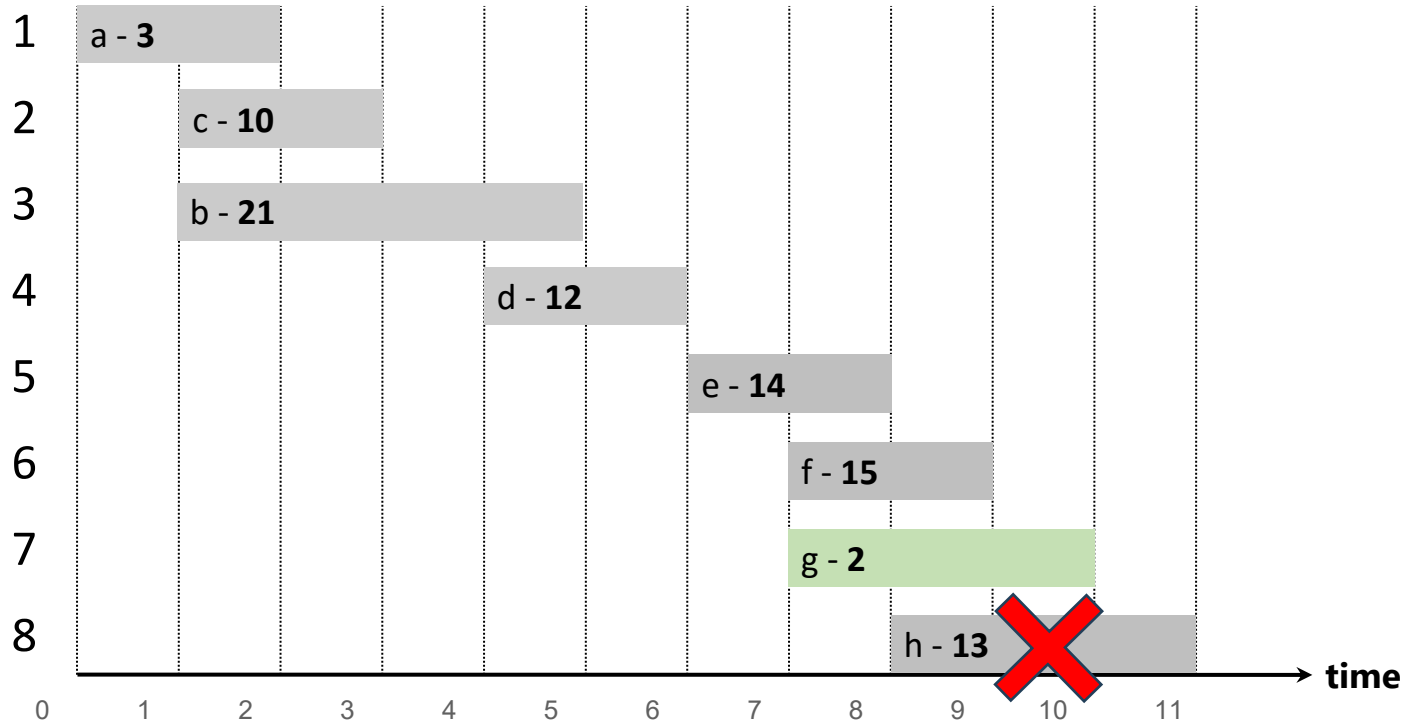
`Max_Value`(n)= W_n + **`Max_Value`**(Last non overlapping job *w.r.t* ***n***)

Weighted Interval Scheduling Problem



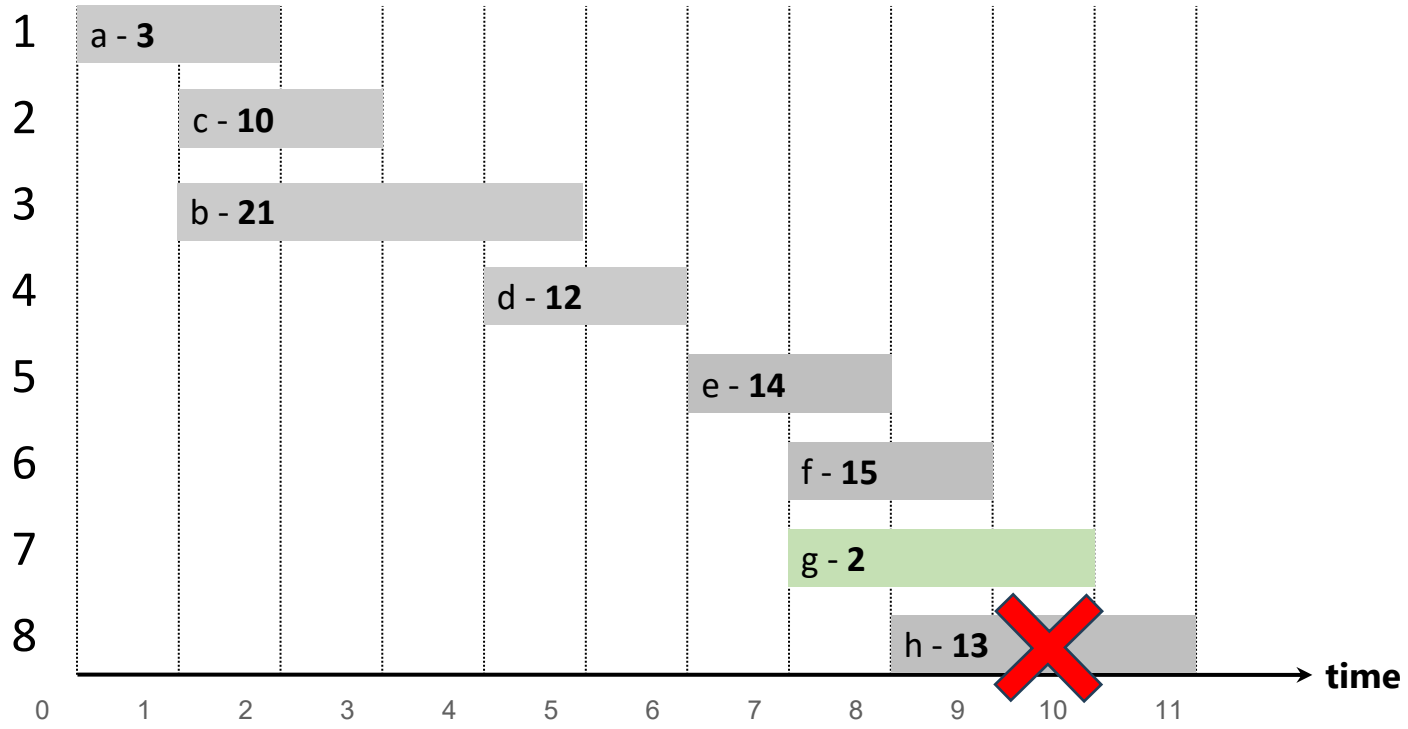
Max_Value(n)= W_n + **Max_Value**(Last non overlapping job *w.r.t* n)

Weighted Interval Scheduling Problem



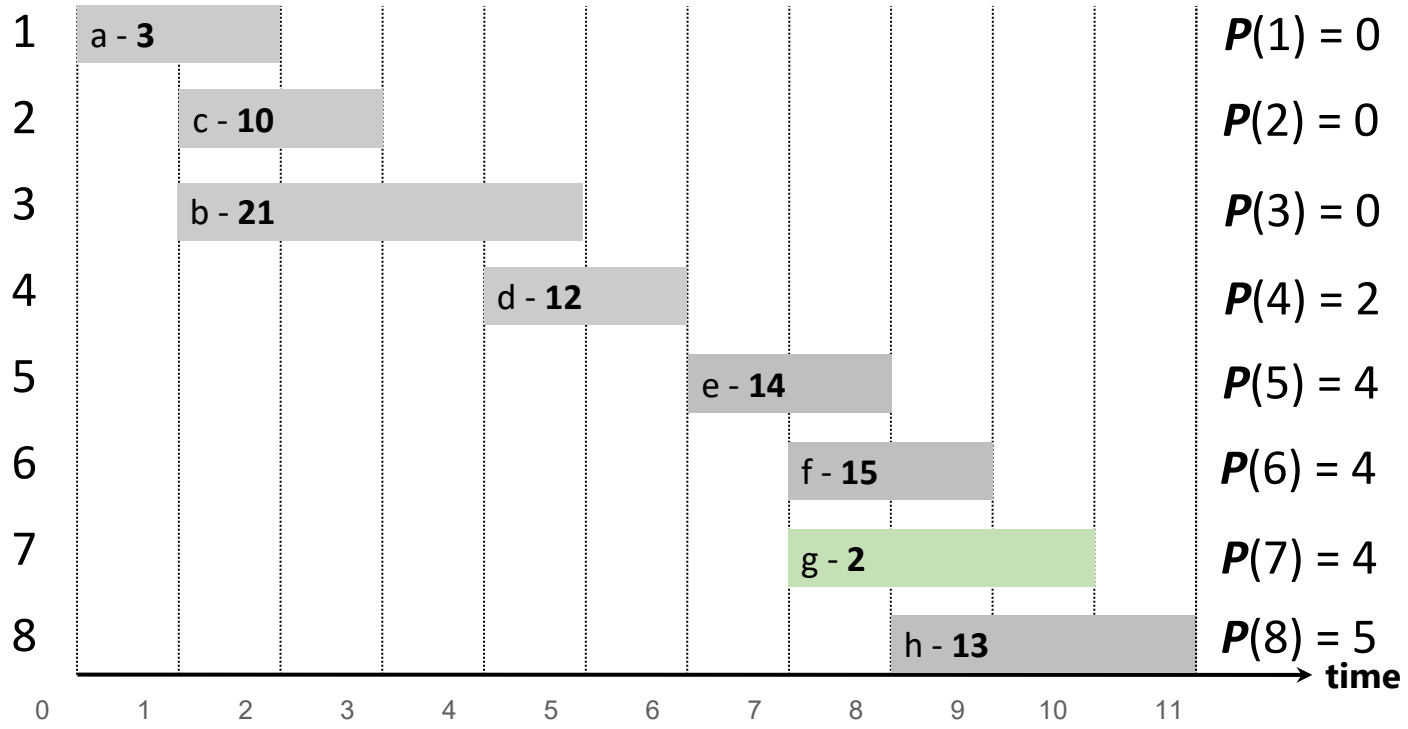
Max_Value(n)= W_n + **Max_Value**(Last non overlapping job *w.r.t* n)

Weighted Interval Scheduling Problem



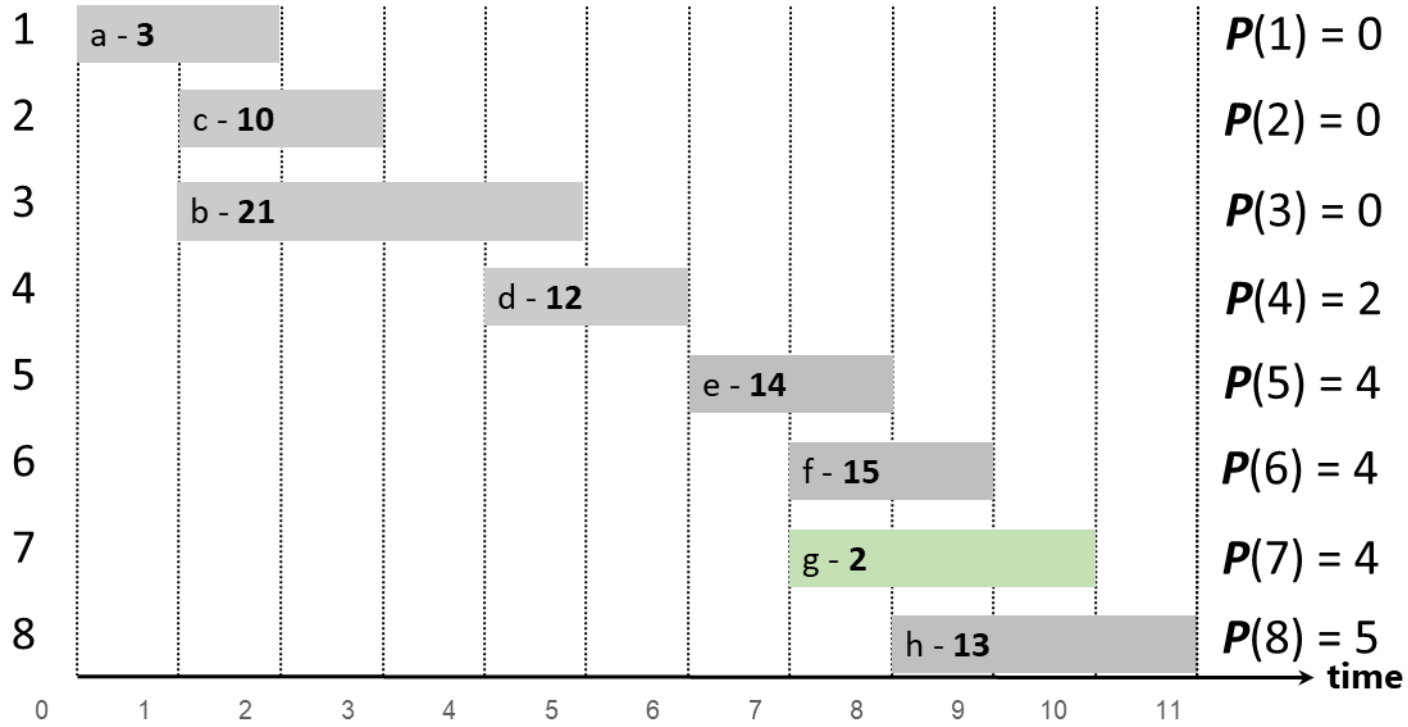
`Max_Value`(n)= ***max*** (W_n + **`Max_Value`**(Last non overlapping job *w.r.t* n), **`Max_Value`**(n-1))

Weighted Interval Scheduling Problem



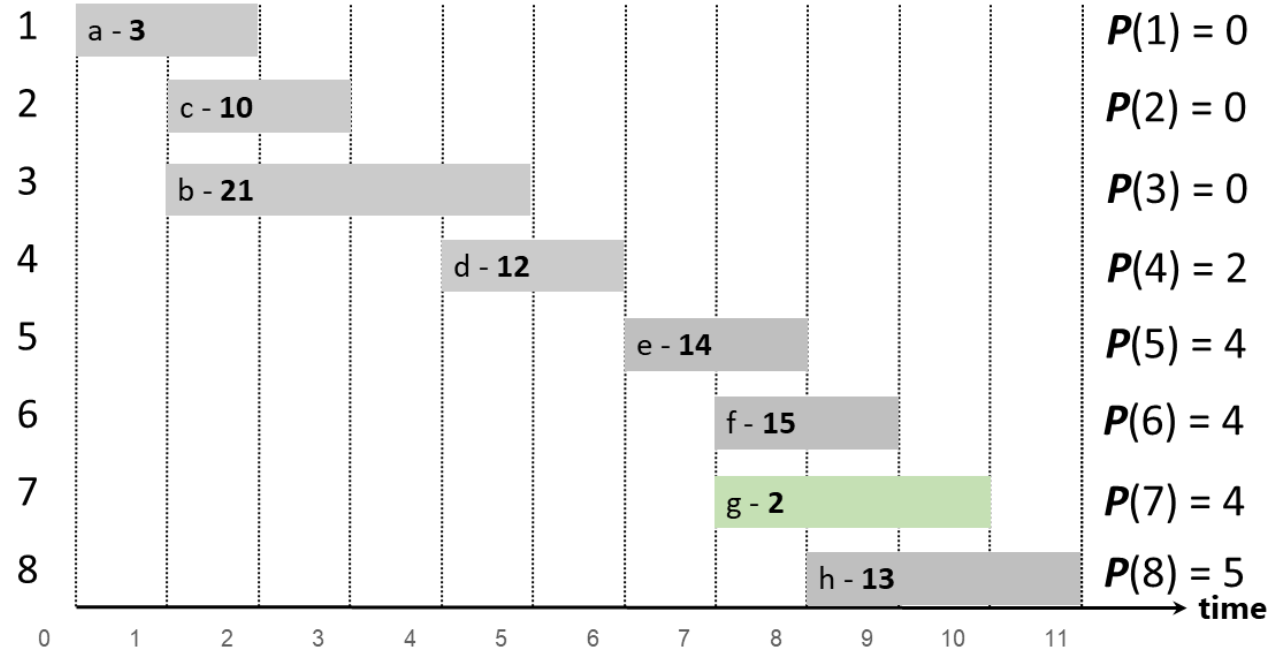
Max_Value(n)= ***max*** (W_n + **Max_Value**(Last non overlapping job *w.r.t* n), **Max_Value**(n-1))

Weighted Interval Scheduling Problem



Max_Value(n)= ***max*** (W_n + **Max_Value**(P(n)), **Max_Value**(n-1))

Weighted Interval Scheduling Problem



$$\textbf{Max\_Value}(n) = \max \begin{cases} 0 & \text{if } n=0 \\ W_n + \textbf{Max\_Value}(P(n)) & \text{if } W_n \in \text{Optimal Jobs} \\ \textbf{Max\_Value}(n-1) & \text{if } W_n \notin \text{Optimal Jobs} \end{cases}$$

Weighted Interval Scheduling Problem

Brute-force Algorithm

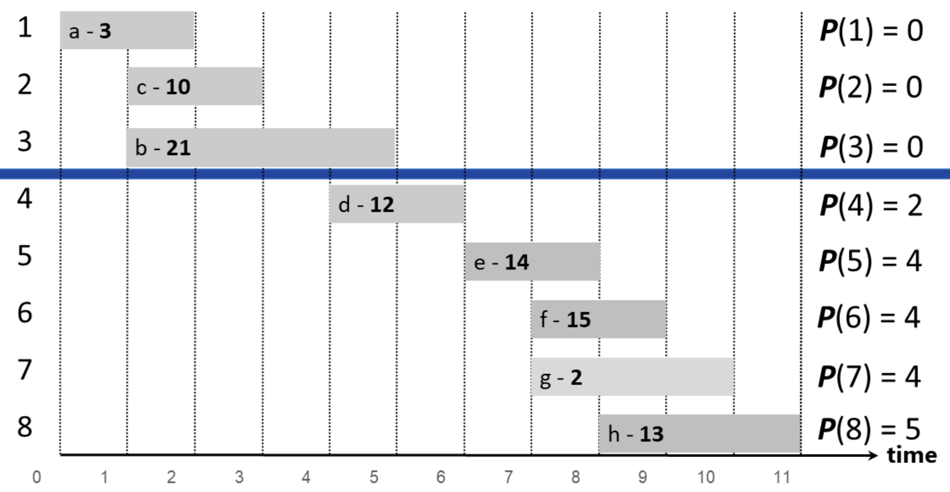
```
Input: n, s1,…,sn , f1,…,fn , w1,…,wn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

Compute p(1), p(2), …, p(n)

Max_Value(n) {
   if (n = 0)
      return 0
   else
      return max(Wn + Max_Value(p(n)), Max_Value(n-1))
}
```

O(nlogn)

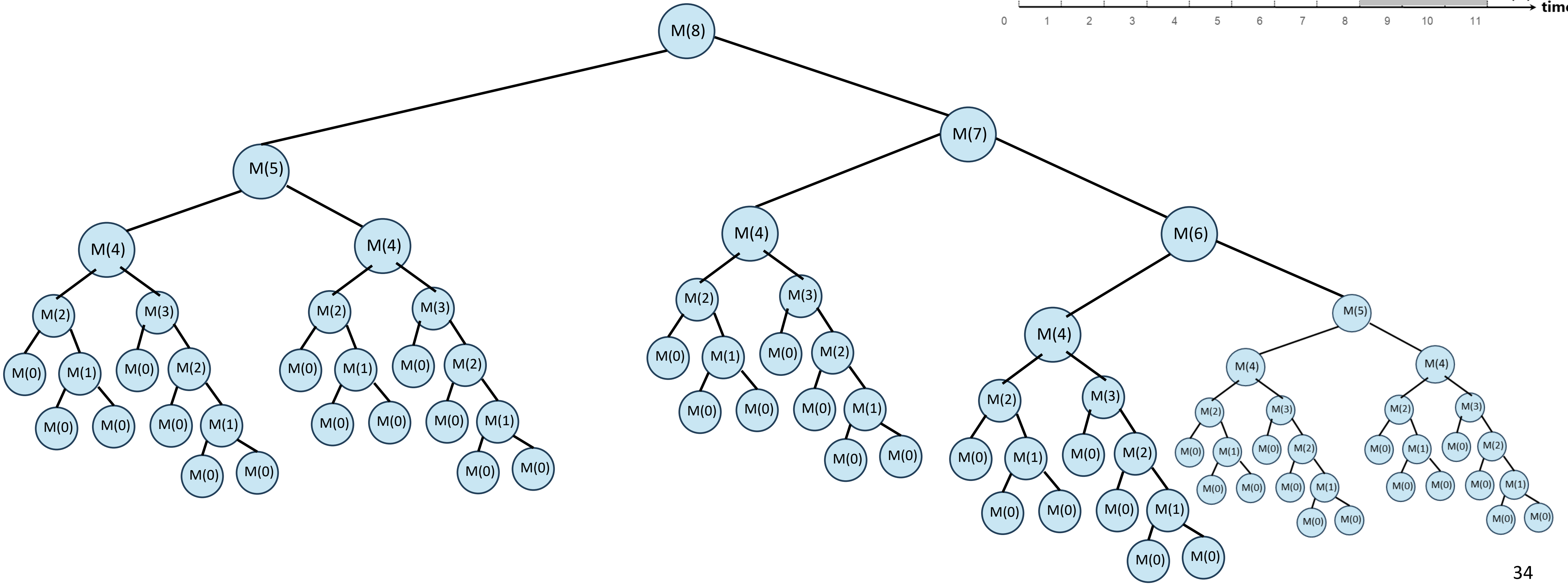
O(nlogn)

O(2^n)

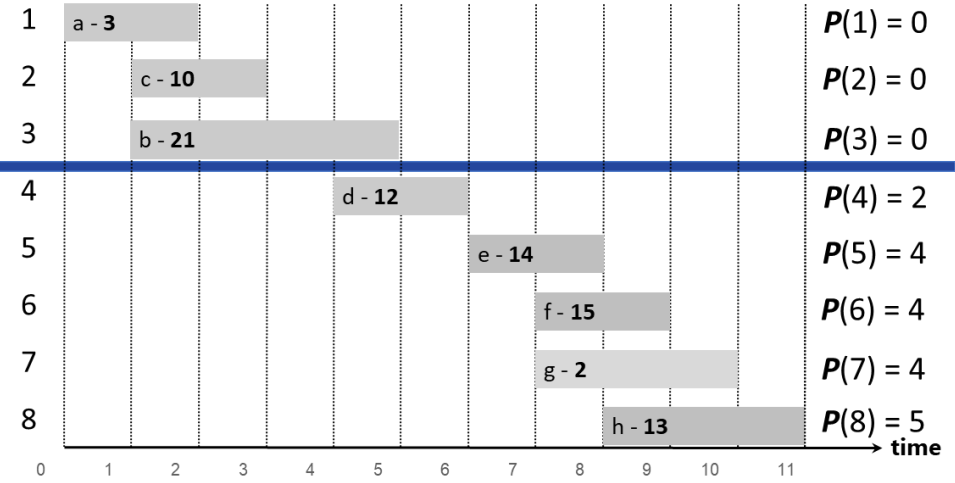
Weighted Interval Scheduling Problem



$$\texttt{Max\_Value}(n) = \mathbf{max} \begin{cases} 0 & \text{if } n=0 \\ W_n + \texttt{Max\_Value}(P(n)) & \text{if } W_n \in \text{Optimal Jobs} \\ \texttt{Max\_Value}(n-1) & \text{if } W_n \notin \text{Optimal Jobs} \end{cases}$$



Weighted Interval Scheduling Problem



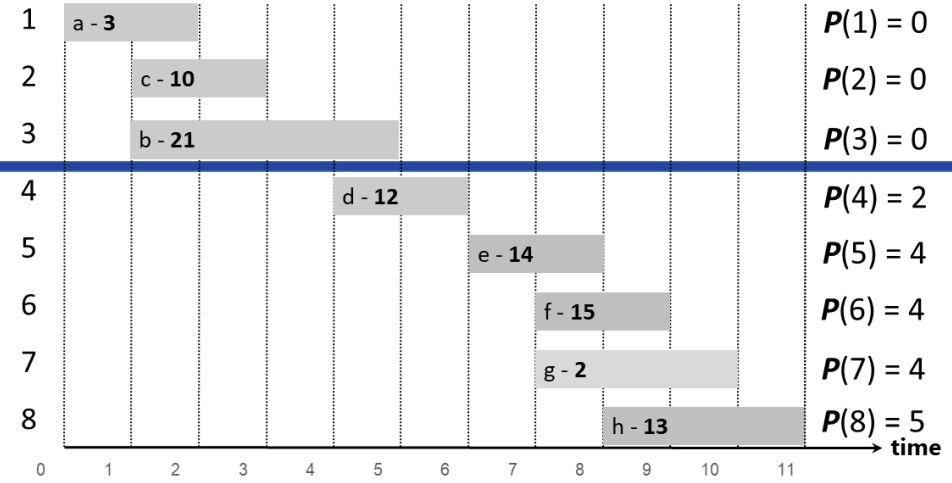
Memoization: Store results of each sub-problem in a cache; lookup as needed.

```
Input: n, s1,…,sn , f1,…,fn , w1,…,wn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.
Compute p(1), p(2), …, p(n)

for j = 1 to n
   M[j] = -1      ← global array
M[0] = 0

Max_Value(n) {
   if (M[n] is -1)
      M[n] = max(wn + Max_Value(p(n)), Max_Value(n-1))
   return M[j]
}
```

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

0
-1
-1
-1
-1
-1
-1
-1
-1

Weighted Interval Scheduling Problem



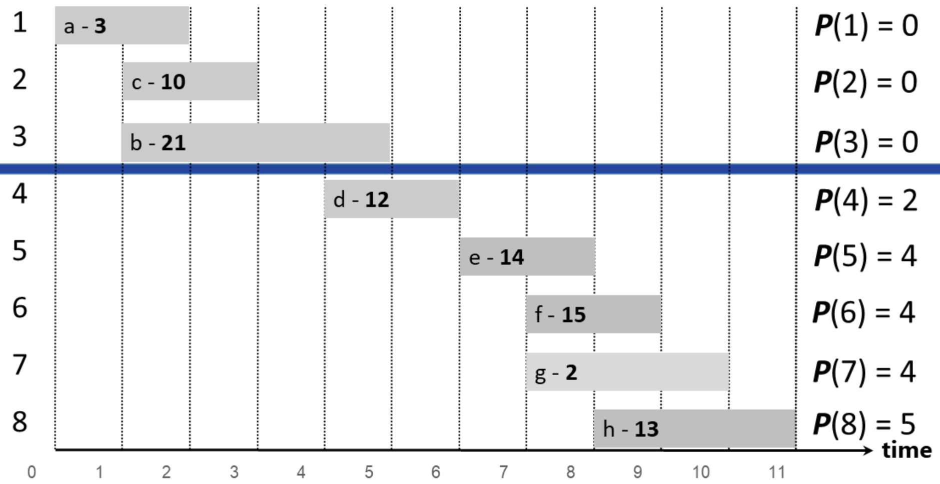
Memoization: Store results of each sub-problem in a cache; lookup as needed.

```
Input: n, s1,…,sn , f1,…,fn , w1,…,wn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.
Compute p(1), p(2), …, p(n)

for j = 1 to n
   M[j] = -1      ← global array
M[0] = 0

Max_Value(n) {
   if (M[n] is -1)
      M[n] = max(wn + Max_Value(p(n)), Max_Value(n-1))
   return M[j]
}
```

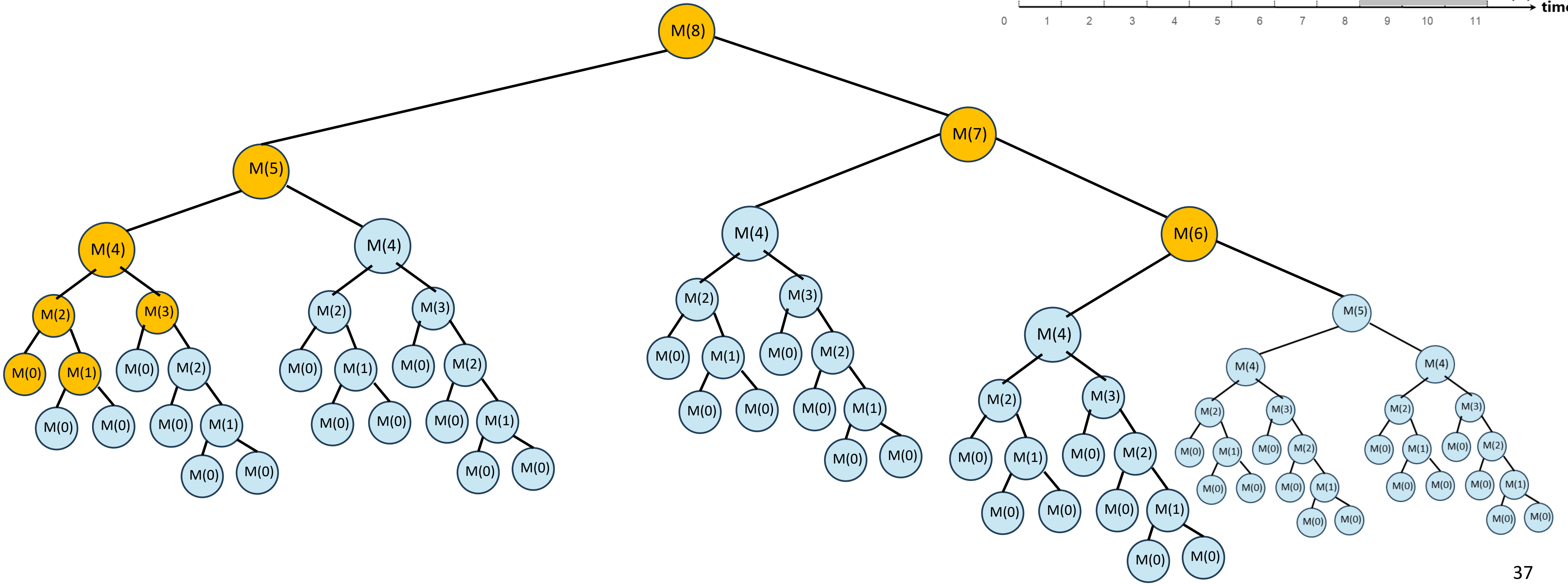
M[ ]	0
	1
	2
	3
	4
	5
	6
	7
	8

0
3
10
21
22
36
37
37
49

Weighted Interval Scheduling Problem



$$\textbf{Max\_Value}(n) = \textbf{\textit{max}} \begin{cases} 0 & \text{if } n=0 \\ W_n + \textbf{Max\_Value}(P(n)) & \text{if } W_n \in \text{Optimal Jobs} \\ \textbf{Max\_Value}(n-1) & \text{if } W_n \notin \text{Optimal Jobs} \end{cases}$$



Weighted Interval Scheduling Problem

Overall Time Complexity

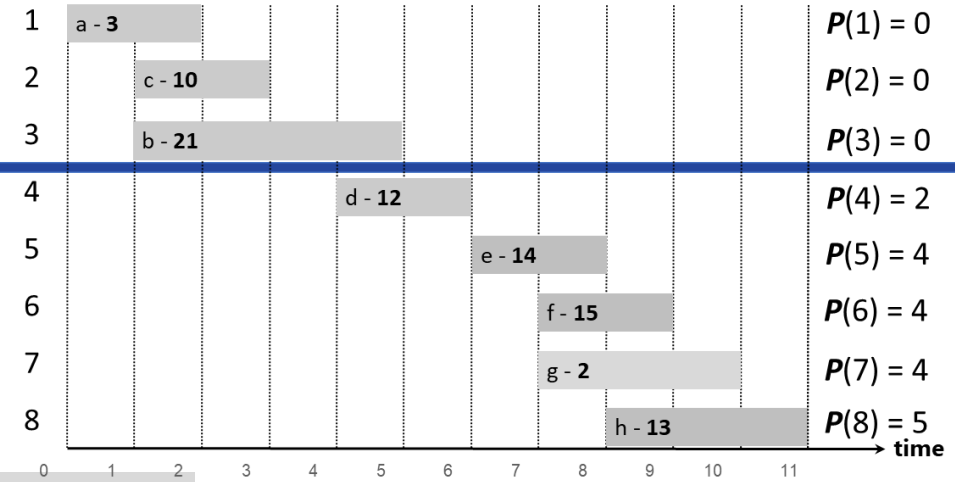
```
Input: n, s1,…,sn , f1,…,fn , w1,…,wn

Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.
Compute p(1), p(2), …, p(n)

for j = 1 to n
   M[j] = empty ← global array
M[0] = 0

Max_Value(n) {
   if (M[n] is empty)
      M[n] = max(wn + Max_Value(p(n)), Max_Value(n-1))
   return M[j]
}
```

O(nlogn)
O(nlogn)

O(n)

O(n)

WISP: Finding a Solution



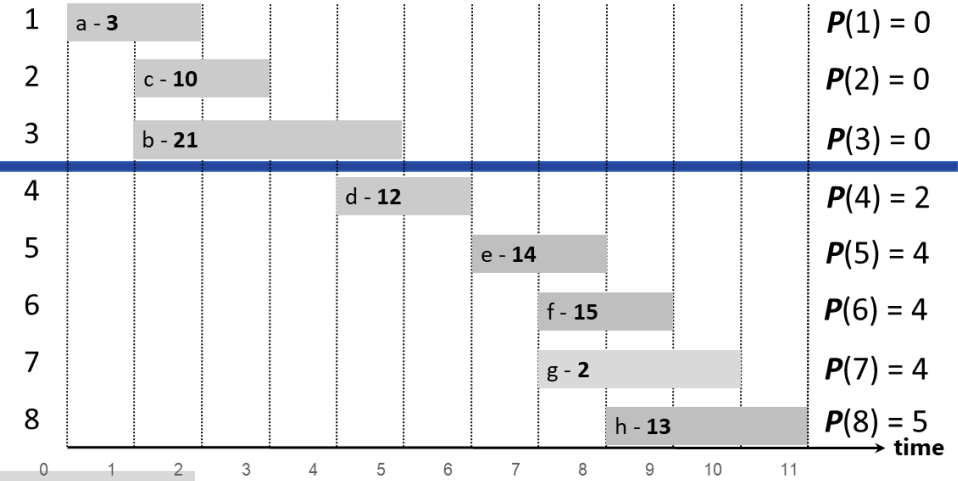
Q: Dynamic programming algorithms computes optimal value. How can we get the solution itself?

A: Do some post-processing.

```
Run Max_Value(n)
Run Find-Solution(n)

Find-Solution(j) {
    if (j = 0)
        output nothing
    else if (w_j + M[p(j)] > M[j-1])
        print j
        Find-Solution(p(j))
    else
        Find-Solution(j-1)
}
```

0	0
1	3
2	10
3	21
4	22
5	36
6	37
7	37
8	49

WISP: Finding a Solution



Q: Dynamic programming algorithms computes optimal value. How can we get the solution itself?

A: Do some post-processing.

```
Run Max_Value(n)
Run Find-Solution(n)

Find-Solution(j) {
    if (j = 0)
        output nothing
    else if (w_j + M[p(j)] > M[j-1])
        print j
        Find-Solution(p(j))
    else
        Find-Solution(j-1)
}
```

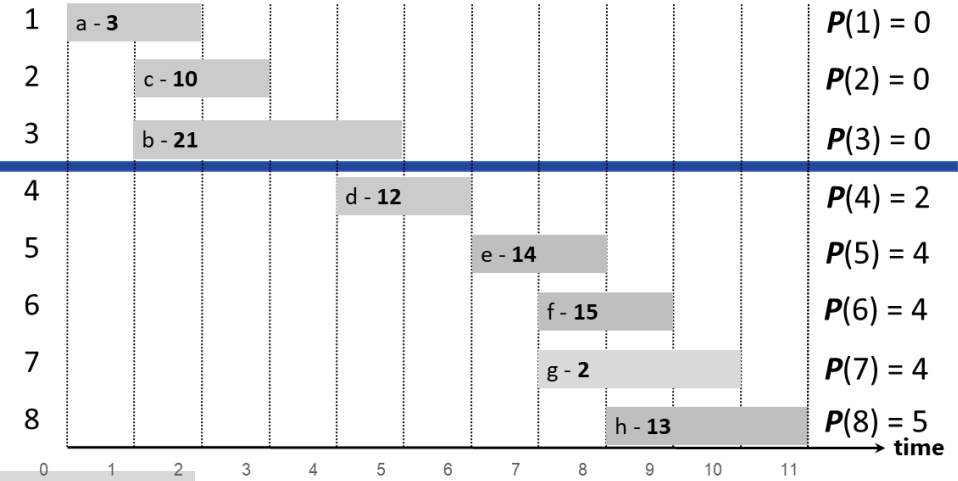
j = 8

13 + 36 > 37

8

0	0
1	3
2	10
3	21
4	22
5	36
6	37
7	37
8	49

WISP: Finding a Solution



Q: Dynamic programming algorithms computes optimal value. How can we get the solution itself?

A: Do some post-processing.

```
Run Max_Value(n)
Run Find-Solution(n)

Find-Solution(j) {
    if (j = 0)
        output nothing
    else if (w_j + M[p(j)] > M[j-1])
        print j
        Find-Solution(p(j))
    else
        Find-Solution(j-1)
}
```

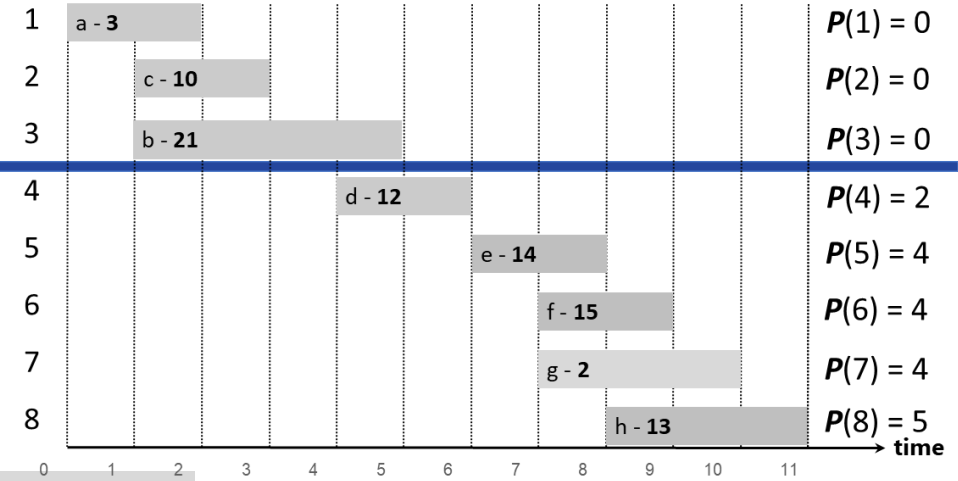
j = 5

14 + 22 > 22

0	0
1	3
2	10
3	21
4	22
5	36
6	37
7	37
8	49

8 5

WISP : Finding a Solution



Q: Dynamic programming algorithms computes optimal value. How can we get the solution itself?

A: Do some post-processing.

```
Run Max_Value(n)
Run Find-Solution(n)

Find-Solution(j) {
    if (j = 0)
        output nothing
    else if (w_j + M[p(j)] > M[j-1])
        print j
        Find-Solution(p(j))
    else
        Find-Solution(j-1)
}
```

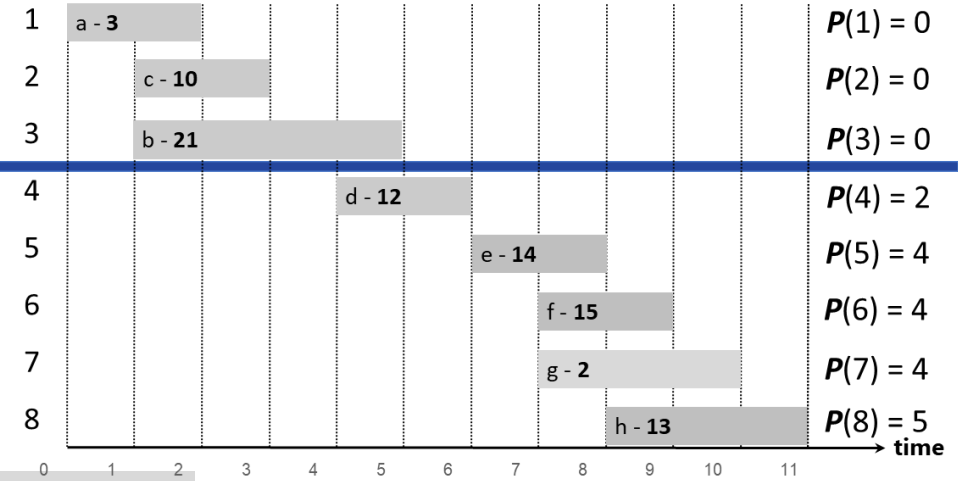
j = 4

12 + 10 > 21

0	0
1	3
2	10
3	21
4	22
5	36
6	37
7	37
8	49

8 5 4

WISP : Finding a Solution



Q: Dynamic programming algorithms computes optimal value. How can we get the solution itself?

A: Do some post-processing.

```
Run Max_Value(n)
Run Find-Solution(n)

Find-Solution(j) {
    if (j = 0)
        output nothing
    else if (w_j + M[p(j)] > M[j-1])
        print j
        Find-Solution(p(j))
    else
        Find-Solution(j-1)
}
```

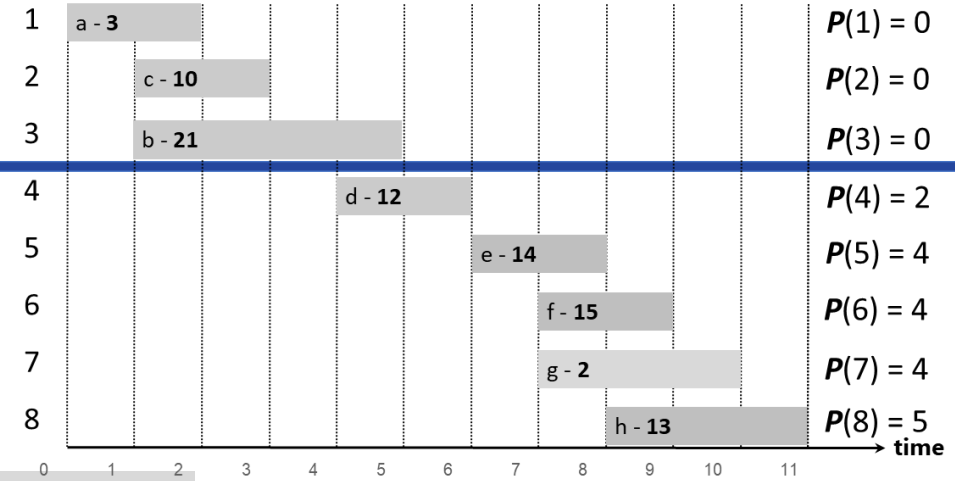
j = 2

10 + 0 > 3

8	5	4	2

0	0
1	3
2	10
3	21
4	22
5	36
6	37
7	37
8	49

WISP : Finding a Solution



Q: Dynamic programming algorithms computes optimal value. How can we get the solution itself?

A: Do some post-processing.

```
Run Max_Value(n)
Run Find-Solution(n)

Find-Solution(j) {
    if (j = 0)
        output nothing
    else if (w_j + M[p(j)] > M[j-1])
        print j
        Find-Solution(p(j))
    else
        Find-Solution(j-1)
}
```

j = 0

8	5	4	2

0	0
1	3
2	10
3	21
4	22
5	36
6	37
7	37
8	49

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

Thanks a lot

If you are taking a Nap, **wake up**.....***Lecture OVER***