

# Dynamic Programming

---

Algorithm Design 6.1, 6.2, 6.4

# Applications

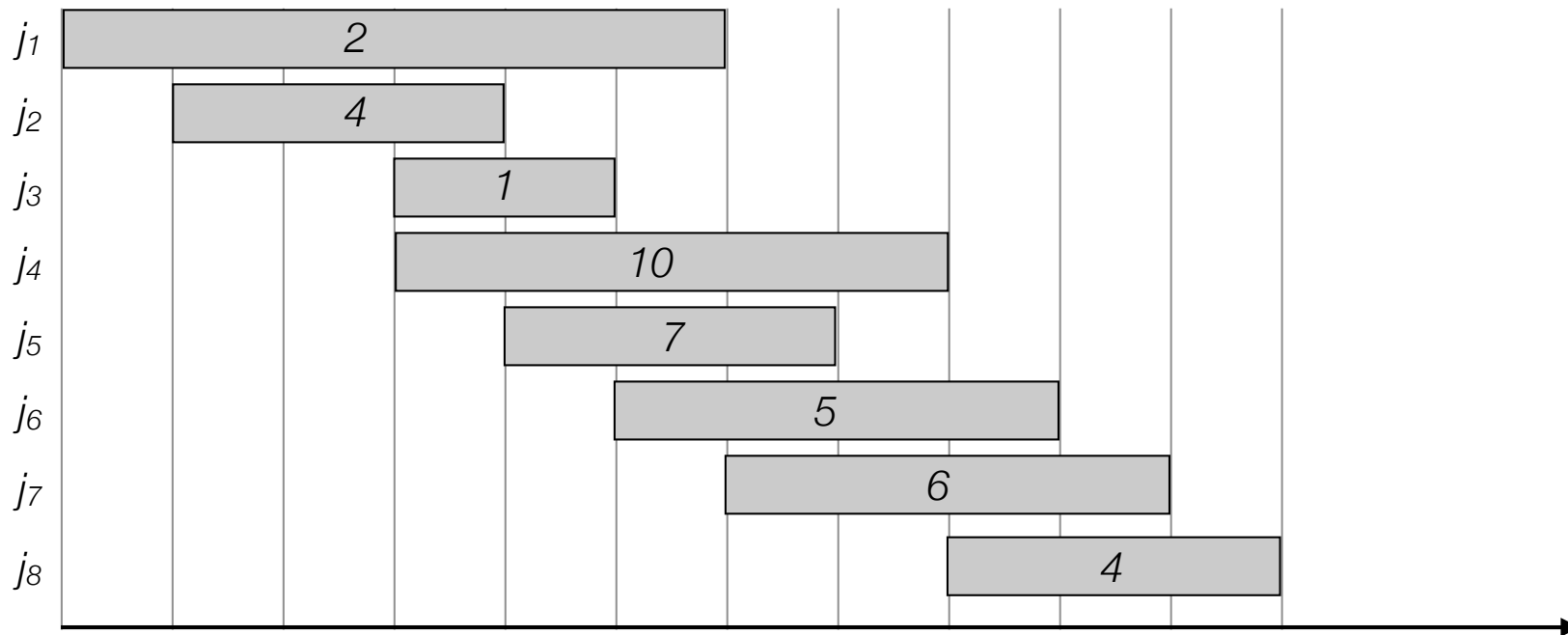
---

- In class (today and next time)

# Applications

---

- In class (today and next time)
  - Weighted interval scheduling
    - Set of weighted intervals with start and finishing times
    - Goal: find maximum weight subset of non-overlapping intervals



# Applications

---

- Today and next time
  - Weighted interval scheduling
  - Subset Sum and Knapsack
    - Set of items each having a weight and a value
    - Knapsack with a bounded capacity
    - Goal: fill knapsack so as to maximise the total value.



Capacity 8

value	10	8	2	5	15	4
						
weight	2	3	1	2	5	4

# Applications

---

- Today and next time
  - Weighted interval scheduling
  - Subset Sum and Knapsack
  - Sequence alignment
    - Given two strings A and B how many edits (insertions, deletions, relabelings) is needed to turn A into B?

A C A A G T C  
- C A T G T -

1 mismatch, 2 gaps

A C A A - G T C  
- C A - T G T -

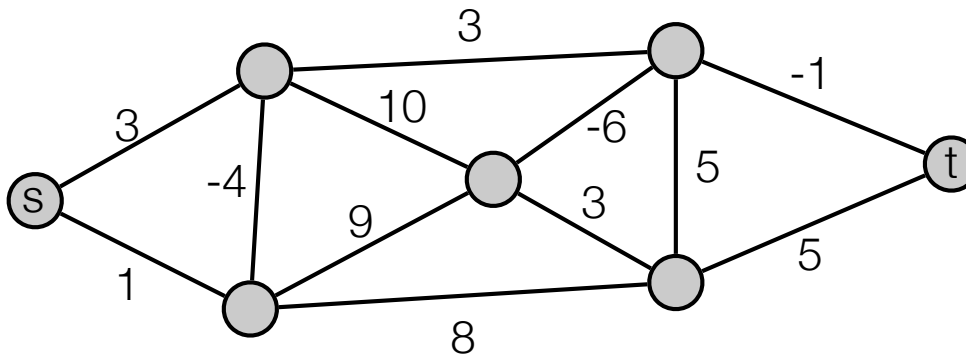
0 mismatches, 4 gaps

# Applications

---

- Today and next time

- Weighted interval scheduling
- Subset Sum and Knapsack
- Sequence alignment
- Shortest paths with negative weights
  - Given a weighted graph, where edge weights can be negative, find the shortest path between two given vertices.



# Applications

---

- Today and next time
  - Weighted interval scheduling
  - Subset Sum and Knapsack
  - Sequence alignment
  - Shortest paths with negative weights
- Some other famous applications
  - Unix diff for comparing 2 files
  - Vovke-Kasami-Younger for parsing context-free grammars
  - Viterbi for hidden Markov models
  - ....

# Dynamic Programming

---

- **Greedy.** Build solution incrementally, optimizing some local criterion.
- **Divide-and-conquer.** Break up problem into **independent** subproblems, solve each subproblem, and combine to get solution to original problem.
- **Dynamic programming.** Break up problem into **overlapping** subproblems, and build up solutions to larger and larger subproblems.
  - Can be used when the problem have “**optimal substructure**”:
    - ✦ *Solution can be constructed from optimal solutions to subproblems*
    - ✦ *Use dynamic programming when subproblems overlap.*



# Computing Fibonacci numbers

---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

# Computing Fibonacci numbers

---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
    return 0  
else if n = 1  
    return 1  
else  
    return Fib(n-1) + Fib(n-2)
```

# Computing Fibonacci numbers

---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
    return 0  
else if n = 1  
    return 1  
else  
    return Fib(n-1) + Fib(n-2)
```

# Computing Fibonacci numbers

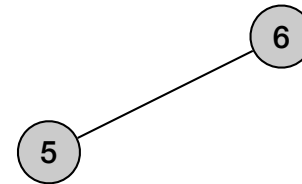
---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
    return 0  
else if n = 1  
    return 1  
else  
    return Fib(n-1) + Fib(n-2)
```



# Computing Fibonacci numbers

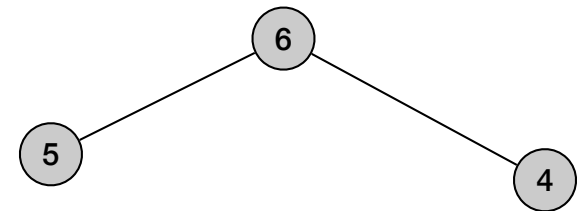
---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
    return 0  
else if n = 1  
    return 1  
else  
    return Fib(n-1) + Fib(n-2)
```



# Computing Fibonacci numbers

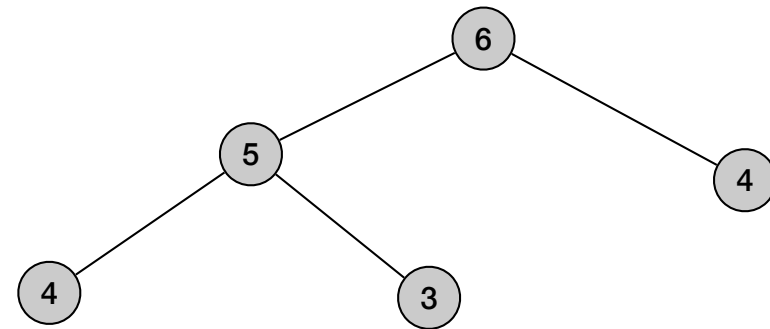
---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
    return 0  
else if n = 1  
    return 1  
else  
    return Fib(n-1) + Fib(n-2)
```



# Computing Fibonacci numbers

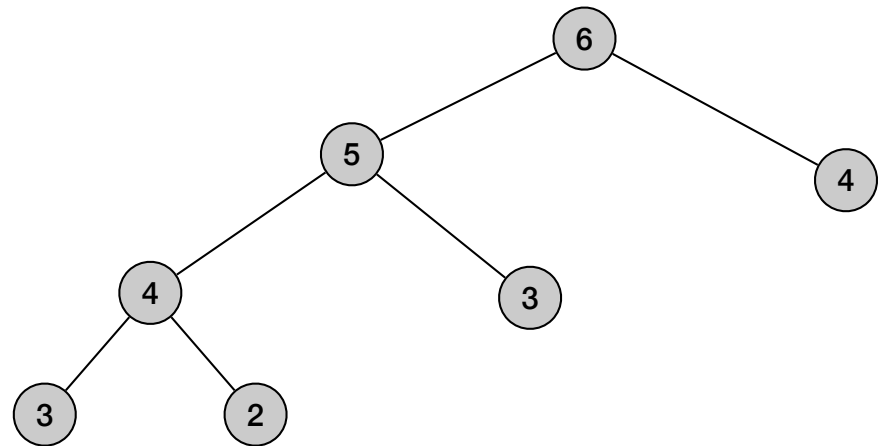
---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
  return 0  
else if n = 1  
  return 1  
else  
  return Fib(n-1) + Fib(n-2)
```



# Computing Fibonacci numbers

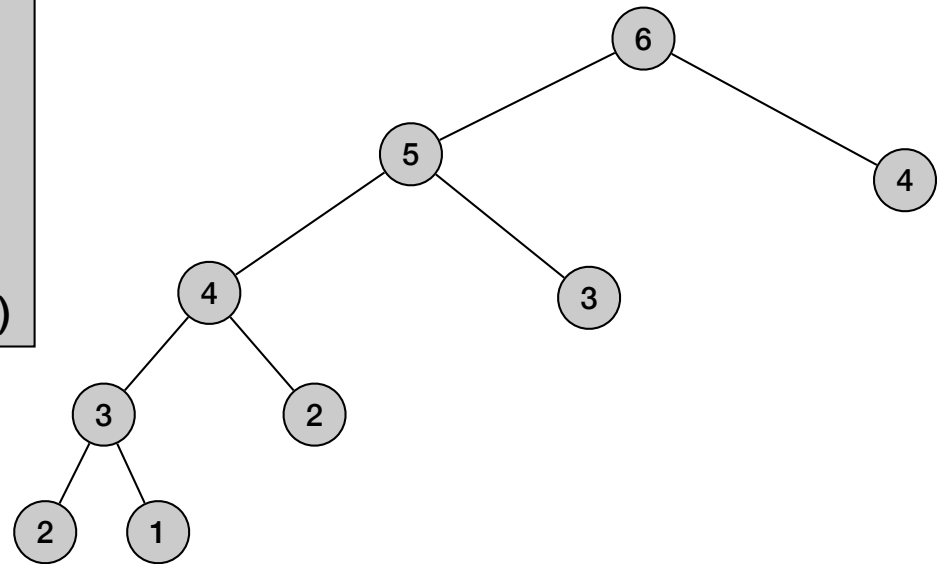
---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
  return 0  
else if n = 1  
  return 1  
else  
  return Fib(n-1) + Fib(n-2)
```





# Computing Fibonacci numbers

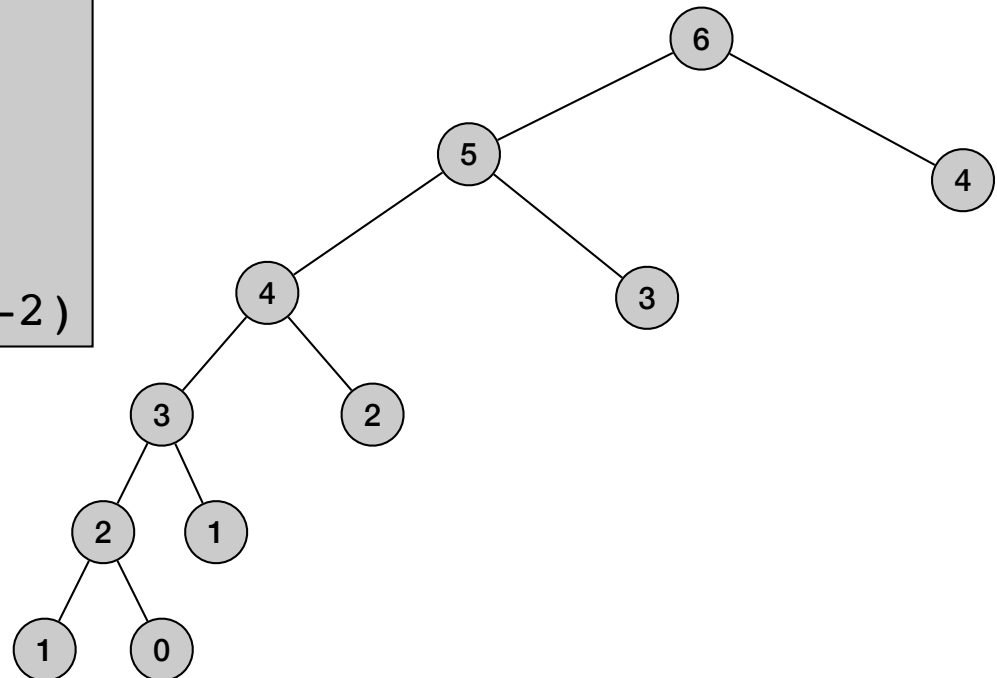
---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
  return 0  
else if n = 1  
  return 1  
else  
  return Fib(n-1) + Fib(n-2)
```



# Computing Fibonacci numbers

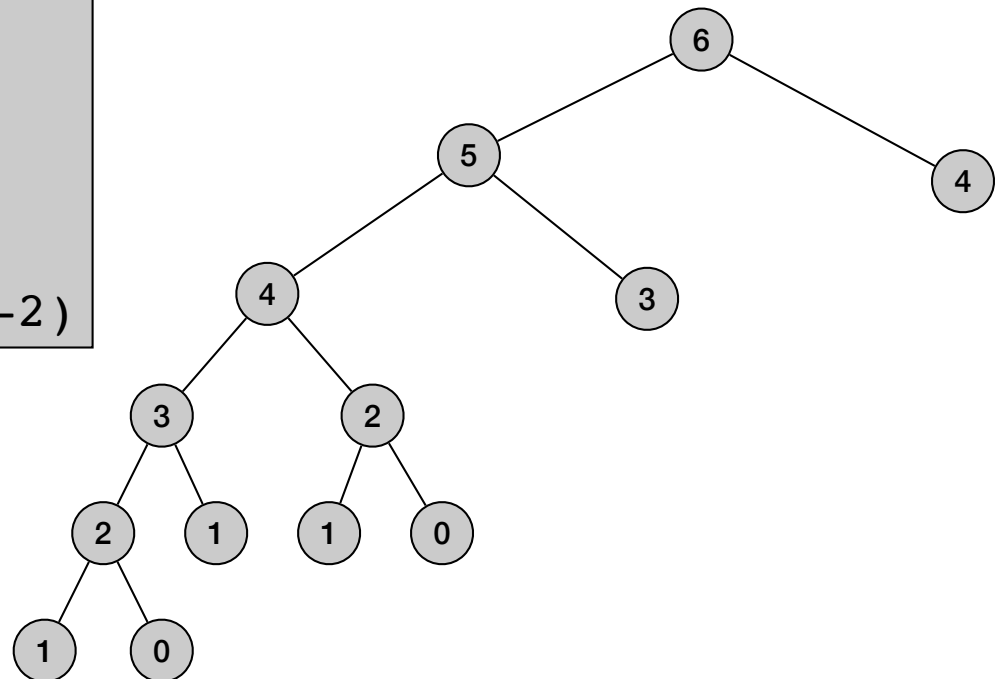
---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
  return 0  
else if n = 1  
  return 1  
else  
  return Fib(n-1) + Fib(n-2)
```



# Computing Fibonacci numbers

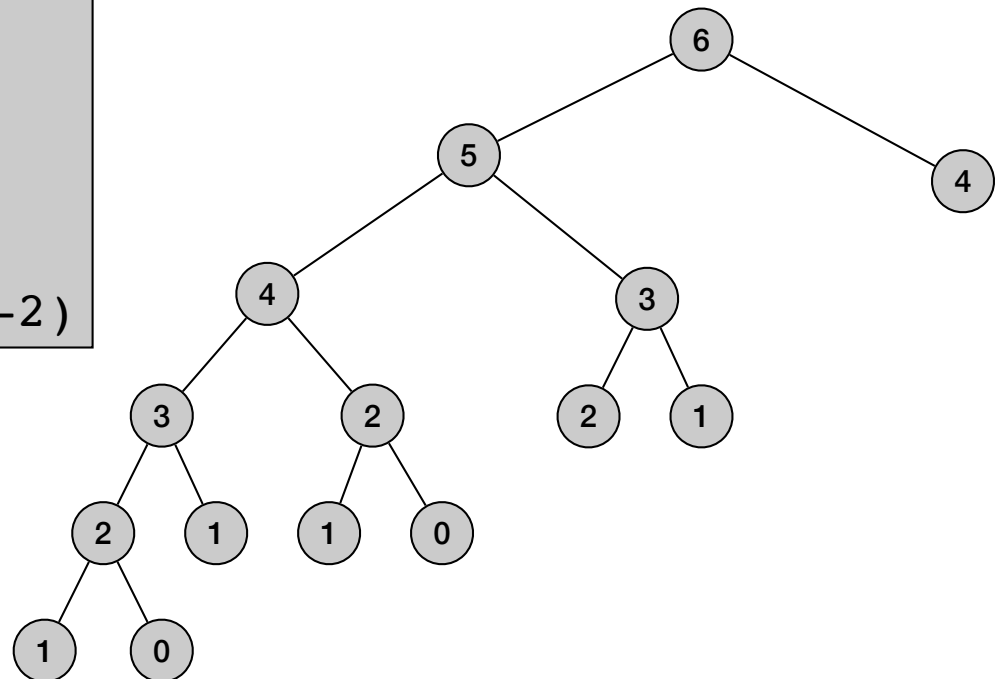
---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
  return 0  
else if n = 1  
  return 1  
else  
  return Fib(n-1) + Fib(n-2)
```



# Computing Fibonacci numbers

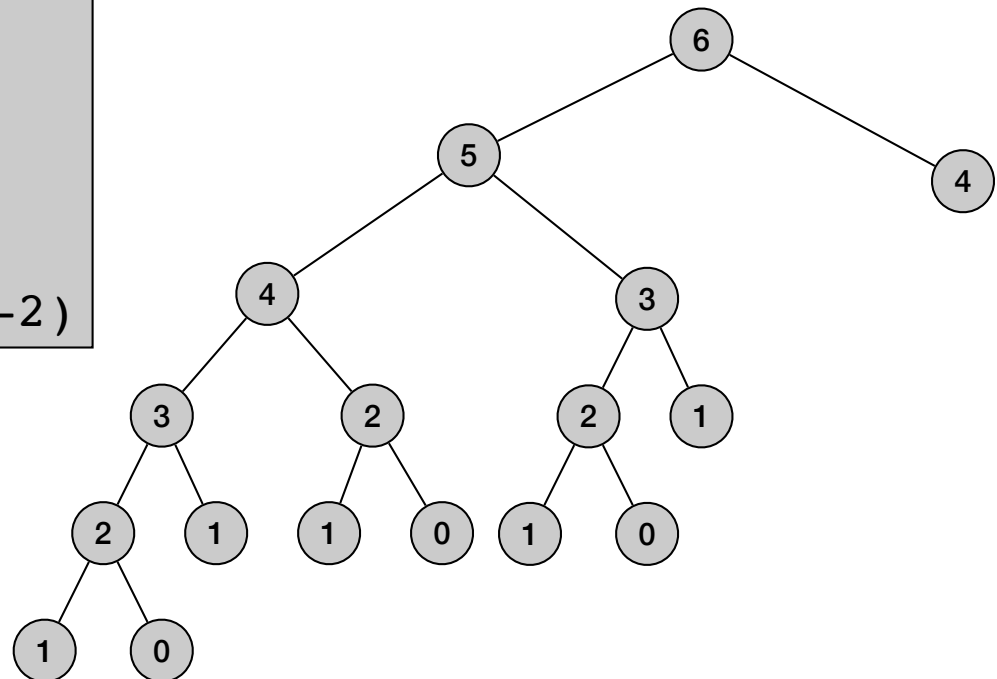
---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
  return 0  
else if n = 1  
  return 1  
else  
  return Fib(n-1) + Fib(n-2)
```



# Computing Fibonacci numbers

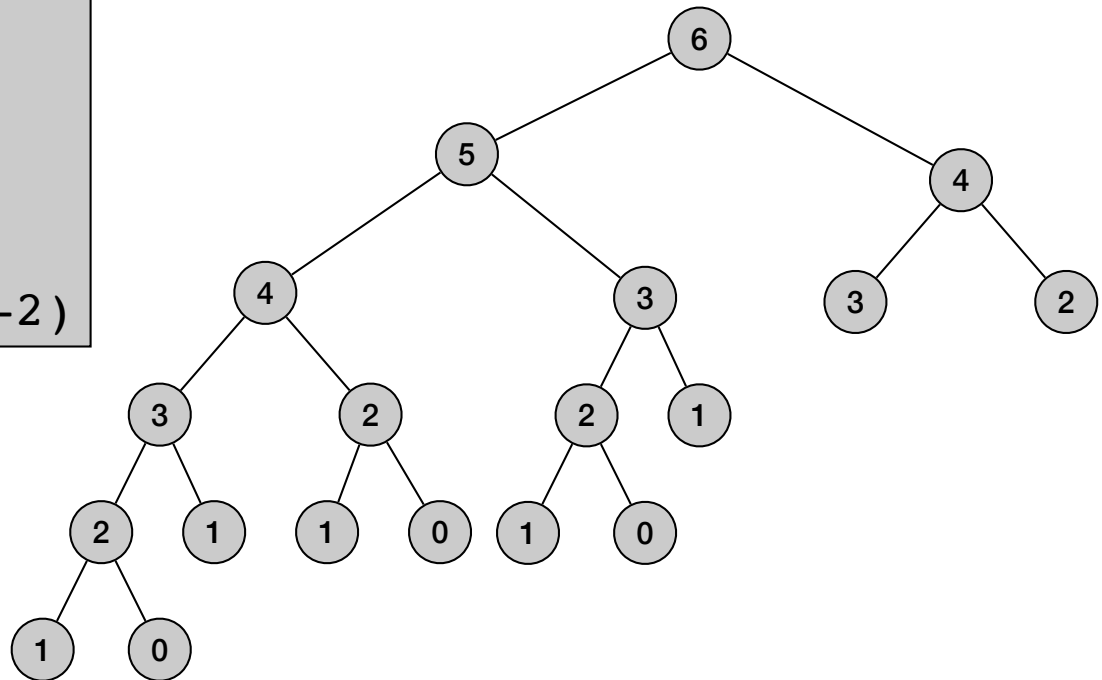
---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
  return 0  
else if n = 1  
  return 1  
else  
  return Fib(n-1) + Fib(n-2)
```



# Computing Fibonacci numbers

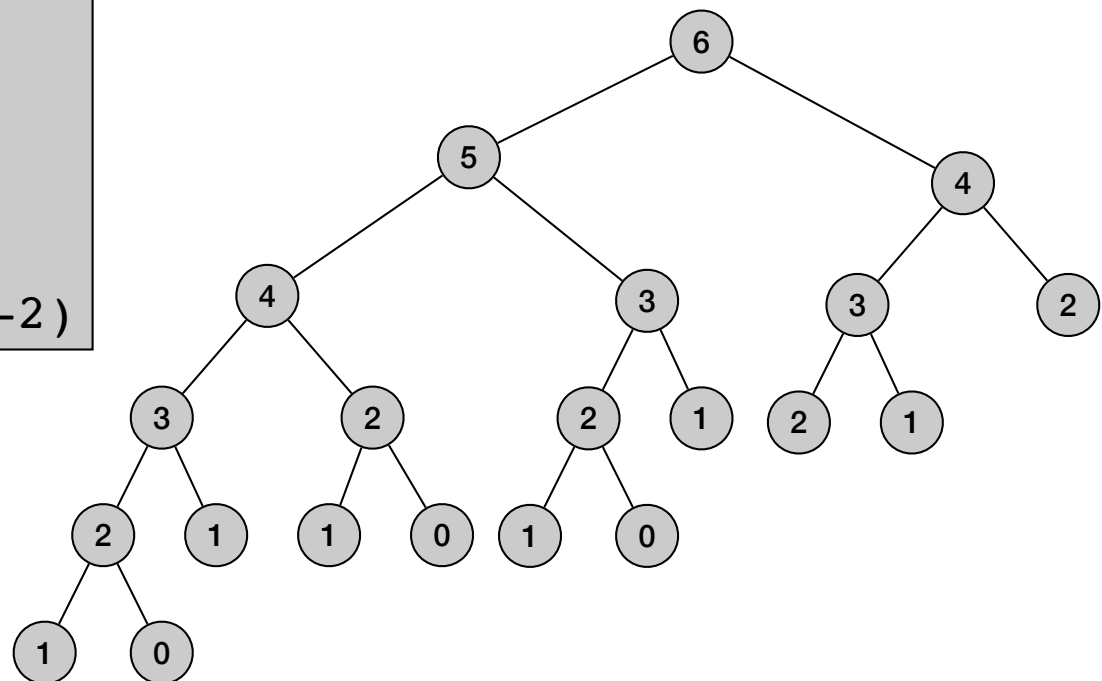
---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
  return 0  
else if n = 1  
  return 1  
else  
  return Fib(n-1) + Fib(n-2)
```



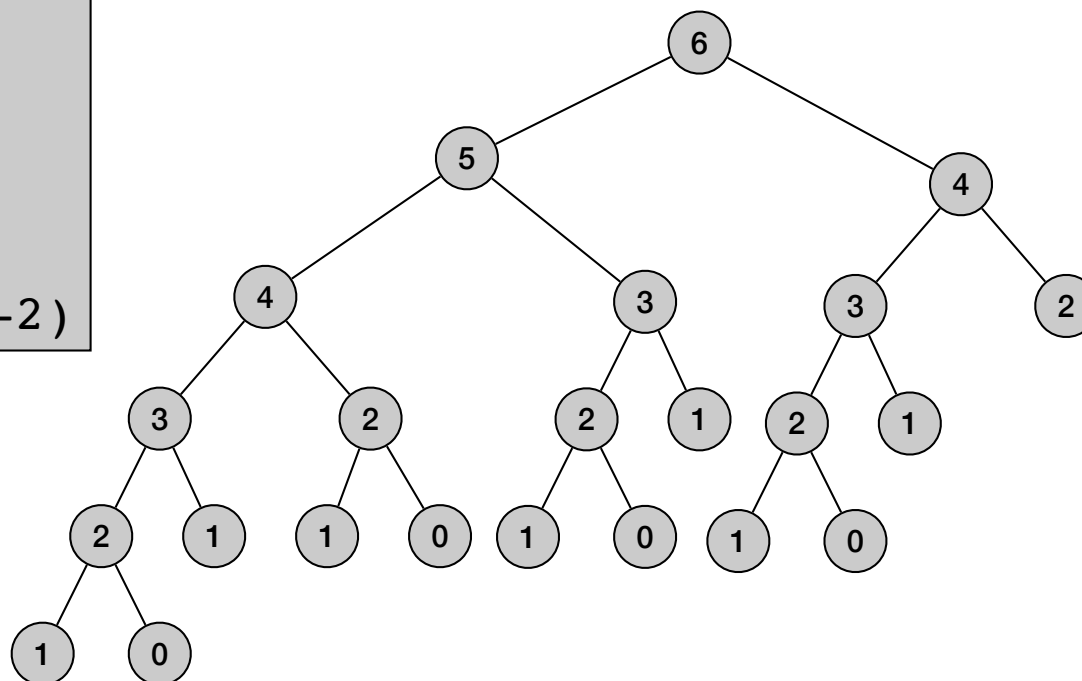
# Computing Fibonacci numbers

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)
if n = 0
  return 0
else if n = 1
  return 1
else
  return Fib(n-1) + Fib(n-2)
```







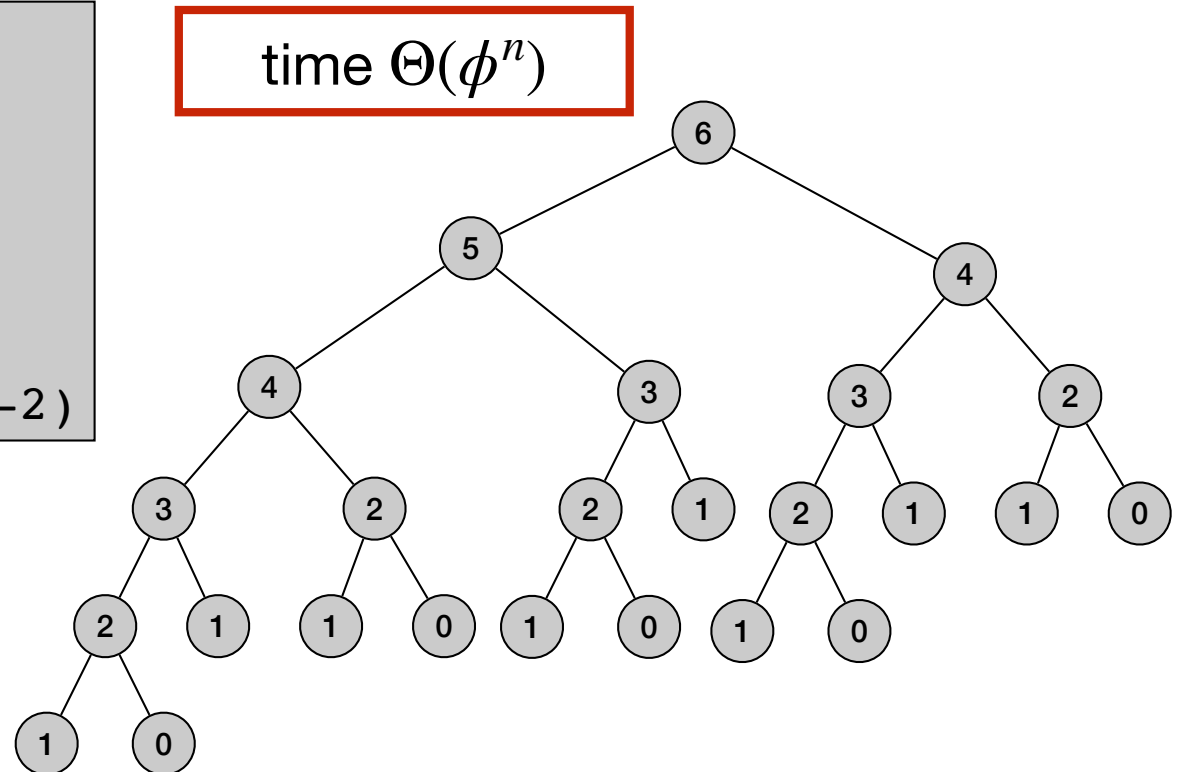
# Computing Fibonacci numbers

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
  return 0  
else if n = 1  
  return 1  
else  
  return Fib(n-1) + Fib(n-2)
```



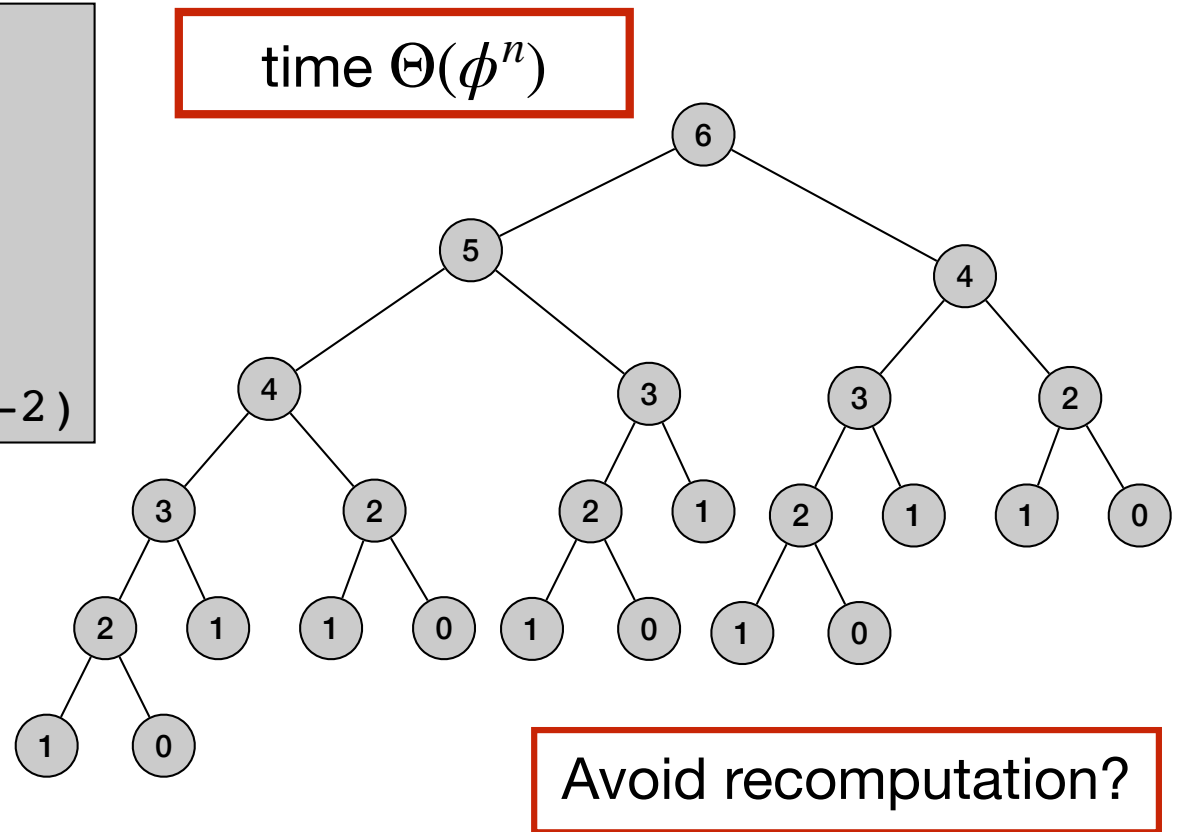
# Computing Fibonacci numbers

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)  
if n = 0  
  return 0  
else if n = 1  
  return 1  
else  
  return Fib(n-1) + Fib(n-2)
```



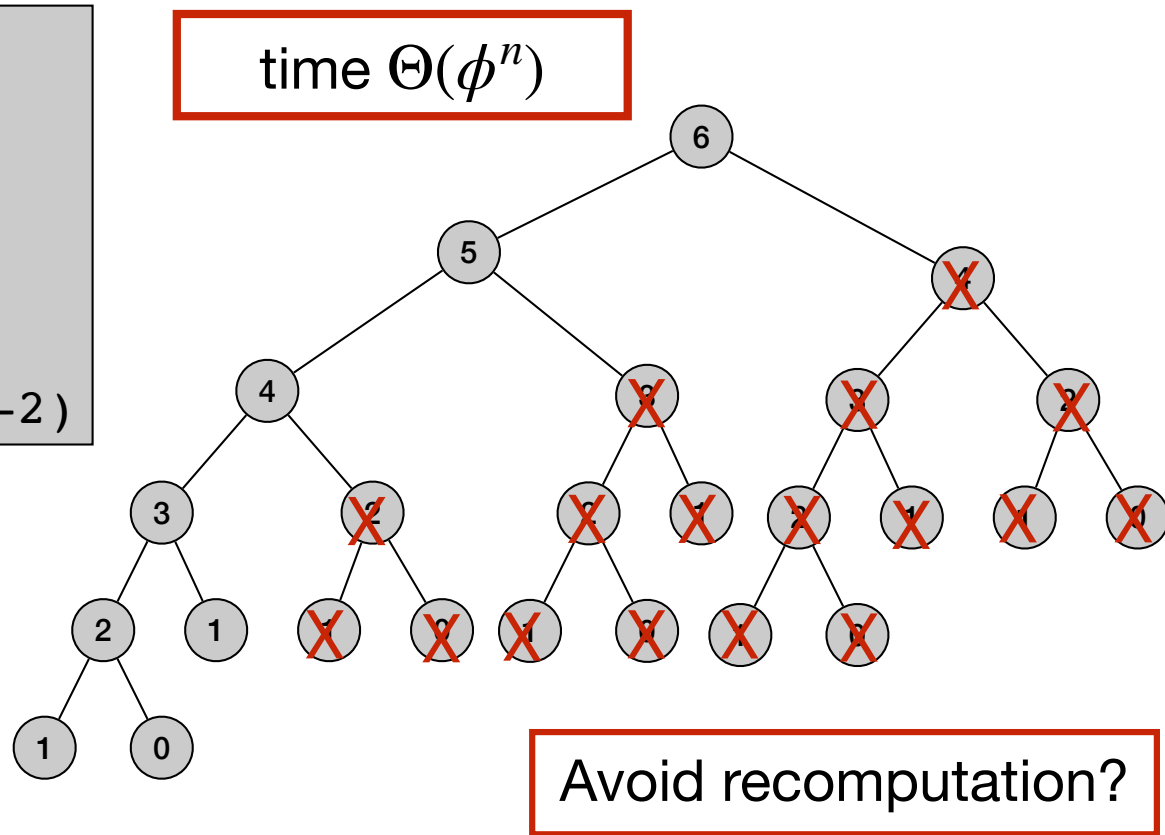
# Computing Fibonacci numbers

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- First try:

```
Fib(n)
if n = 0
  return 0
else if n = 1
  return 1
else
  return Fib(n-1) + Fib(n-2)
```



# Memoized Fibonacci numbers

---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- Remember already computed values:

# Memoized Fibonacci numbers

---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- Remember already computed values:

```
for j=1 to n
  F[j] = null
Mem-Fib(n)
```

# Memoized Fibonacci numbers

---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- Remember already computed values:

```
for j=1 to n
  F[j] = null
Mem-Fib(n)

Mem-Fib(n)
if n = 0
  return 0
else if n = 1
  return 1
else
  if F[n] is empty
    F[n] = Mem-Fib(n-1) + Mem-Fib(n-2)
  return F[n]
```

# Memoized Fibonacci numbers

- Fibonacci numbers:

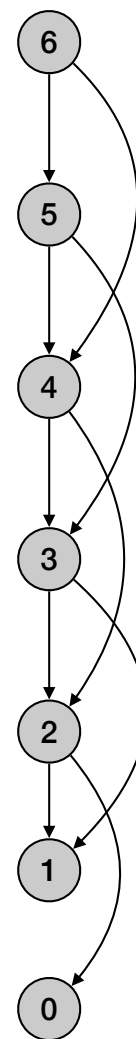
$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- Remember already computed values:

```
for j=1 to n
  F[j] = null
Mem-Fib(n)

Mem-Fib(n)
if n = 0
  return 0
else if n = 1
  return 1
else
  if F[n] is empty
    F[n] = Mem-Fib(n-1) + Mem-Fib(n-2)
  return F[n]
```

time  $\Theta(n)$



# Bottom-up Fibonacci numbers

---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- Remember already computed values:



# Bottom-up Fibonacci numbers

---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- Remember already computed values:

```
Iter-Fib(n)  
F[0] = 0  
F[1] = 1  
for i = 2 to n  
    F[i] = F[i-1] + F[i-2]  
return F[n]
```

time  $\Theta(n)$   
space  $\Theta(n)$

# Bottom-up Fibonacci numbers - save space

---

- Fibonacci numbers:

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- Remember last two computed values:

```
Iter-Fib(n)  
previous = 0  
current = 1  
for i = 1 to n  
    next = previous + current  
    previous = current  
    current = next  
return current
```

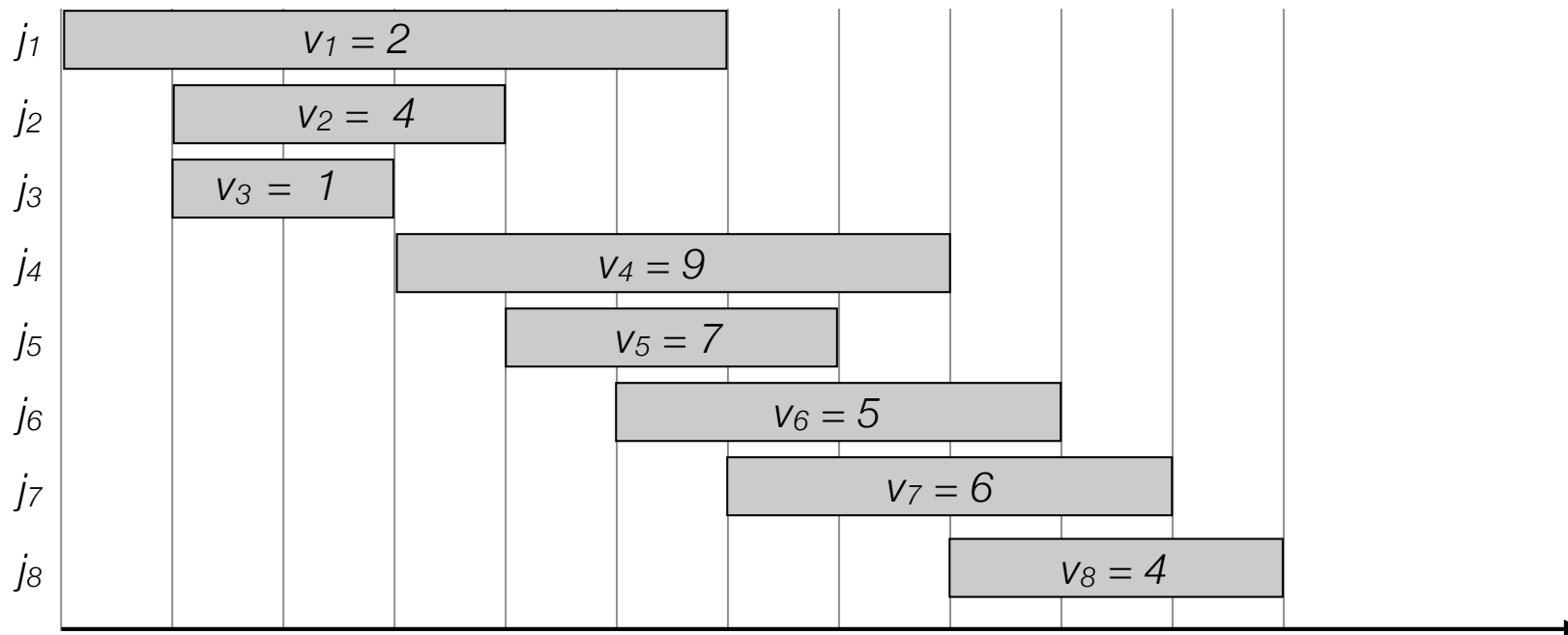
time  $\Theta(n)$   
space  $\Theta(1)$

# Weighted Interval Scheduling

# Weighted interval scheduling

- Weighted interval scheduling problem

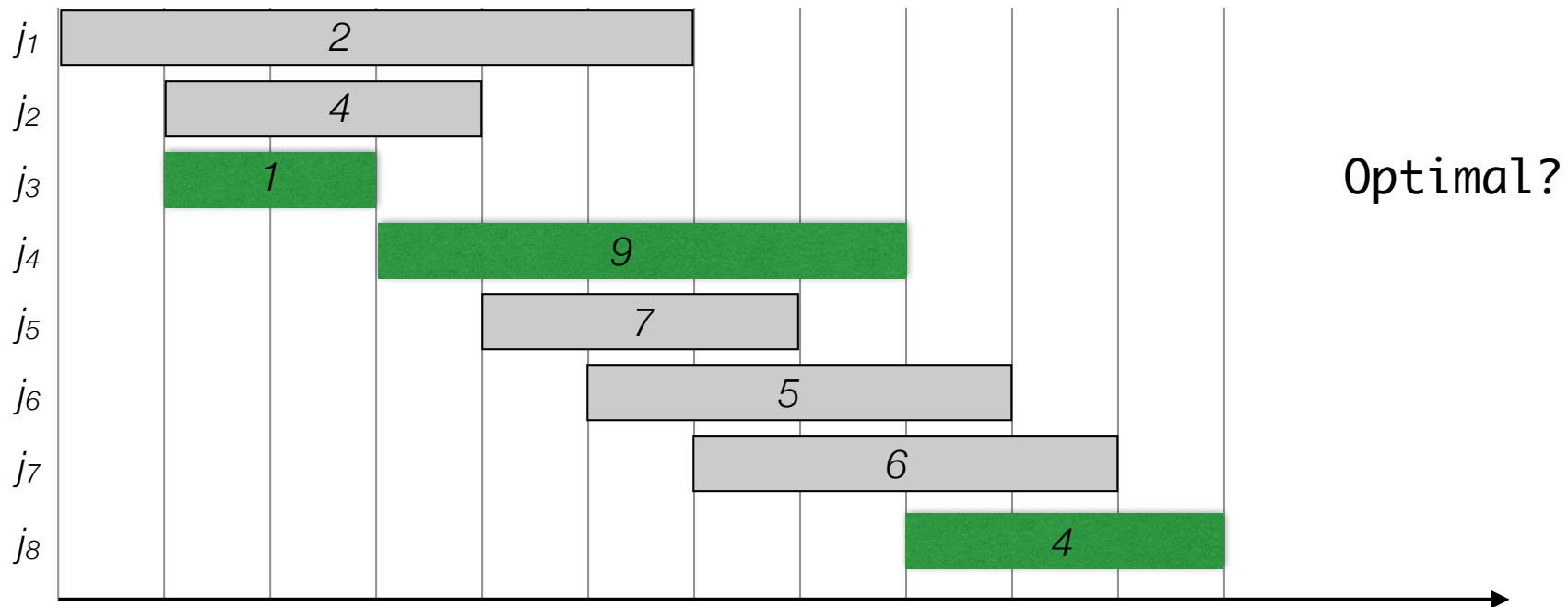
- $n$  jobs (intervals)
- Job  $i$  starts at  $s_i$ , finishes at  $f_i$  and has weight/value  $v_i$ .
- Goal: Find maximum weight subset of non-overlapping (compatible) jobs.



# Weighted interval scheduling

- Weighted interval scheduling problem

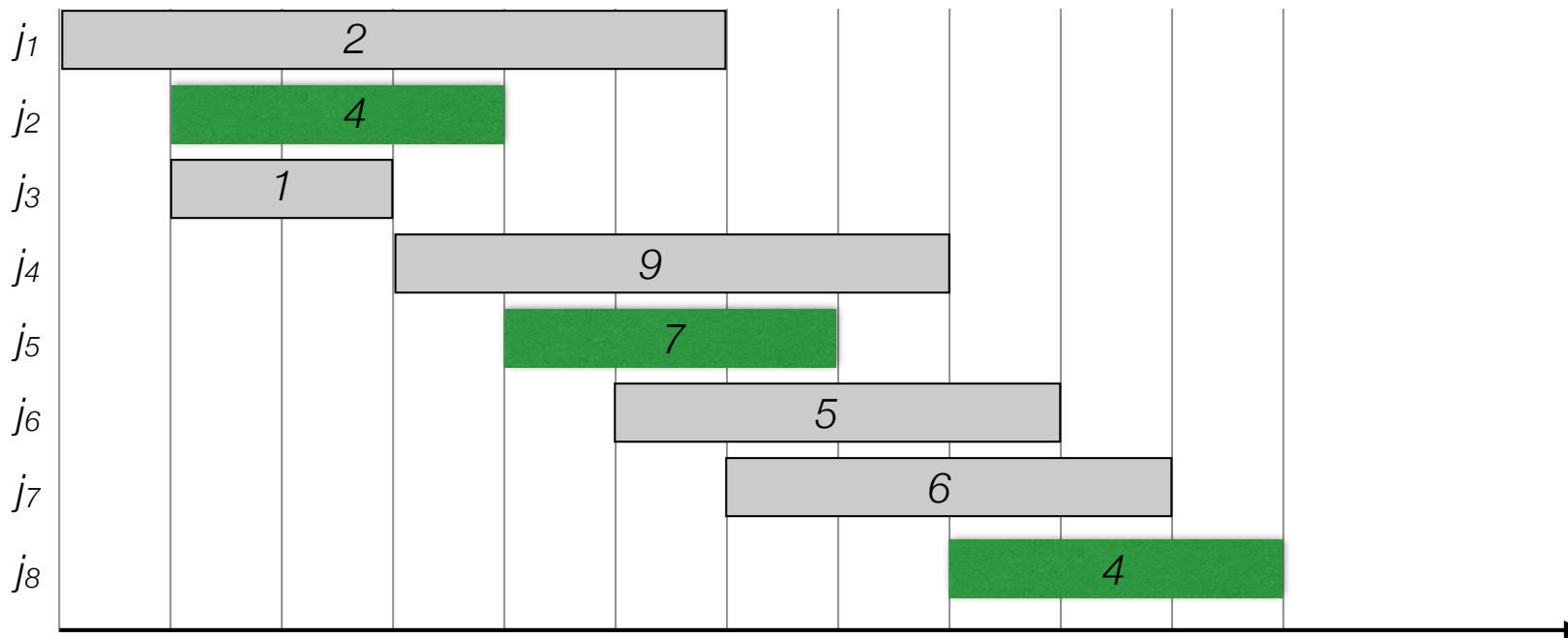
- $n$  jobs (intervals)
- Job  $i$  starts at  $s_i$ , finishes at  $f_i$  and has weight/value  $v_i$ .
- Goal: Find maximum weight subset of non-overlapping (compatible) jobs.



# Weighted interval scheduling

- Weighted interval scheduling problem

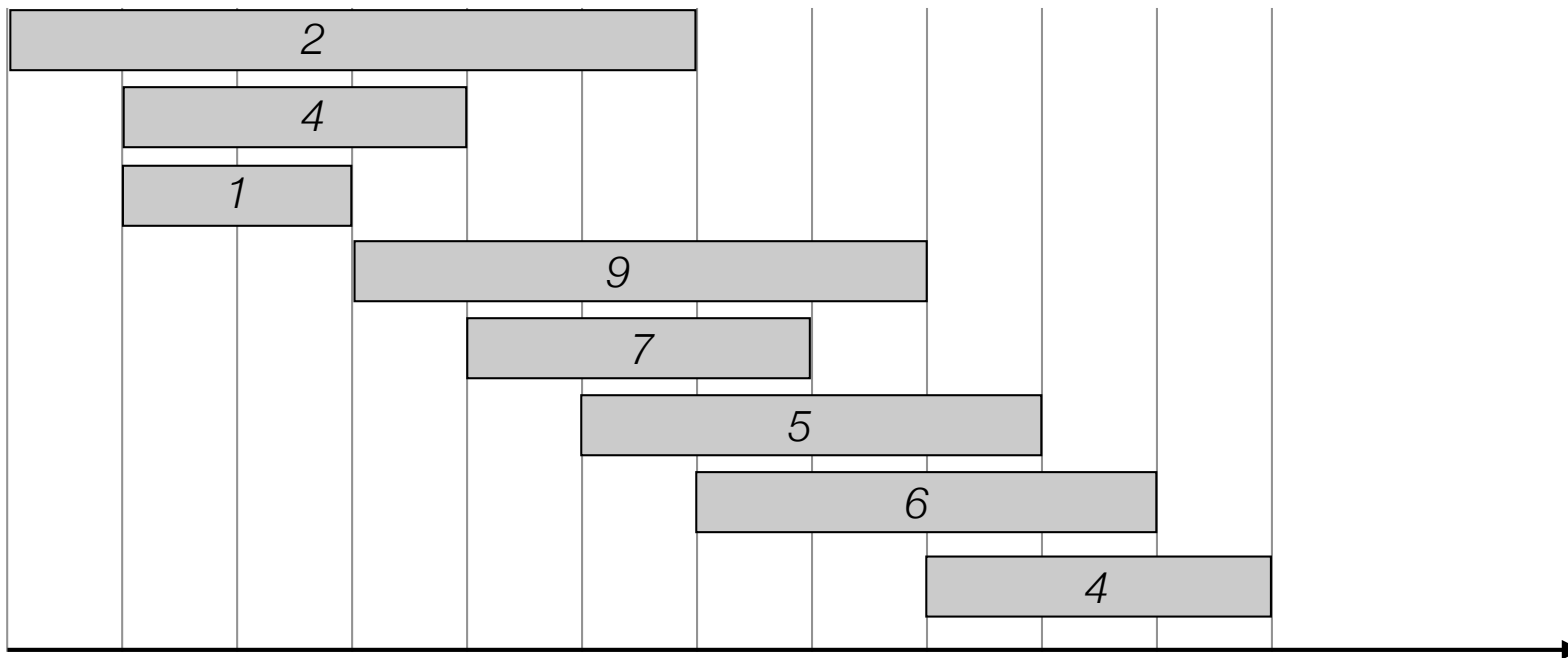
- $n$  jobs (intervals)
- Job  $i$  starts at  $s_i$ , finishes at  $f_i$  and has weight/value  $v_i$ .
- Goal: Find maximum weight subset of non-overlapping (compatible) jobs.



# Weighted interval scheduling

---

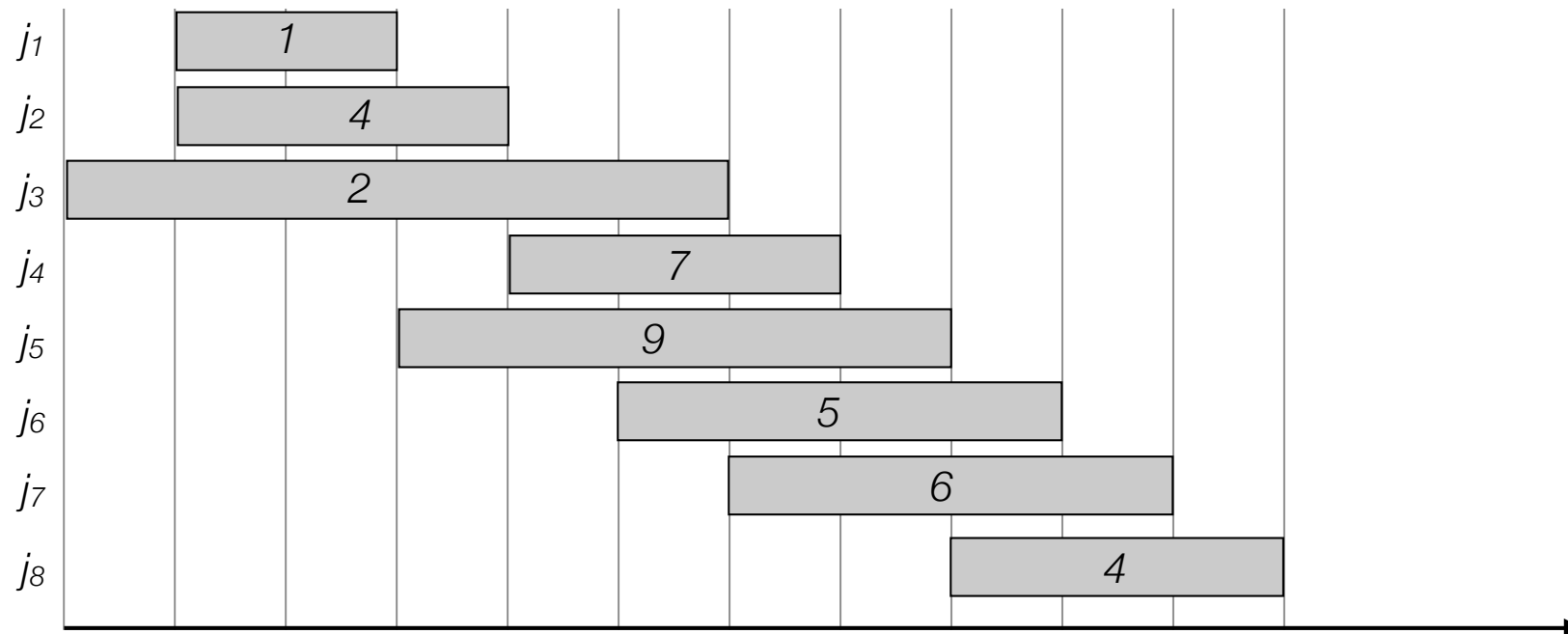
- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$



# Weighted interval scheduling

---

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$

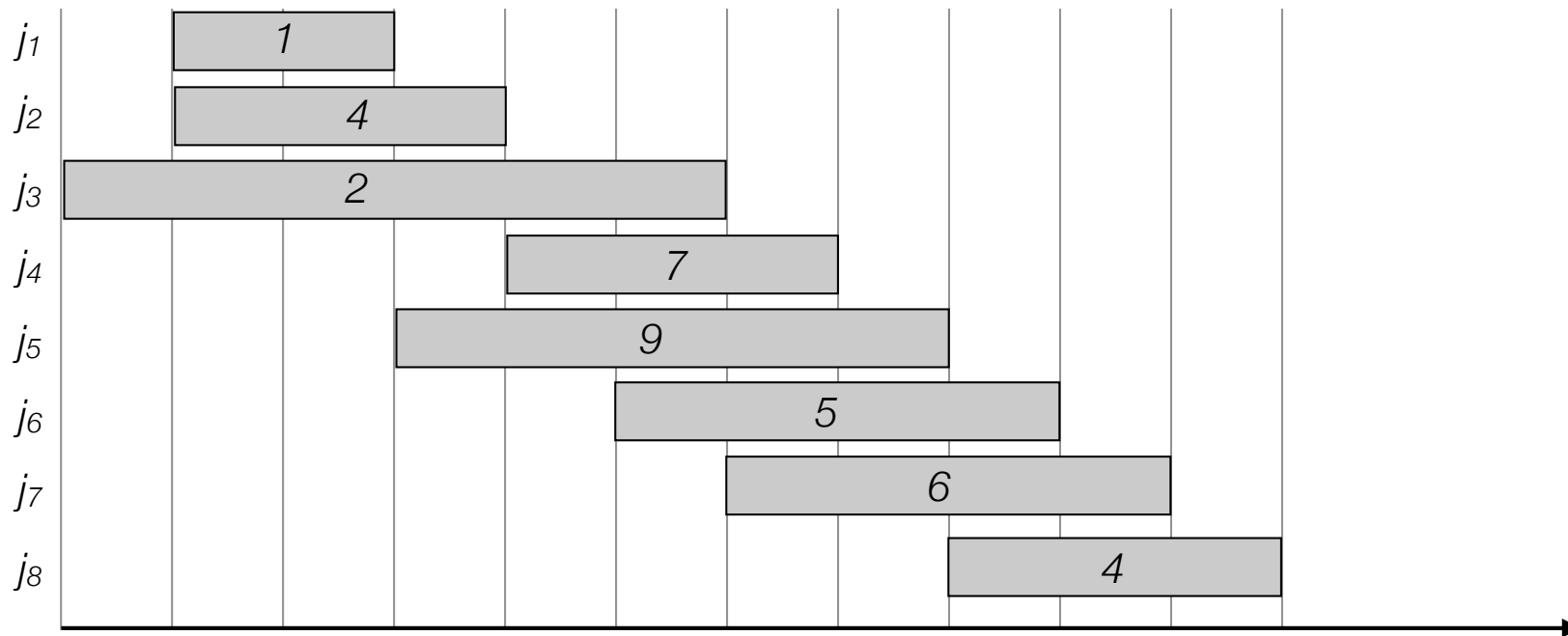




# Weighted interval scheduling

---

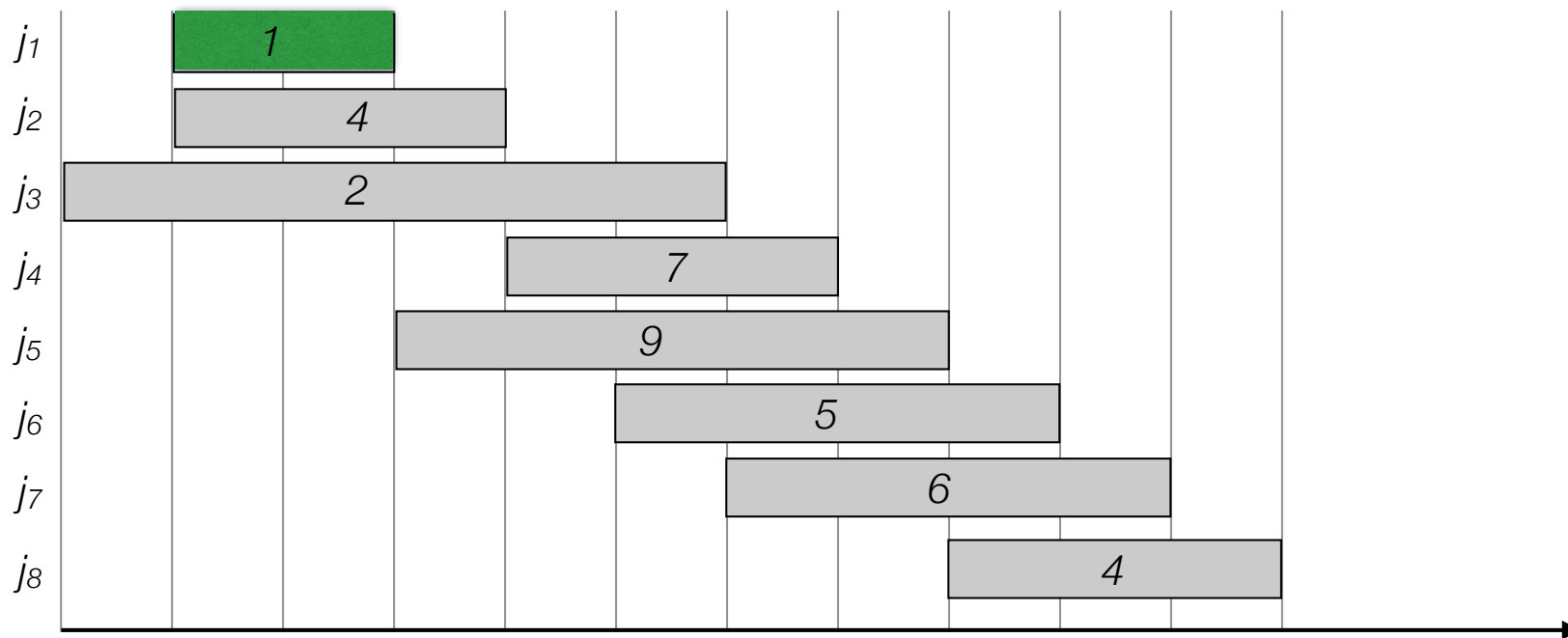
- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- Greedy?



# Weighted interval scheduling

---

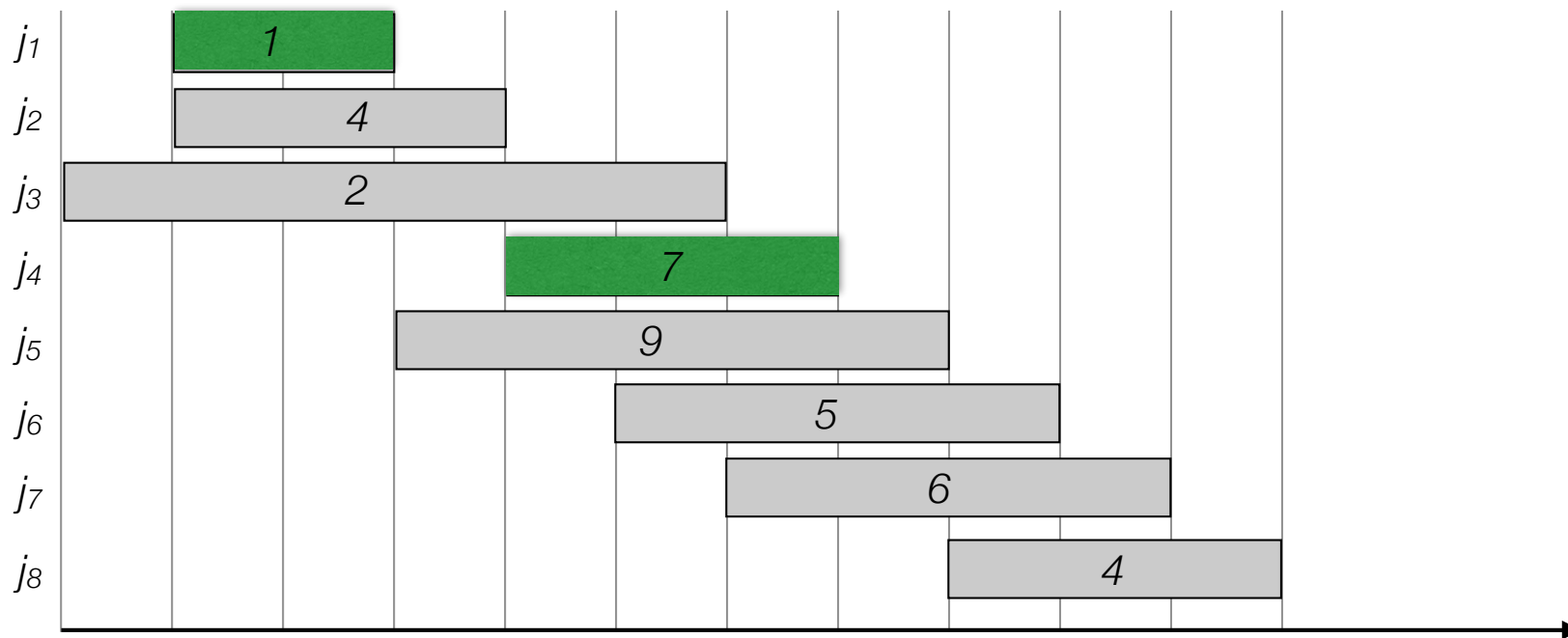
- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- Greedy?



# Weighted interval scheduling

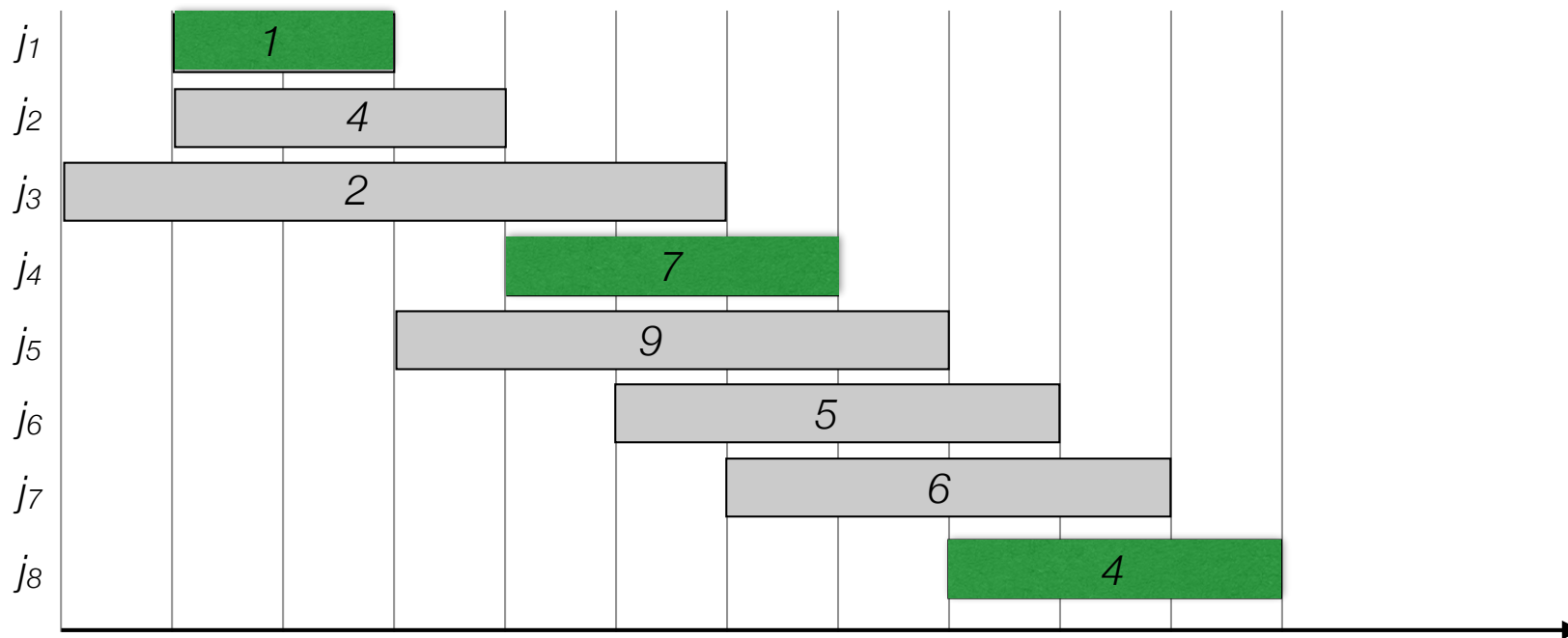
---

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- Greedy?



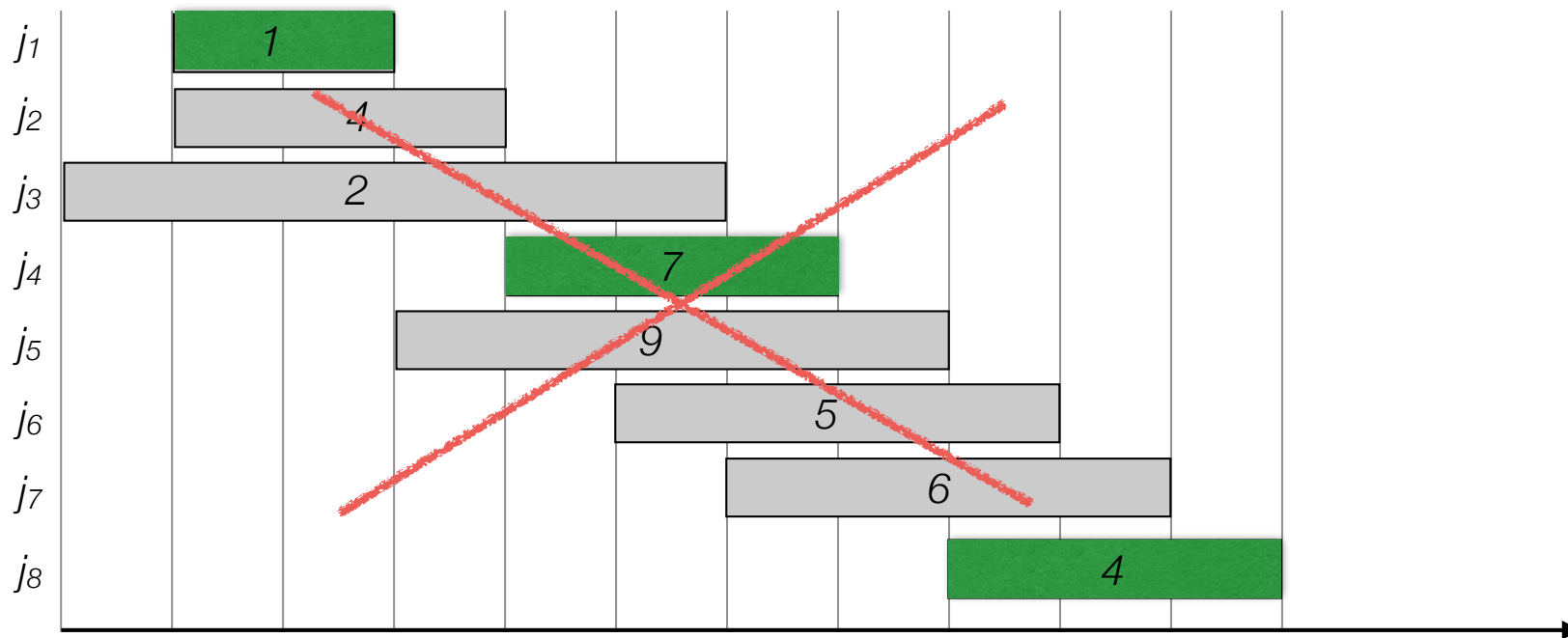
# Weighted interval scheduling

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- Greedy?



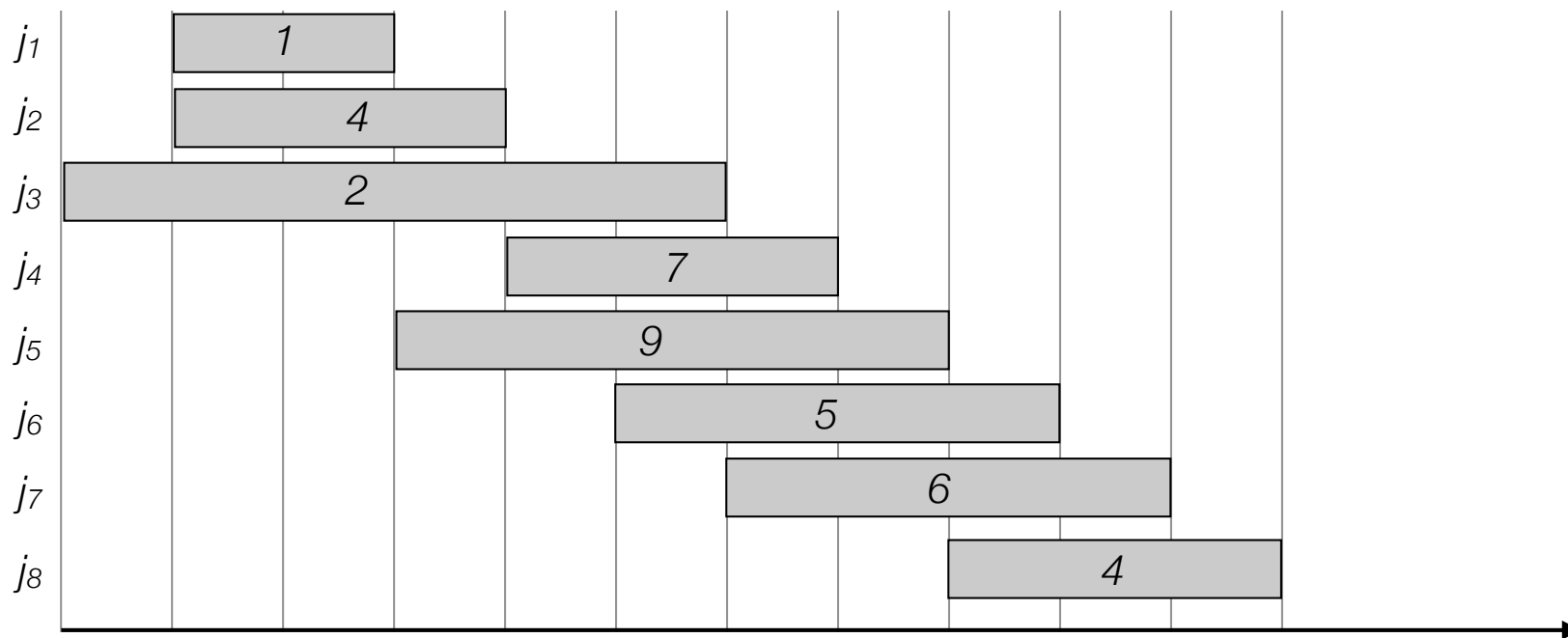
# Weighted interval scheduling

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- ~~Greedy?~~



# Weighted interval scheduling

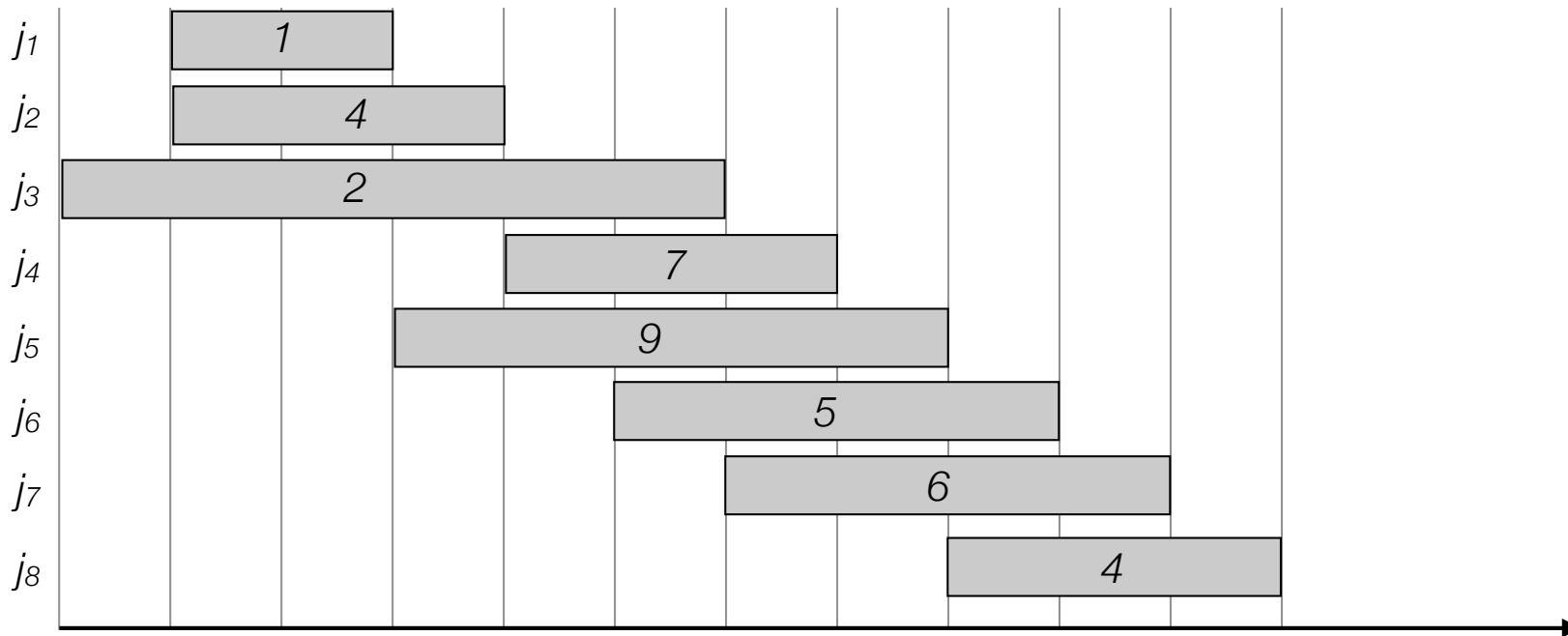
- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$



# Weighted interval scheduling

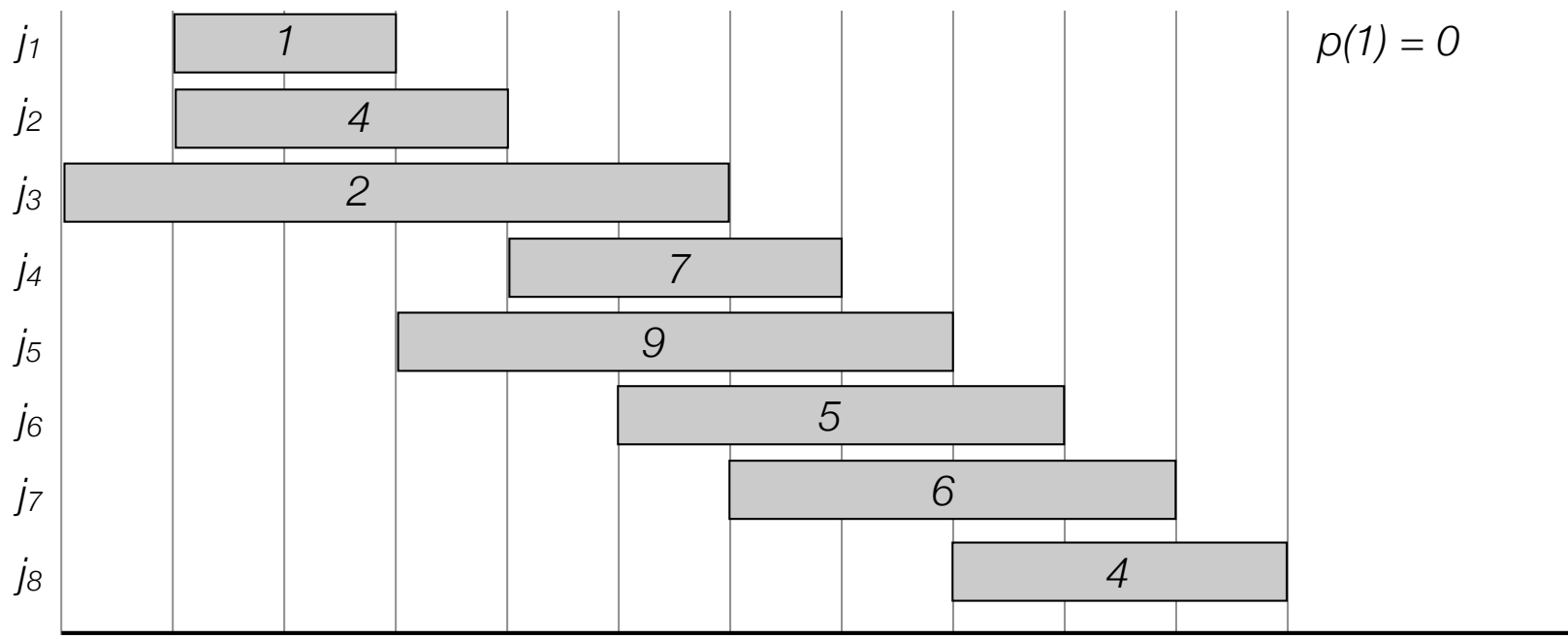
---

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j) =$  largest index  $i < j$  such that job  $i$  is compatible with  $j$ .



# Weighted interval scheduling

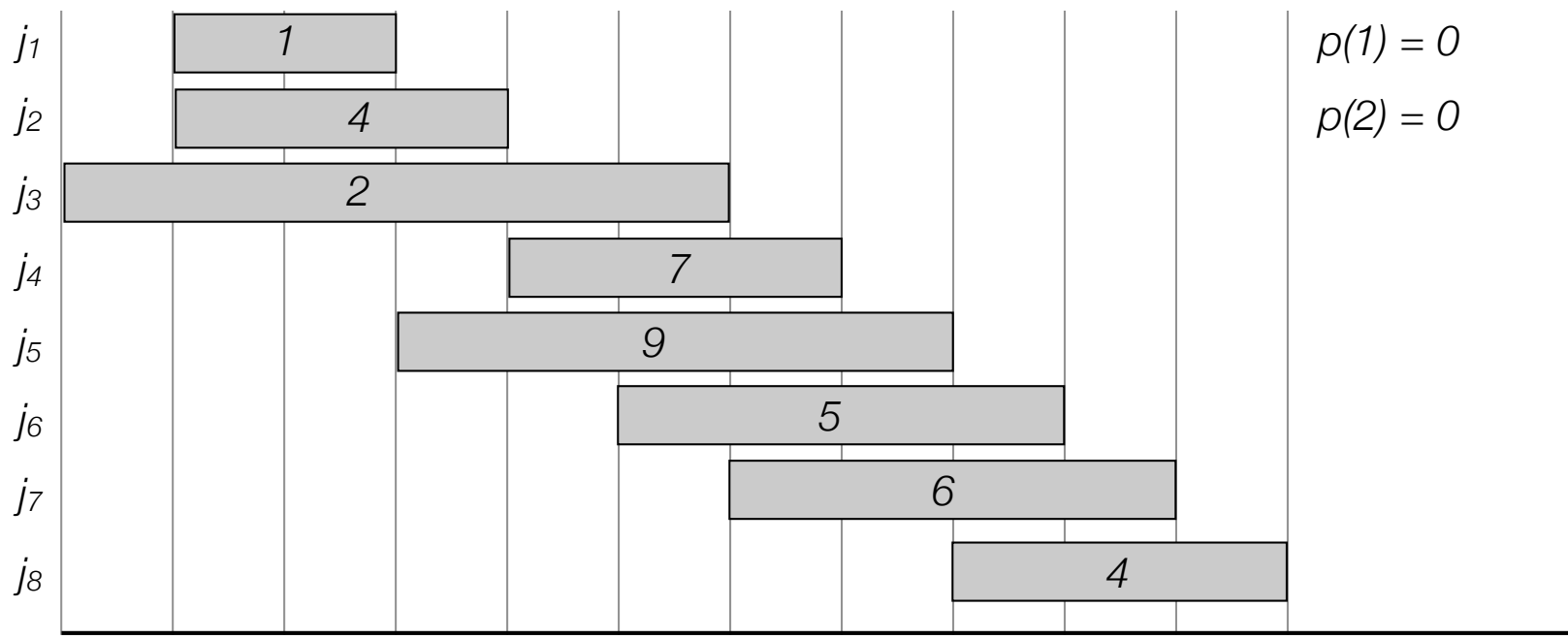
- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j)$  = largest index  $i < j$  such that job  $i$  is compatible with  $j$ .





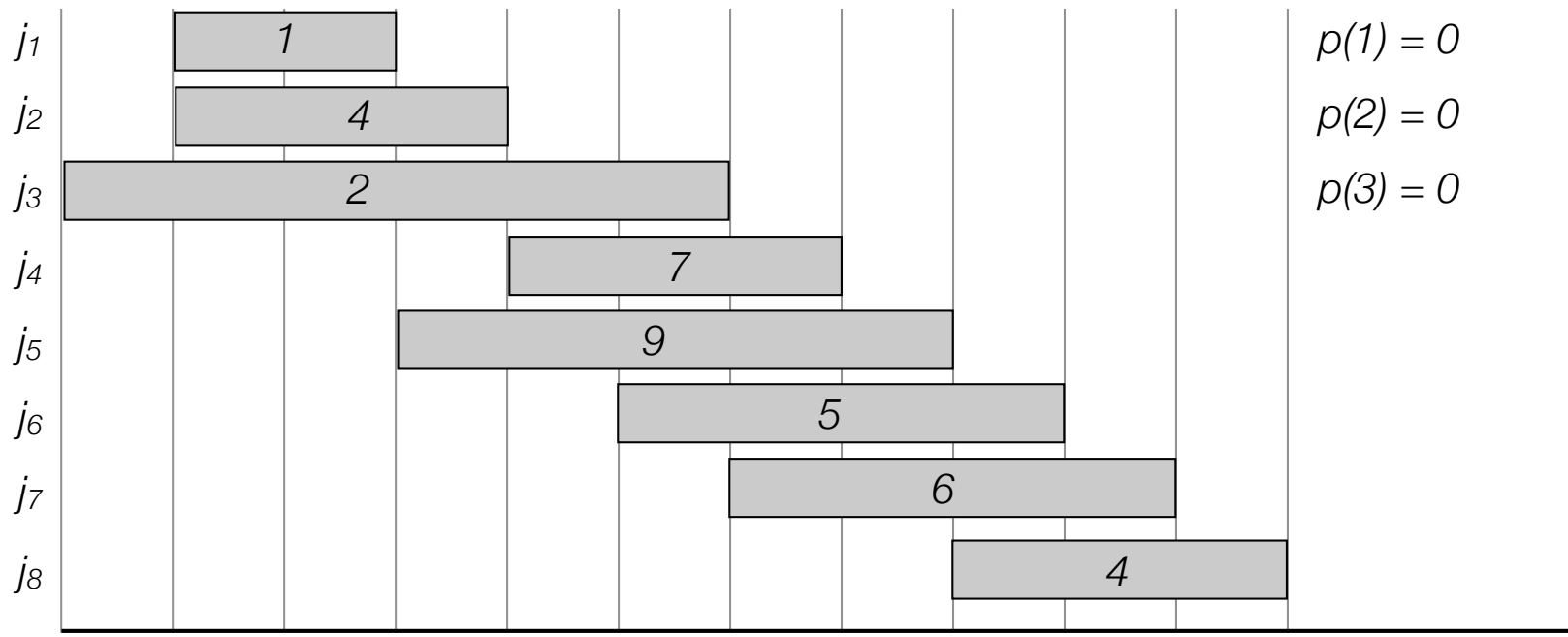
# Weighted interval scheduling

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j)$  = largest index  $i < j$  such that job  $i$  is compatible with  $j$ .



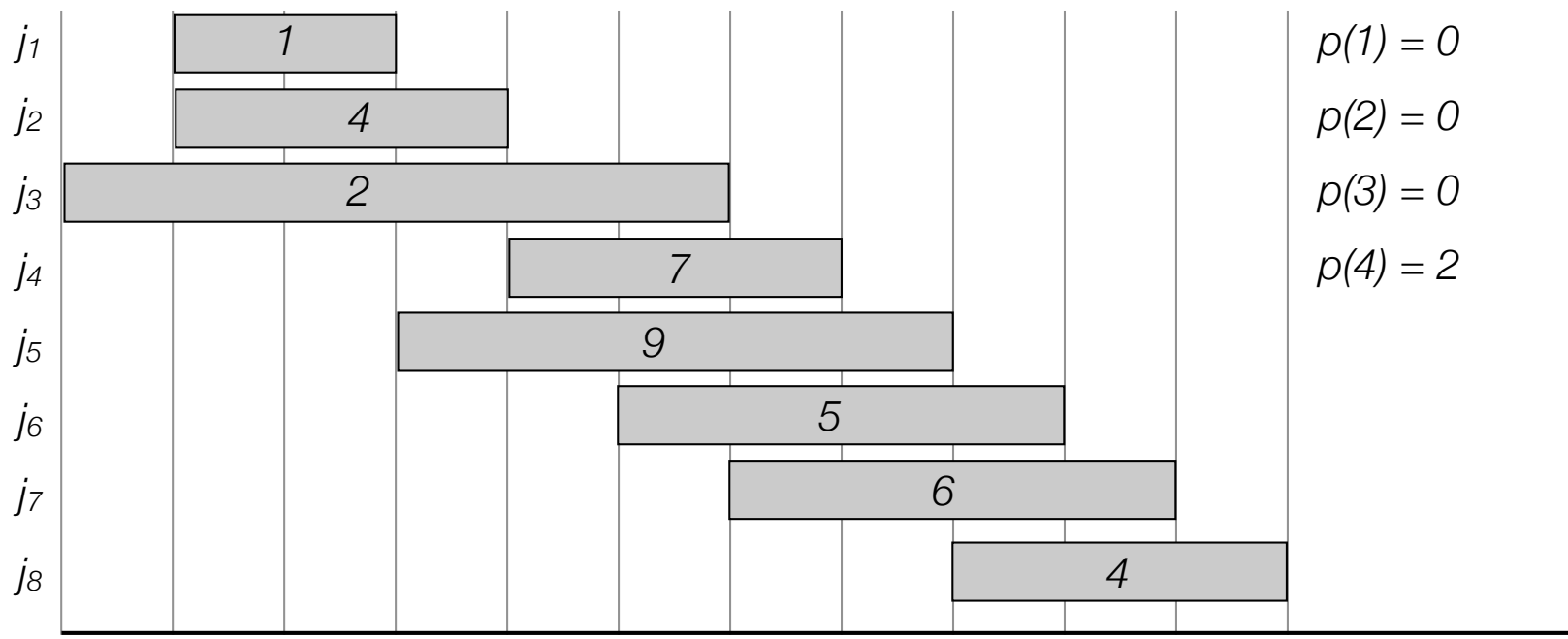
# Weighted interval scheduling

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j)$  = largest index  $i < j$  such that job  $i$  is compatible with  $j$ .



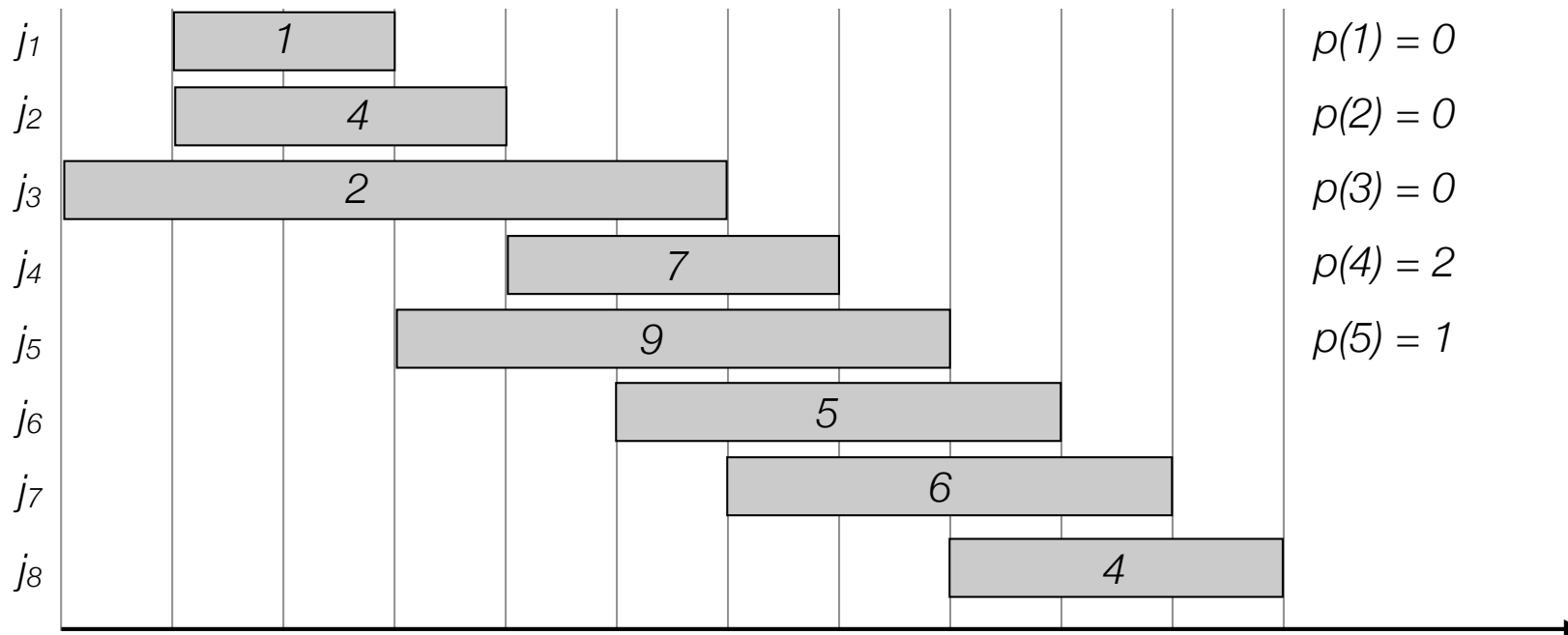
# Weighted interval scheduling

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j)$  = largest index  $i < j$  such that job  $i$  is compatible with  $j$ .



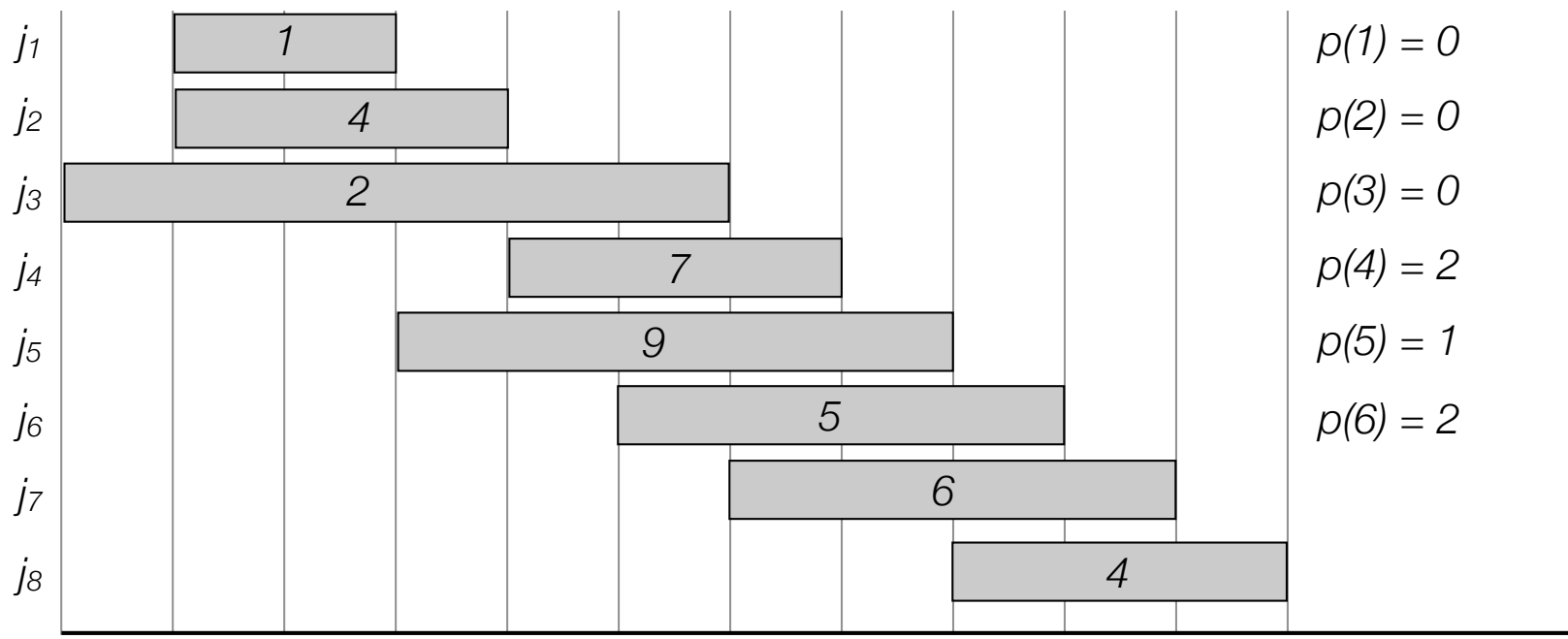
# Weighted interval scheduling

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j) =$  largest index  $i < j$  such that job  $i$  is compatible with  $j$ .



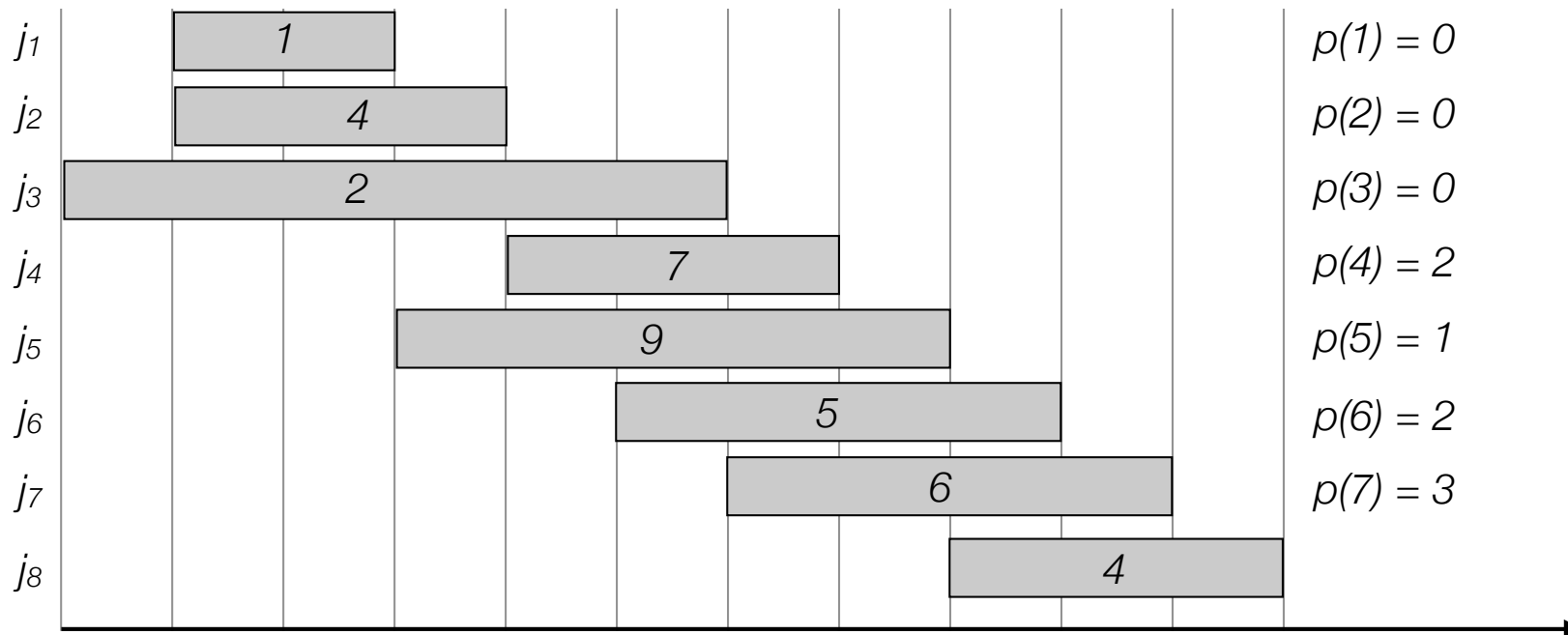
# Weighted interval scheduling

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j) =$  largest index  $i < j$  such that job  $i$  is compatible with  $j$ .



# Weighted interval scheduling

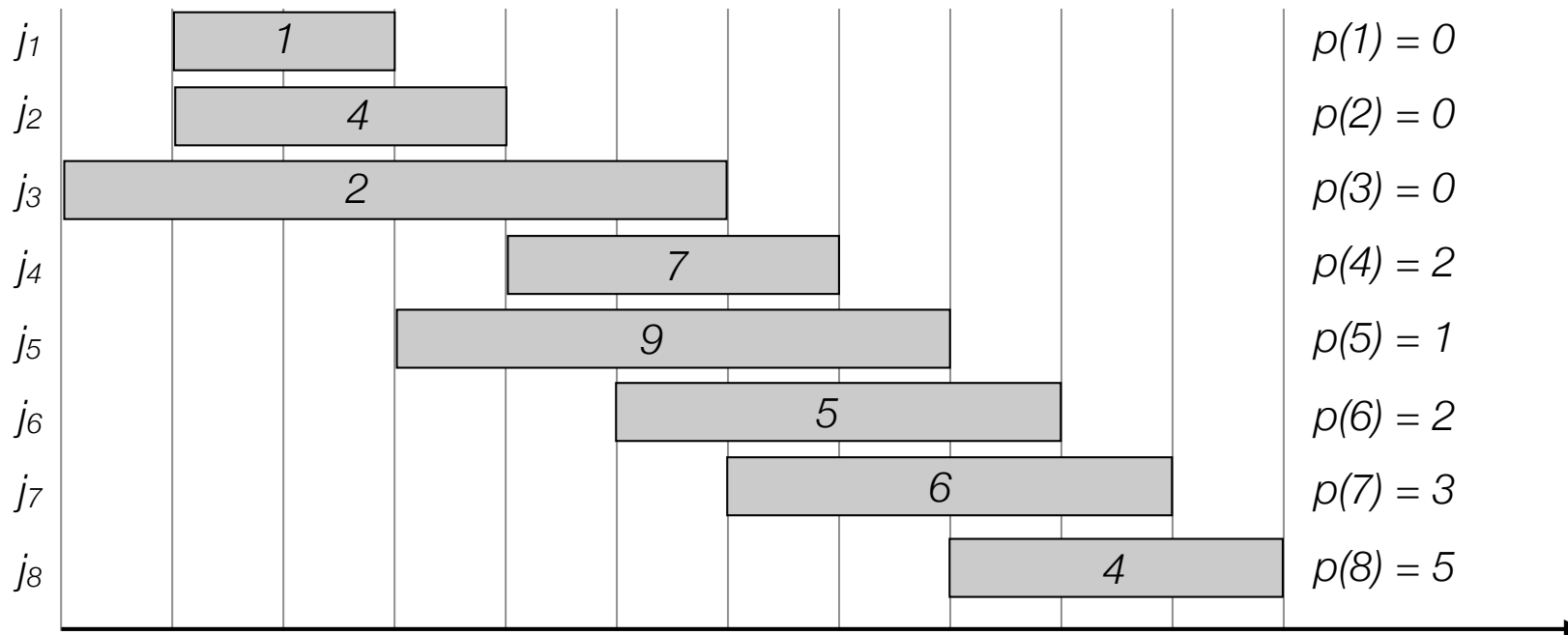
- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j) =$  largest index  $i < j$  such that job  $i$  is compatible with  $j$ .



# Weighted interval scheduling

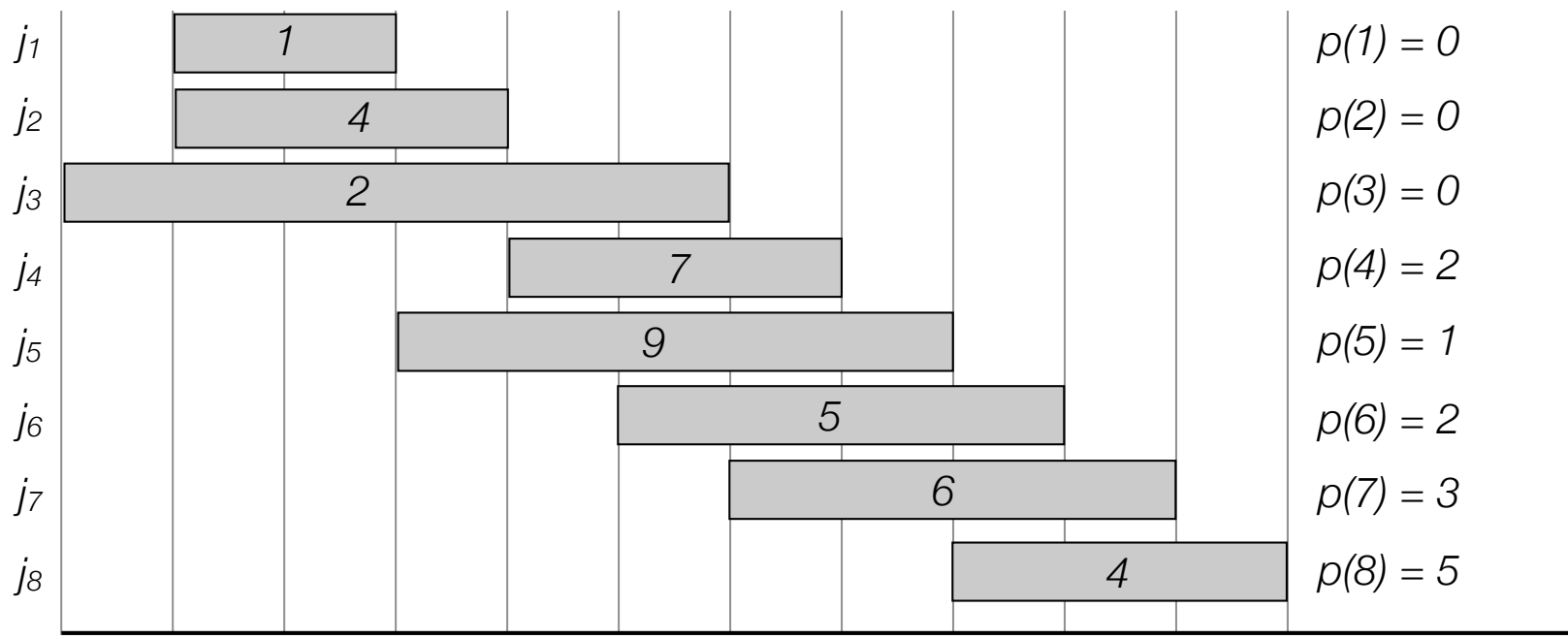
---

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j)$  = largest index  $i < j$  such that job  $i$  is compatible with  $j$ .



# Weighted interval scheduling

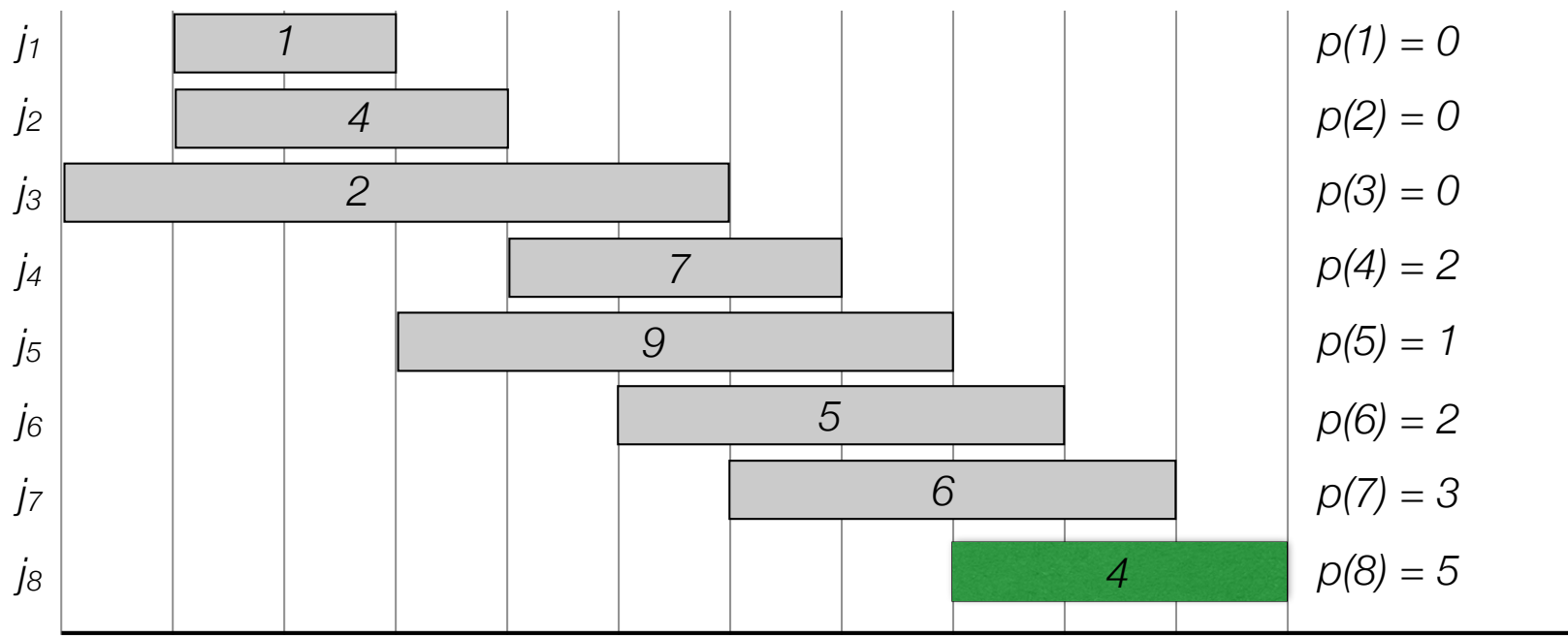
- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j)$  = largest index  $i < j$  such that job  $i$  is compatible with  $j$ .
- Optimal solution OPT:





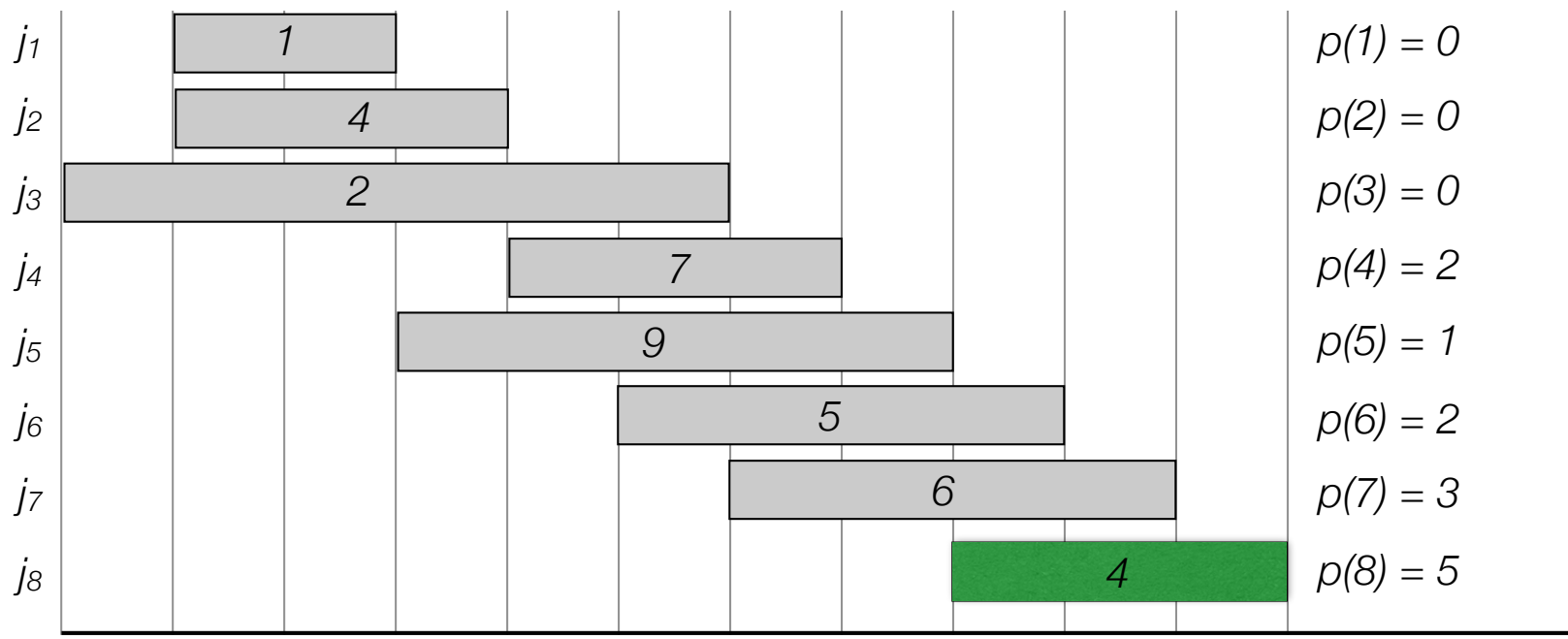
# Weighted interval scheduling

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j)$  = largest index  $i < j$  such that job  $i$  is compatible with  $j$ .
- Optimal solution OPT:



# Weighted interval scheduling

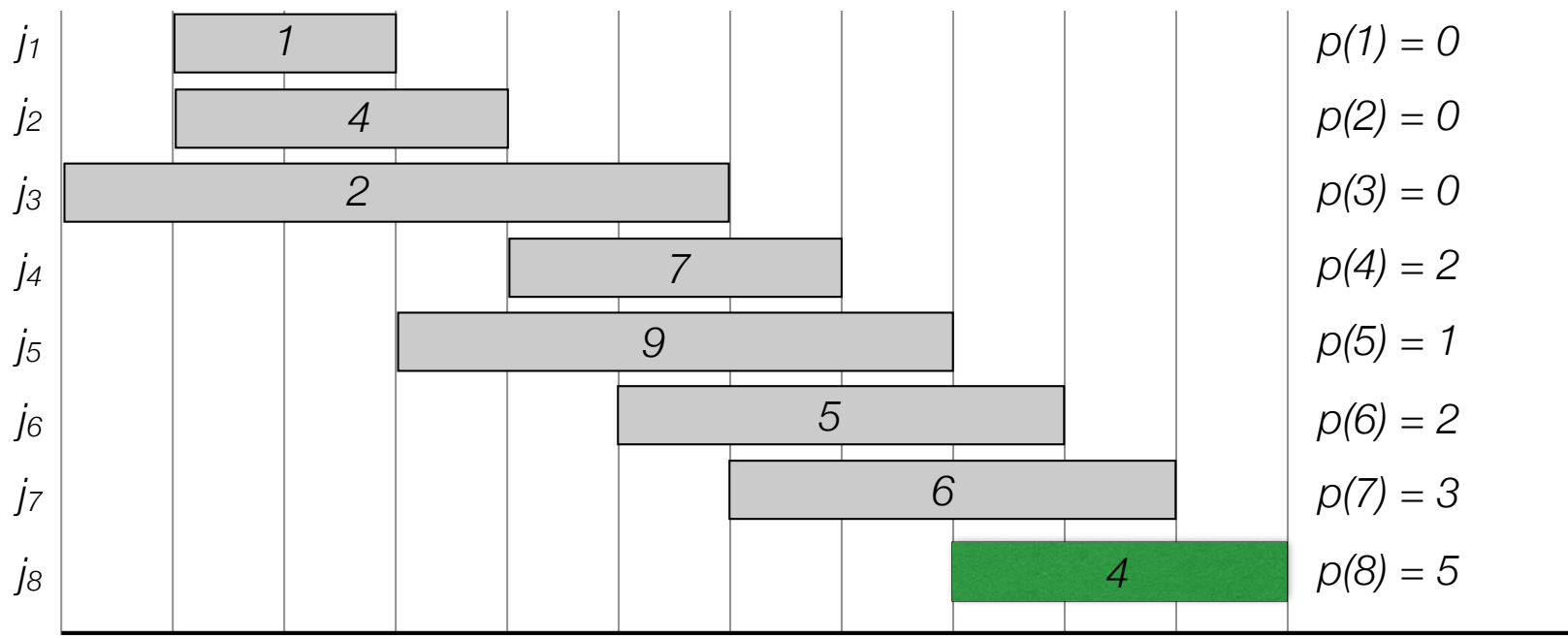
- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j)$  = largest index  $i < j$  such that job  $i$  is compatible with  $j$ .
- Optimal solution OPT:
  - **Case 1.** OPT selects last job



# Weighted interval scheduling

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j)$  = largest index  $i < j$  such that job  $i$  is compatible with  $j$ .
- Optimal solution OPT:
  - **Case 1.** OPT selects last job

$$OPT = v_n + \text{optimal solution to subproblem on } 1, \dots, p(n)$$



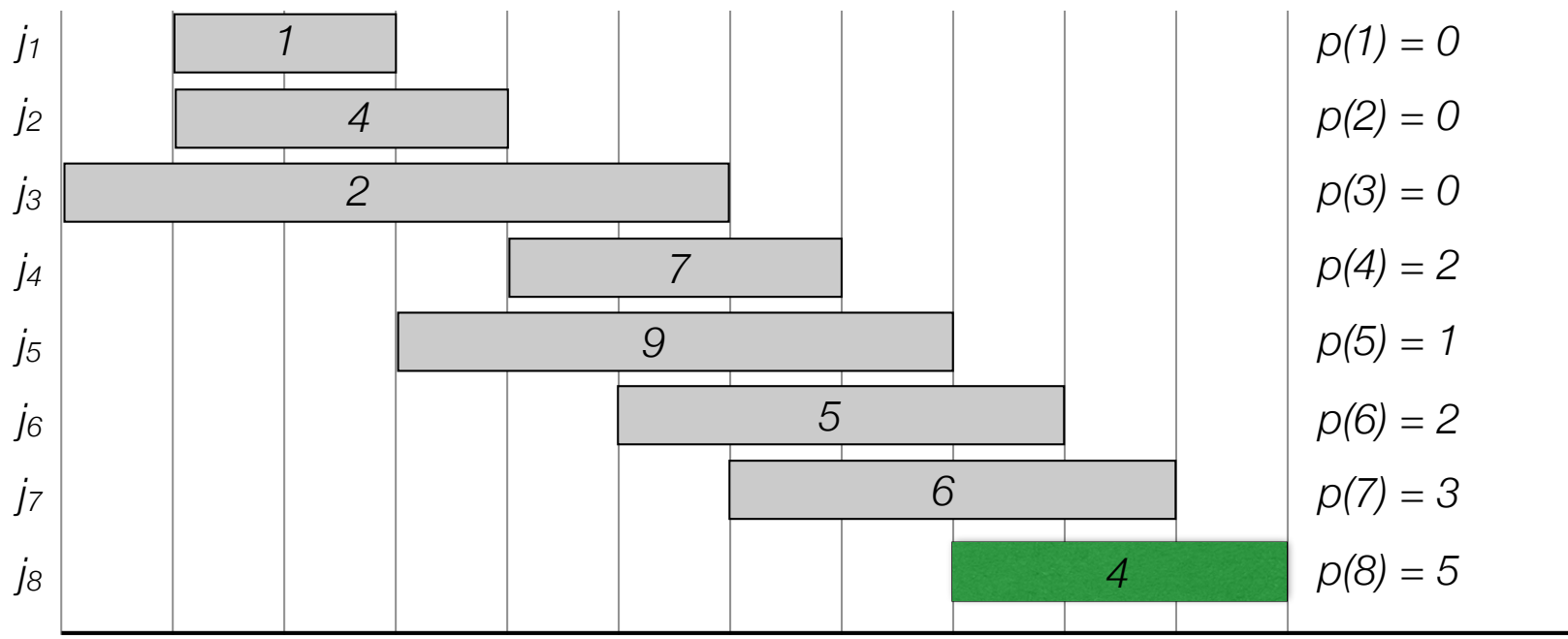
# Weighted interval scheduling

- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j)$  = largest index  $i < j$  such that job  $i$  is compatible with  $j$ .
- Optimal solution OPT:

- **Case 1.** OPT selects last job

$$OPT = v_n + \text{optimal solution to subproblem on } 1, \dots, p(n)$$

- **Case 2.** OPT does not select last job



# Weighted interval scheduling

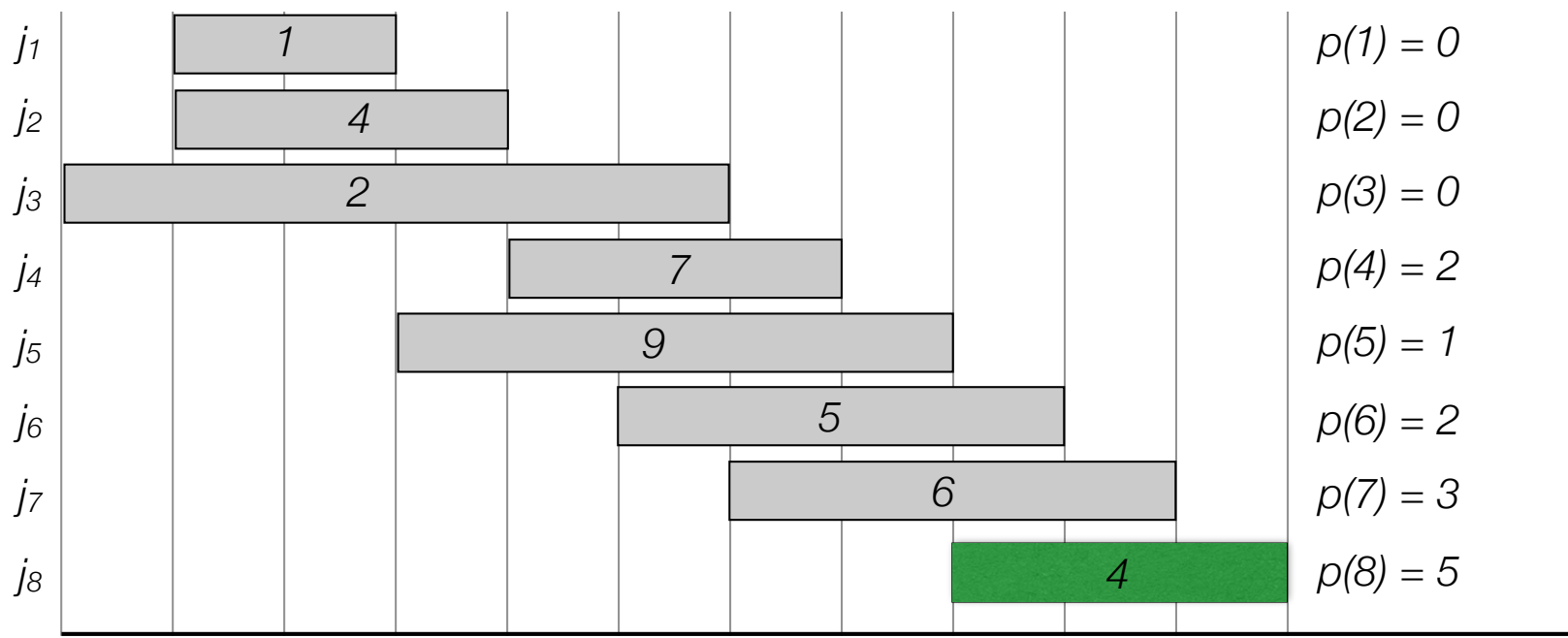
- Label/sort jobs by finishing time:  $f_1 \leq f_2 \leq \dots \leq f_n$
- $p(j)$  = largest index  $i < j$  such that job  $i$  is compatible with  $j$ .
- Optimal solution OPT:

- **Case 1.** OPT selects last job

$$OPT = v_n + \text{optimal solution to subproblem on } 1, \dots, p(n)$$

- **Case 2.** OPT does not select last job

$$OPT = \text{optimal solution to subproblem on } 1, \dots, n-1$$



# Weighted interval scheduling

---

- $OPT(j)$  = value of optimal solution to the problem consisting job requests  $1, 2, \dots, j$ .

- **Case 1.**  $OPT(j)$  selects job  $j$

$$OPT(j) = v_j + \text{optimal solution to subproblem on } 1, \dots, p(j)$$

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

$$OPT = \text{optimal solution to subproblem } 1, \dots, j-1$$

# Weighted interval scheduling

---

- $OPT(j)$  = value of optimal solution to the problem consisting job requests  $1, 2, \dots, j$ .

- **Case 1.**  $OPT(j)$  selects job  $j$

$$OPT(j) = v_j + \text{optimal solution to subproblem on } 1, \dots, p(j)$$

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

$$OPT = \text{optimal solution to subproblem } 1, \dots, j-1$$

- **Recursion:**

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

# Weighted interval scheduling: brute force

---

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

```
Input: n, s[1..n], f[1..n], v[1..n]
```

```
Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
```

```
Compute p[1], p[2], ..., p[n]
```

```
Compute-BruteForce-Opt(n)
```

```
Compute-Brute-Force-Opt(j)
```

```
if j = 0
```

```
    return 0
```

```
else
```

```
    return max(v[j] + Compute-Brute-Force-Opt(p[j]),  
              Compute-Brute-Force-Opt(j-1))
```



# Weighted interval scheduling: brute force

---

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
    return 0
else
    return max(v[j] + Compute-Brute-Force-Opt(p[j]),
              Compute-Brute-Force-Opt(j-1))
```

# Weighted interval scheduling: brute force

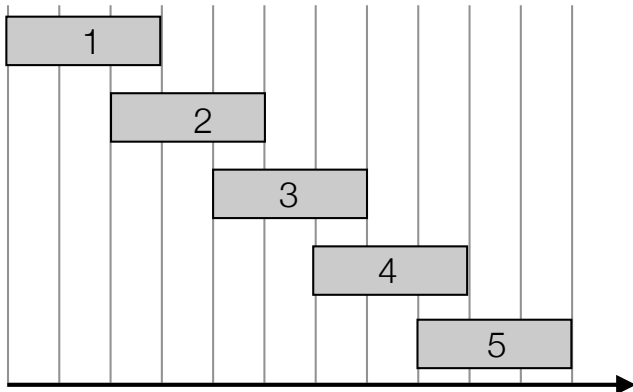
---

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



# Weighted interval scheduling: brute force

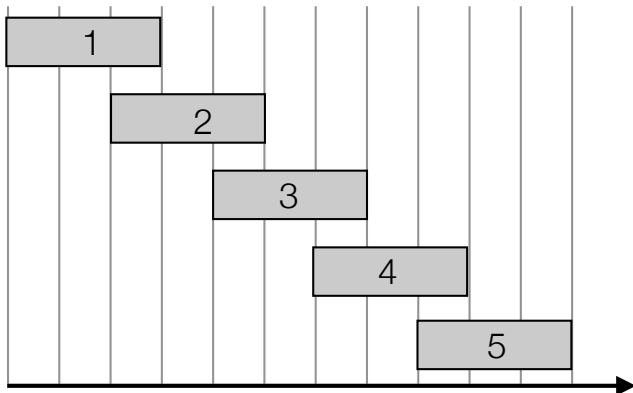
$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{otherwise} \end{cases}$$

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```

5



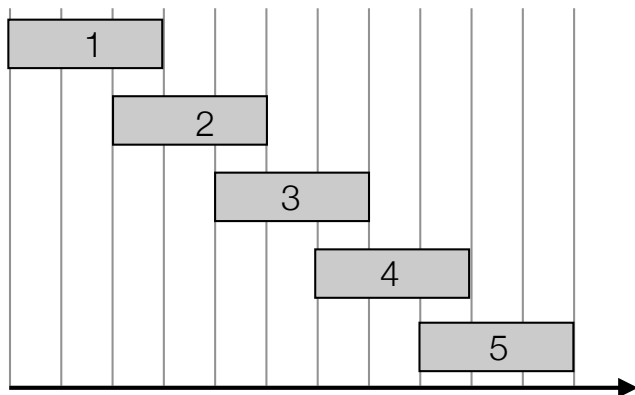
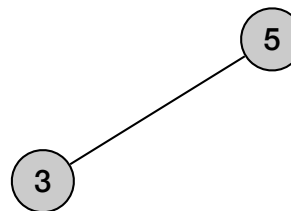
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



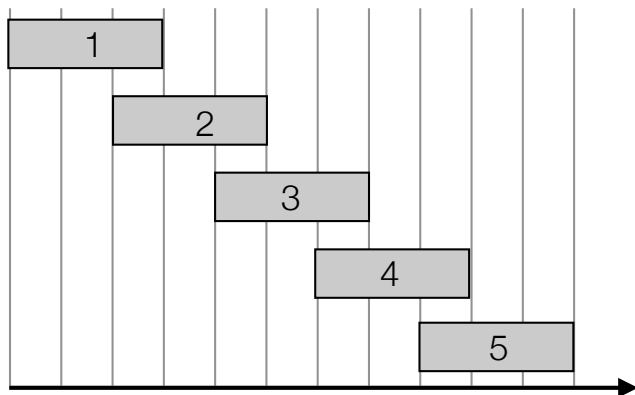
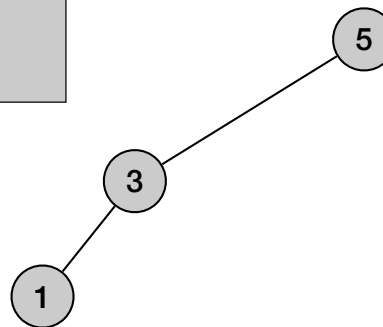
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



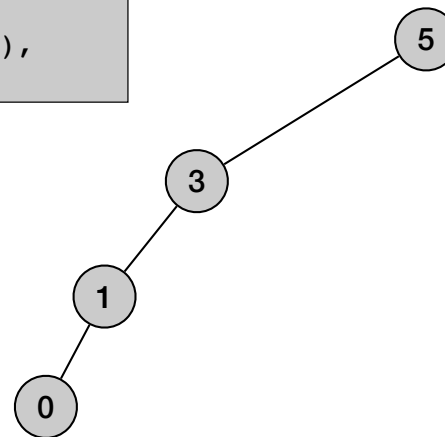
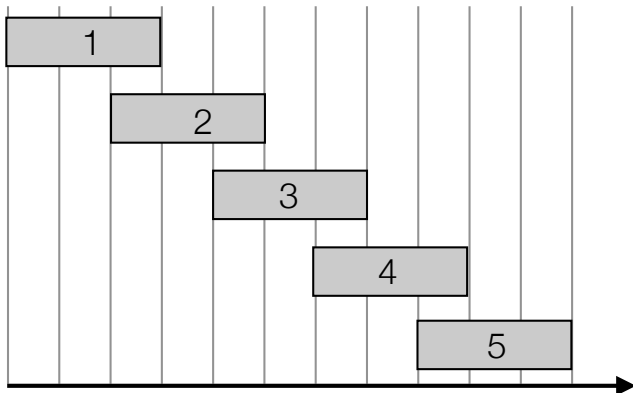
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



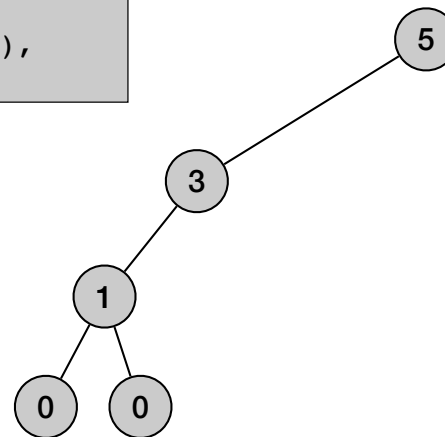
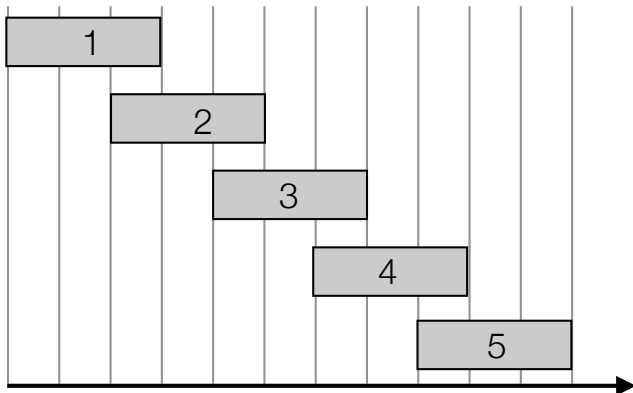
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



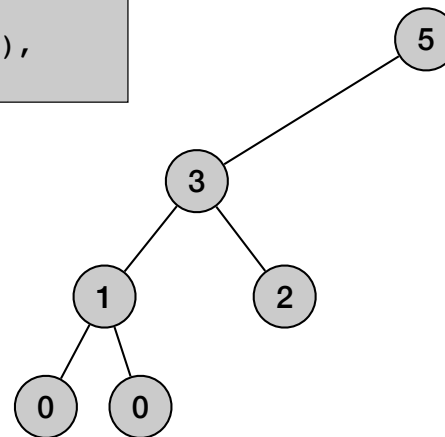
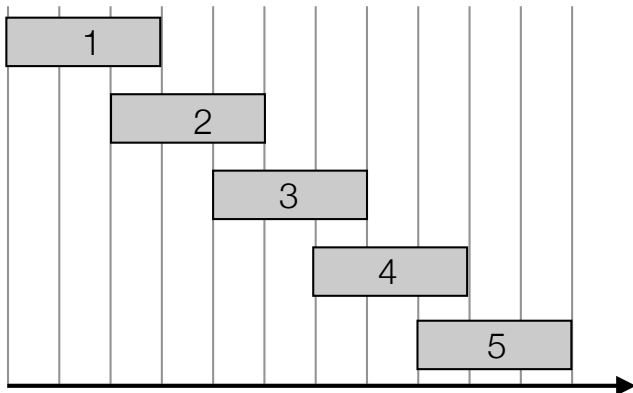
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```





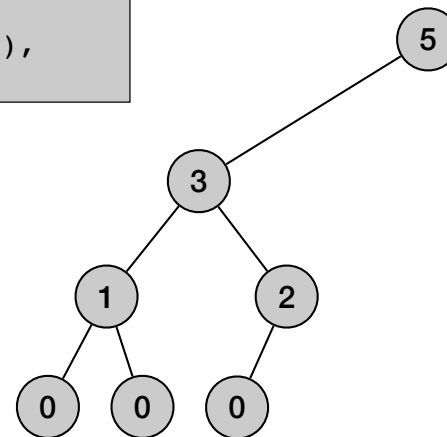
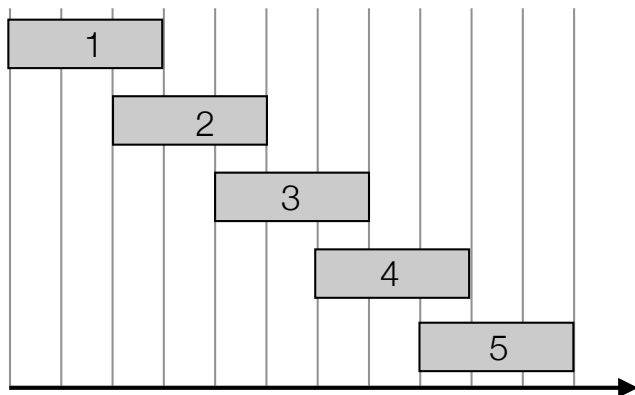
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-Brute-Force-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



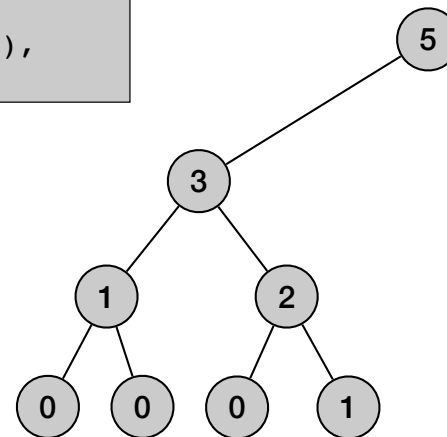
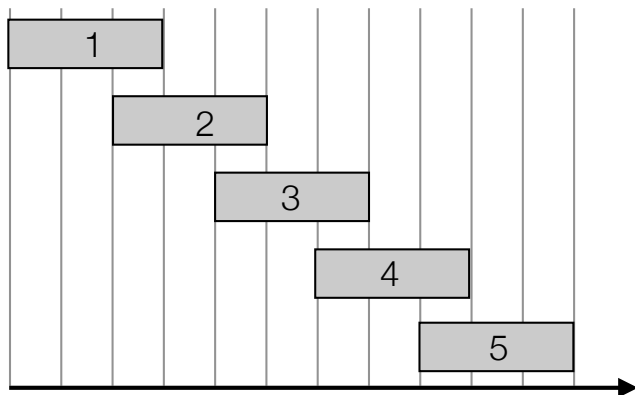
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



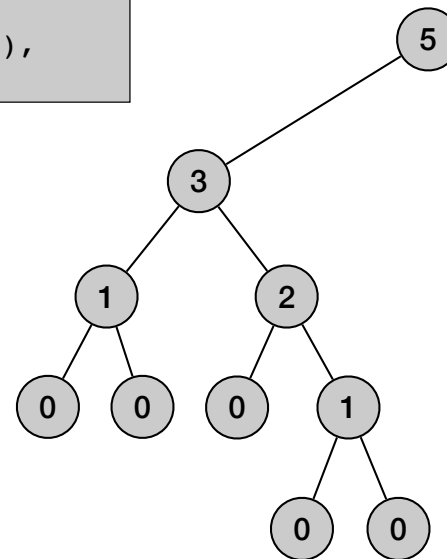
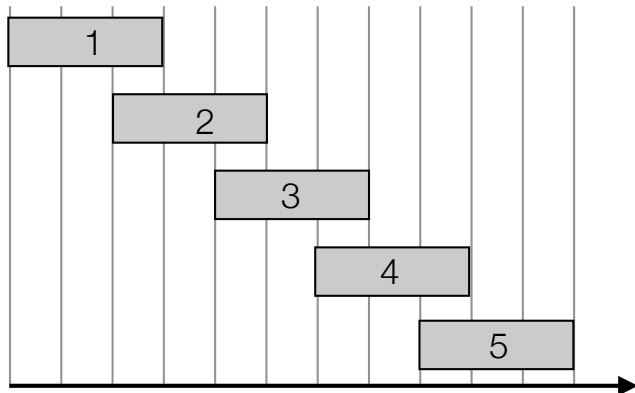
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



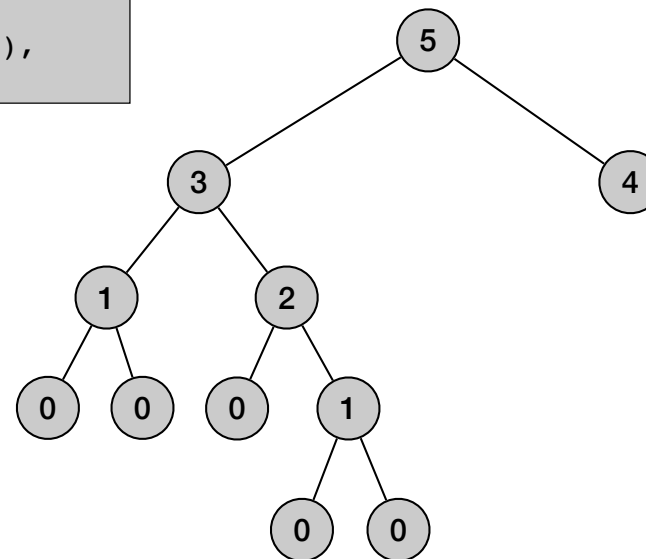
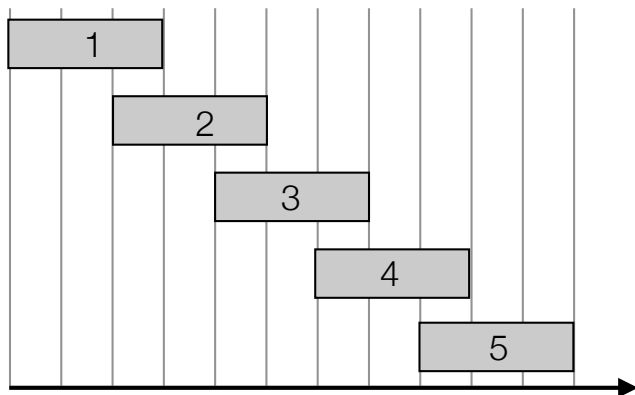
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



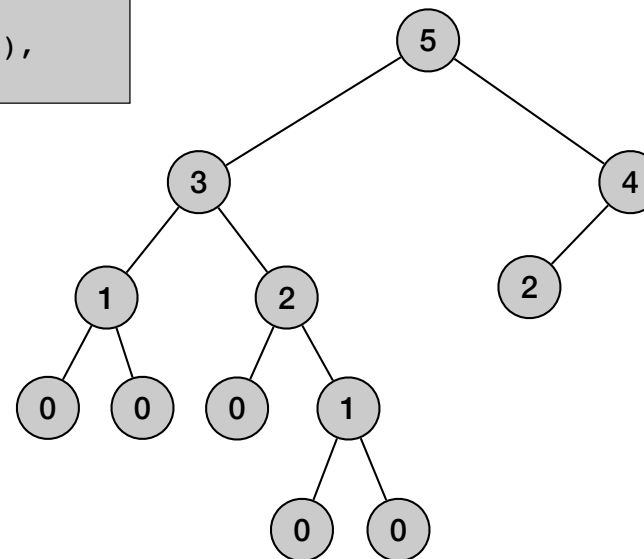
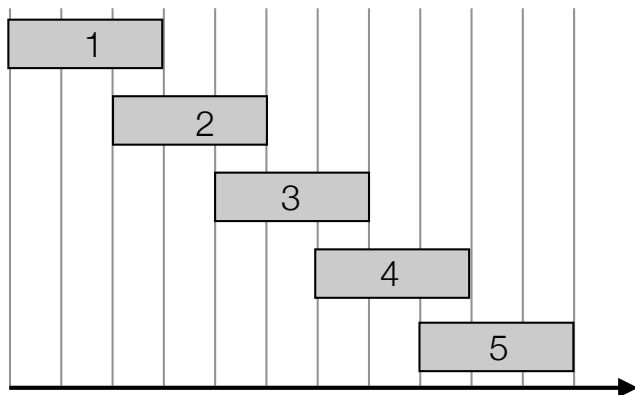
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



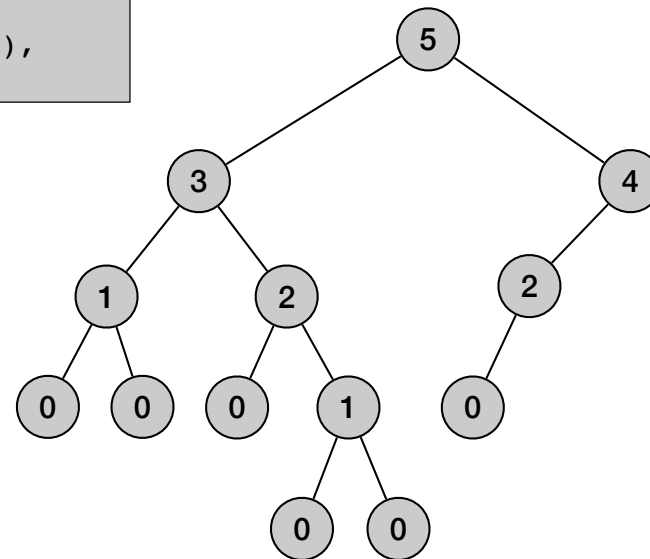
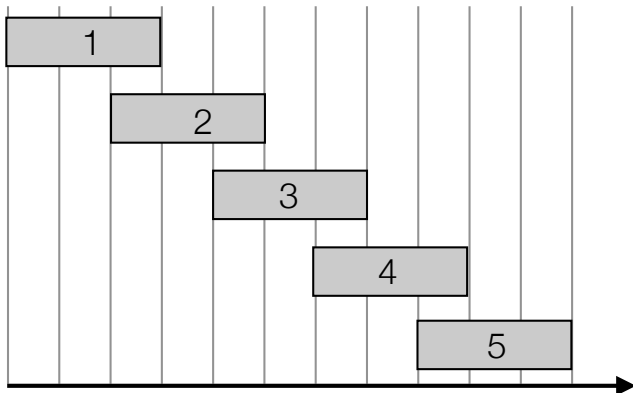
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



# Weighted interval scheduling: brute force

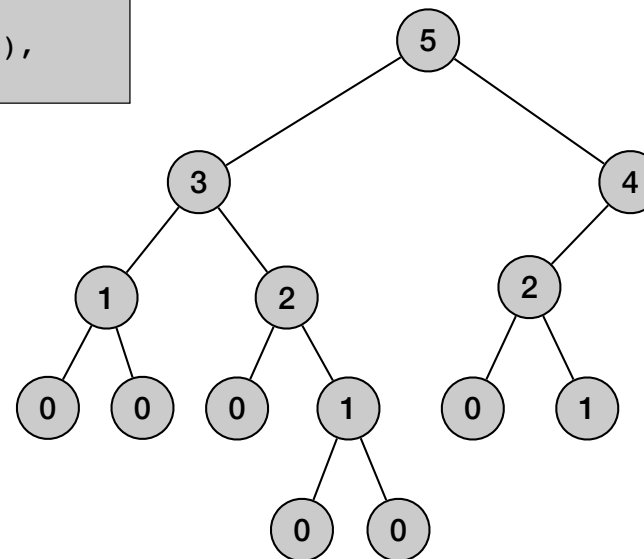
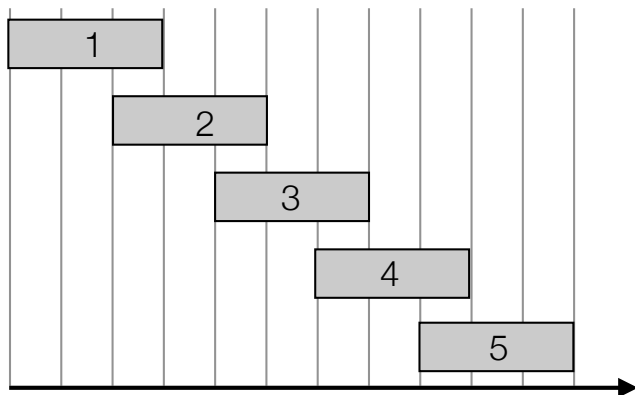
$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{otherwise} \end{cases}$$

```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
             Compute-Brute-Force-Opt(j-1))
    
```



# Weighted interval scheduling: brute force

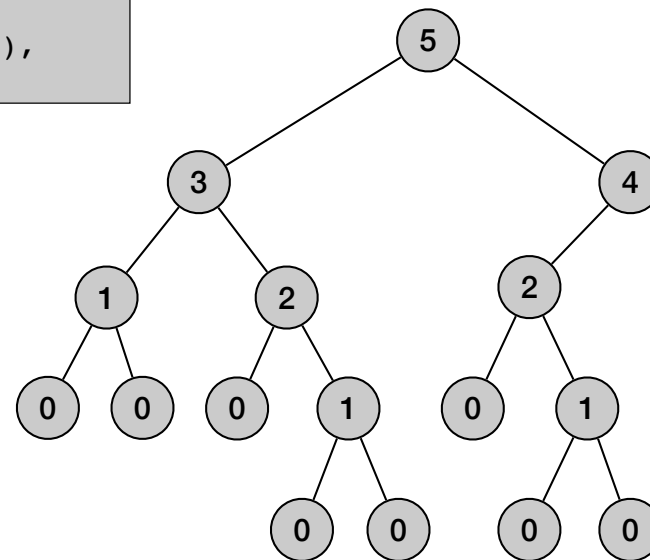
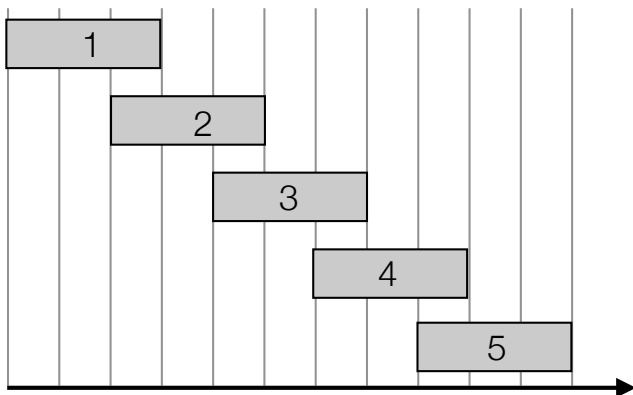
$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{otherwise} \end{cases}$$

```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
    
```





# Weighted interval scheduling: brute force

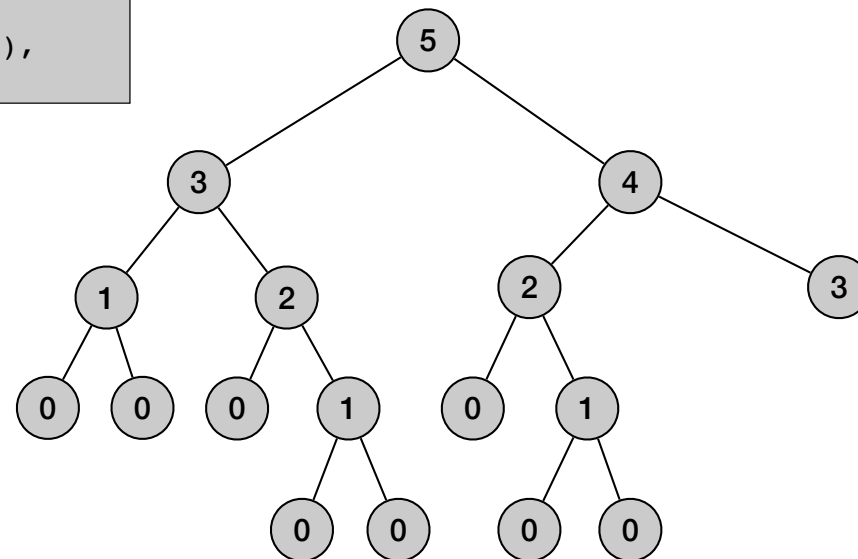
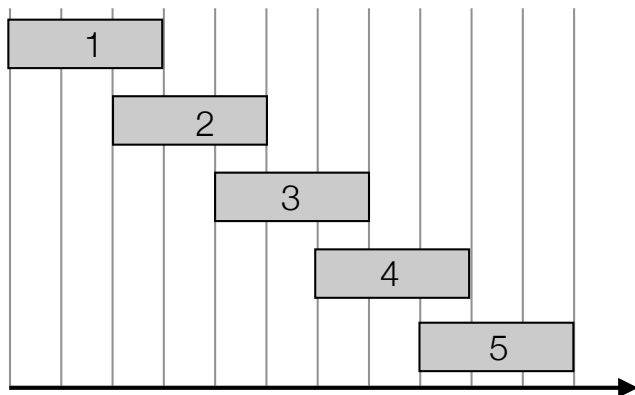
$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{otherwise} \end{cases}$$

```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
             Compute-Brute-Force-Opt(j-1))
    
```



# Weighted interval scheduling: brute force

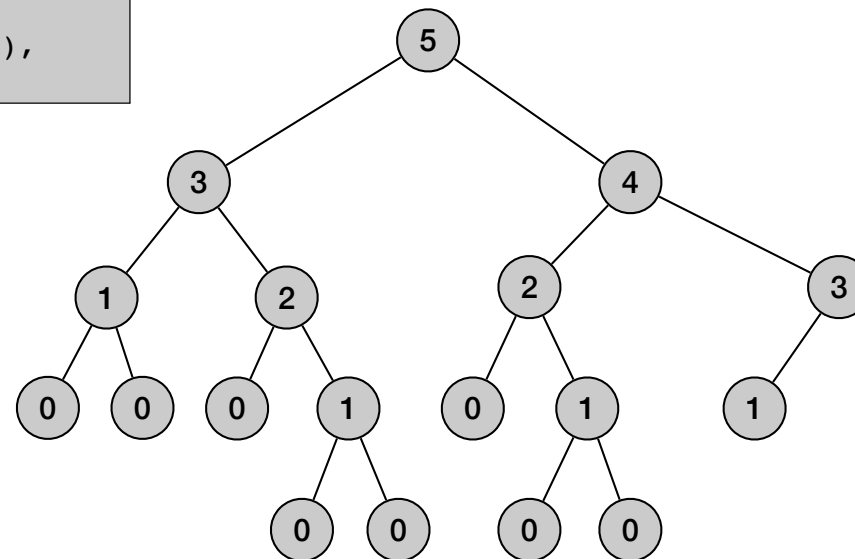
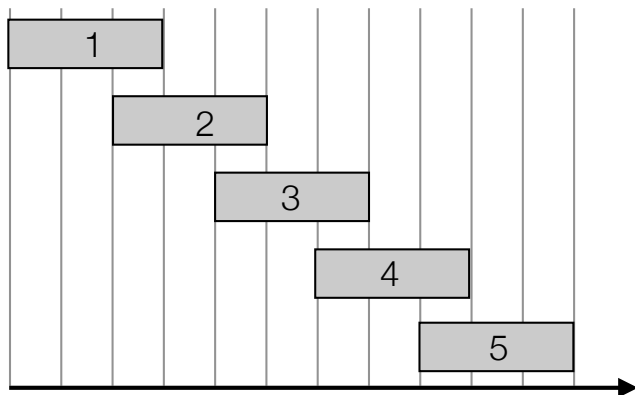
$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{otherwise} \end{cases}$$

```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
    
```



# Weighted interval scheduling: brute force

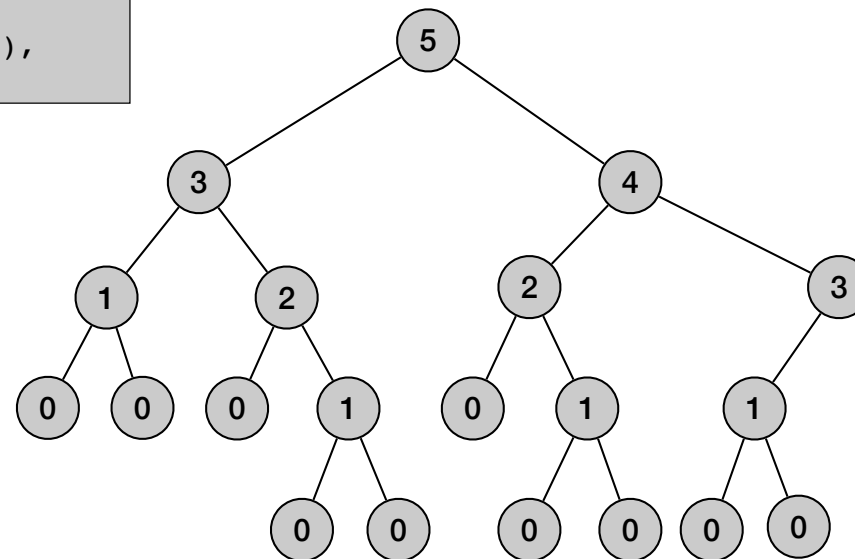
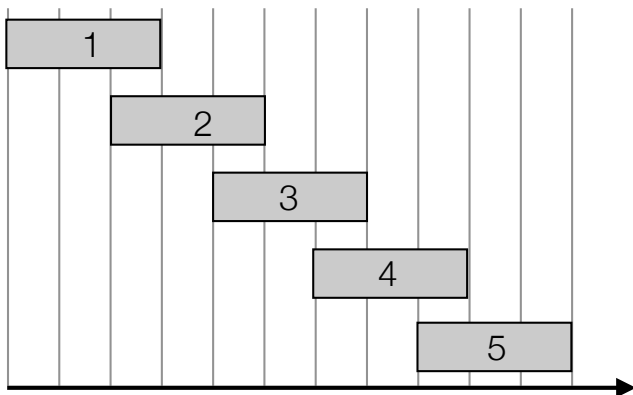
$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{otherwise} \end{cases}$$

```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
             Compute-Brute-Force-Opt(j-1))
    
```



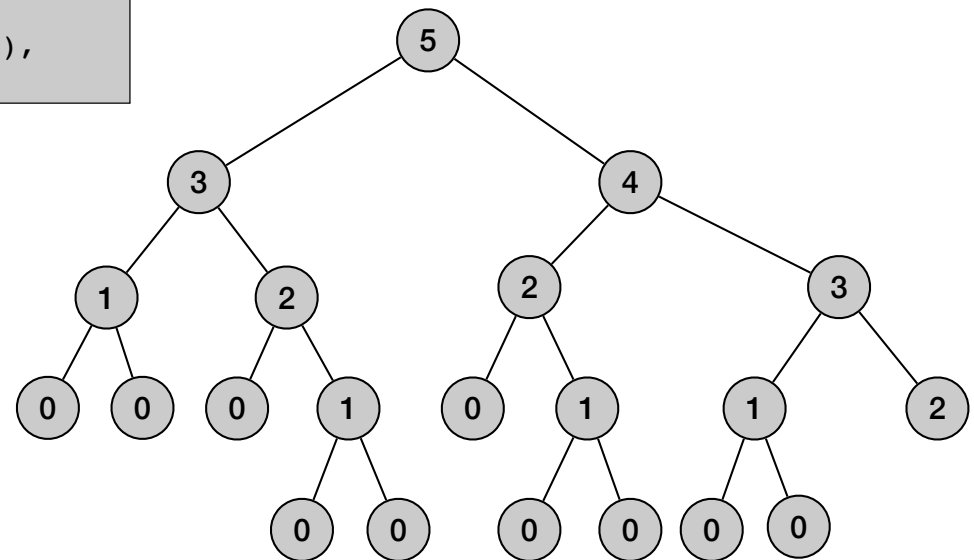
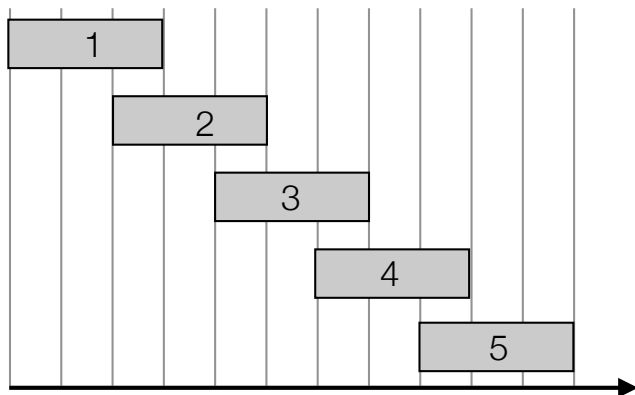
# Weighted interval scheduling: brute force

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

```
Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
```



# Weighted interval scheduling: brute force

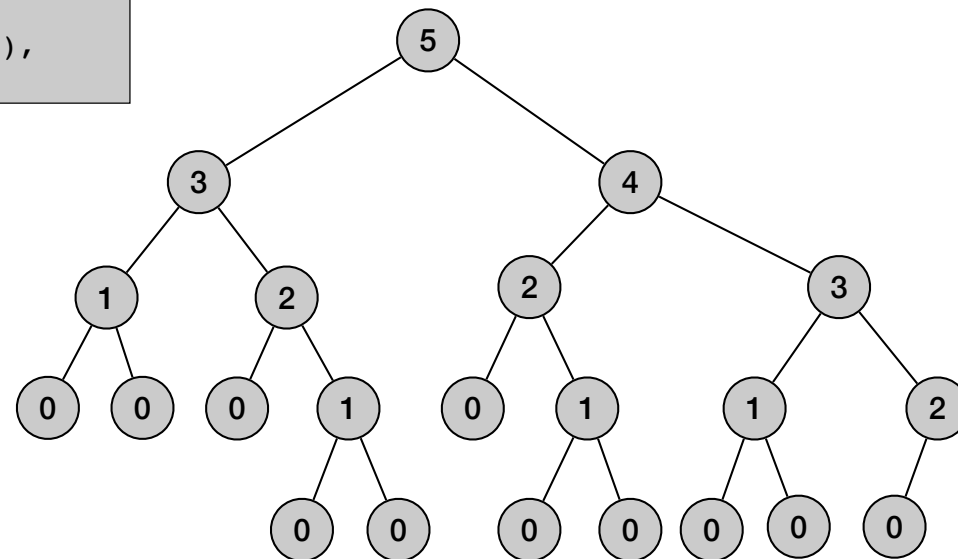
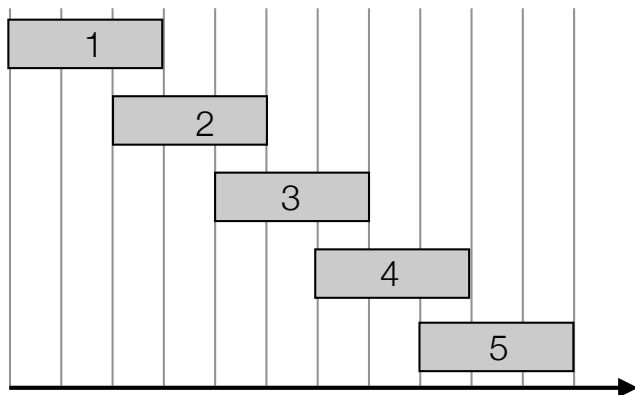
$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{otherwise} \end{cases}$$

```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
             Compute-Brute-Force-Opt(j-1))
    
```



# Weighted interval scheduling: brute force

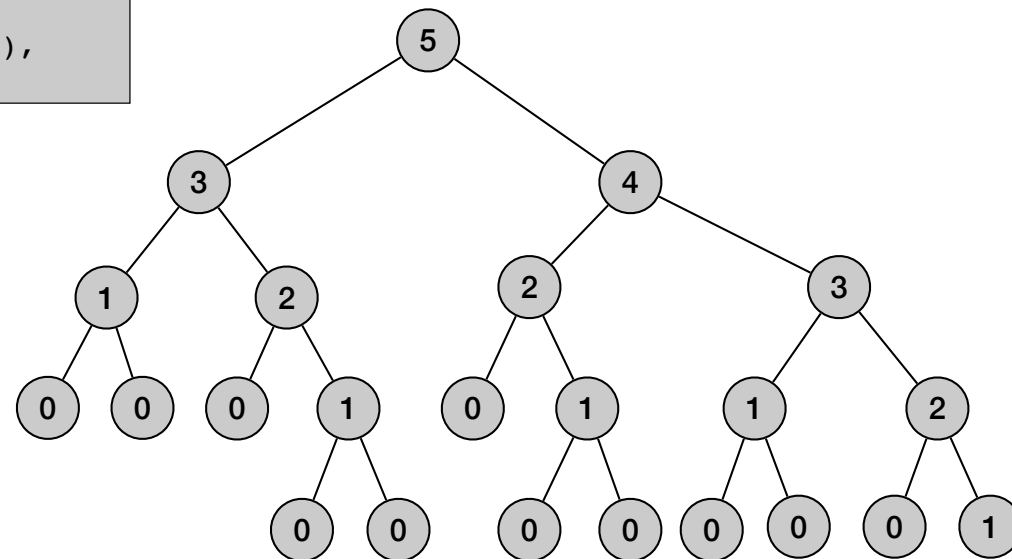
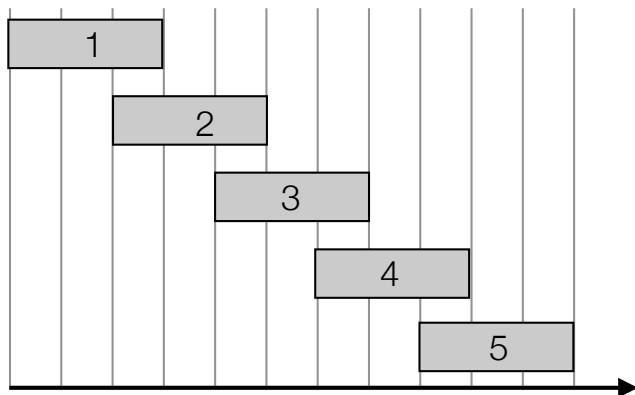
$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{otherwise} \end{cases}$$

```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
             Compute-Brute-Force-Opt(j-1))
    
```



# Weighted interval scheduling: brute force

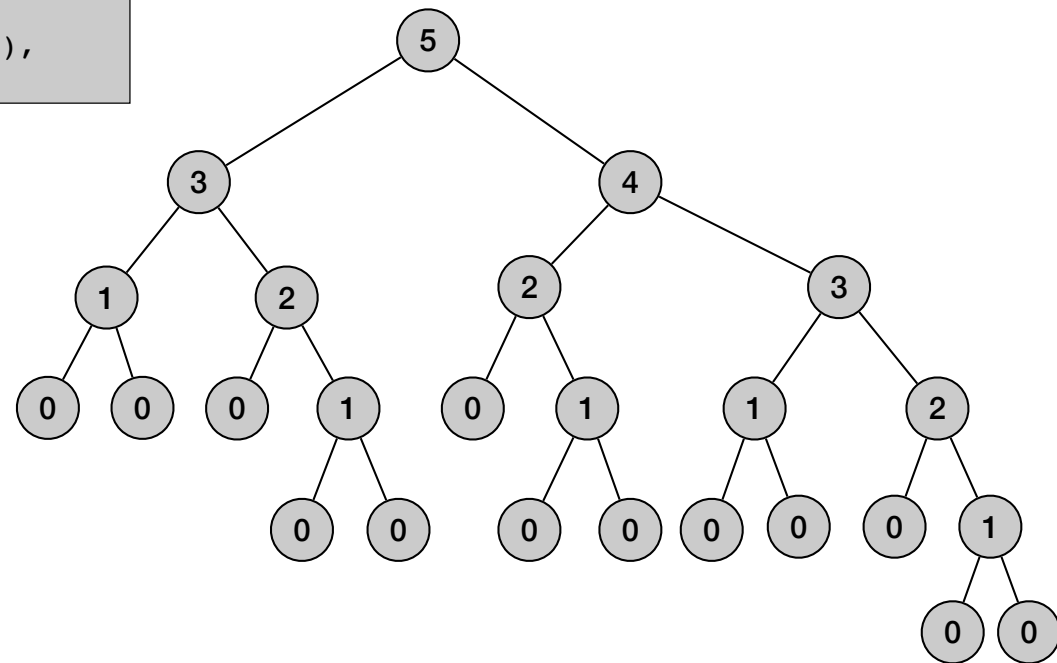
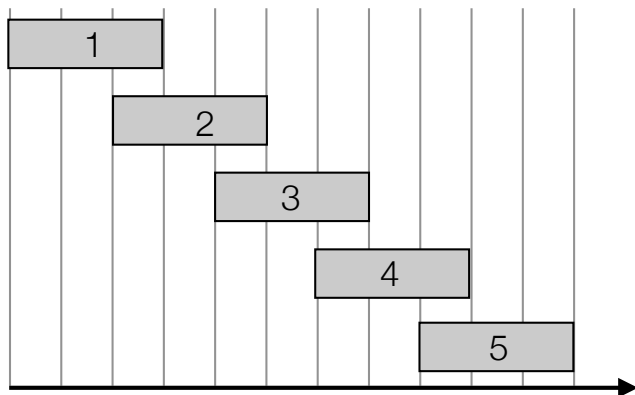
$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{otherwise} \end{cases}$$

```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
             Compute-Brute-Force-Opt(j-1))
    
```



# Weighted interval scheduling: brute force

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

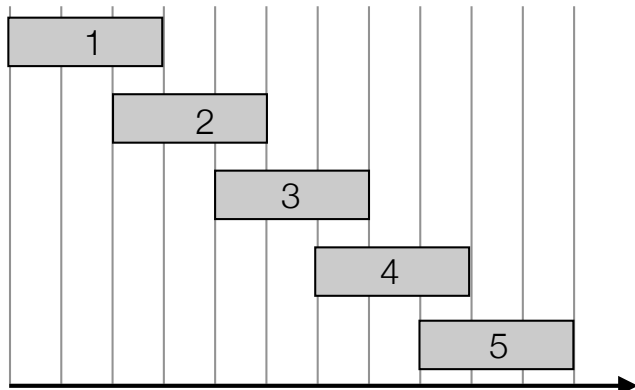
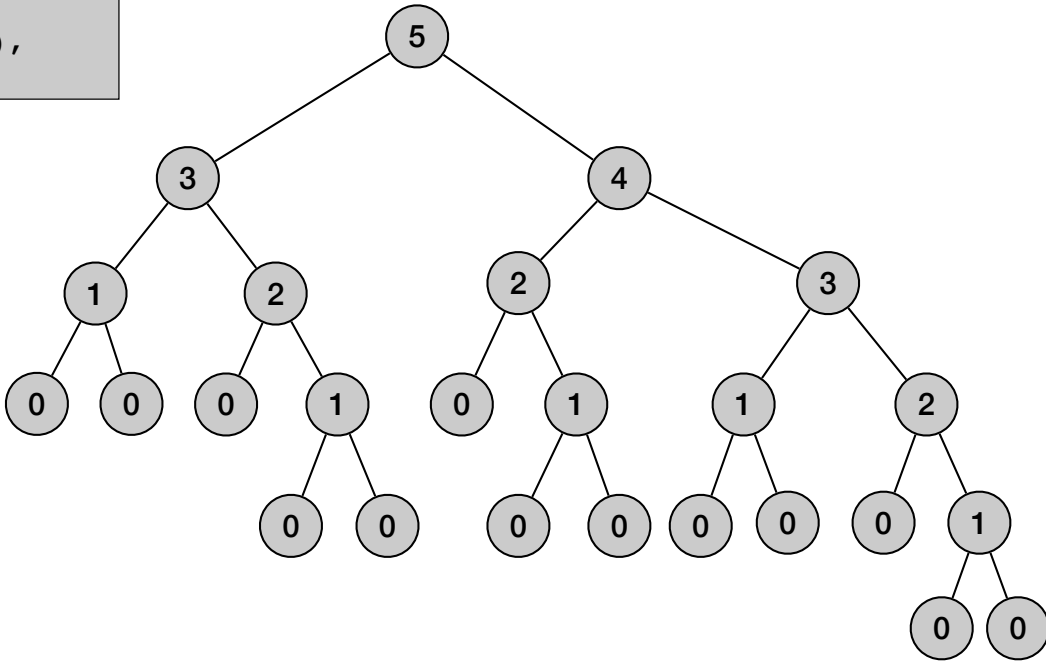
```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
    
```

time  $\Theta(2^n)$





# Weighted interval scheduling: brute force

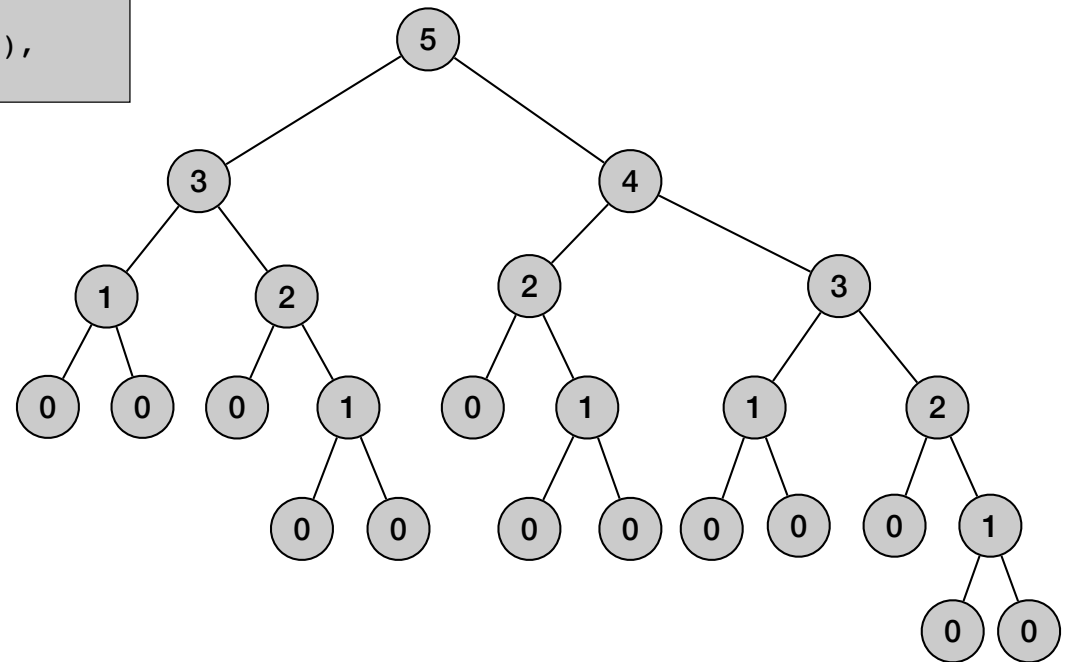
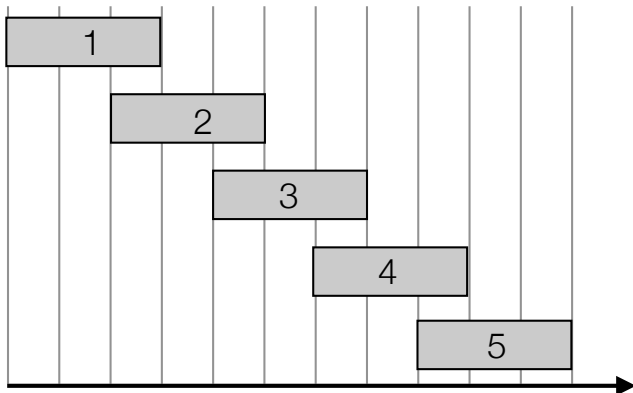
$$OPT(j) = \begin{cases} 0 & \text{if } j = 0 \\ \max\{v_j + OPT(p(j)), OPT(j-1)\} & \text{otherwise} \end{cases}$$

```

Input: n, s[1..n], f[1..n], v[1..n]
Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]
Compute-BruteForce-Opt(n)

Compute-Brute-Force-Opt(j)
if j = 0
  return 0
else
  return max(v[j] + Compute-Brute-Force-Opt(p[j]),
            Compute-Brute-Force-Opt(j-1))
    
```

time  $\Theta(2^n)$



# Weighted interval scheduling: memoization

```
Input: n, s[1..n], f[1..n], v[1..n]
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

```
Compute p[1], p[2], ..., p[n]
```

```
for j=1 to n
```

```
    M[j] = null
```

```
M[0] = 0.
```

```
Compute-Memoized-Opt(n)
```

```
Compute-Memoized-Opt(j)
```

```
if M[j] is empty
```

```
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),  
              Compute-Memoized-Opt(j-1))
```

```
return M[j]
```

# Weighted interval scheduling: memoization

**Input:**  $n$ ,  $s[1..n]$ ,  $f[1..n]$ ,  $v[1..n]$

**Sort** jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

**for**  $j=1$  **to**  $n$

$M[j] = \text{null}$

$M[0] = 0$ .

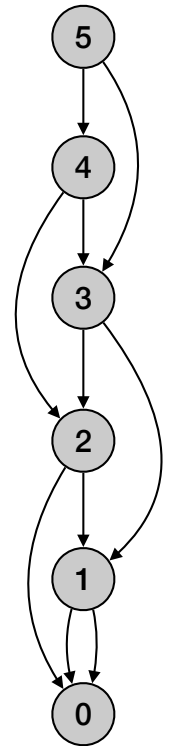
**Compute-Memoized-Opt**( $n$ )

**Compute-Memoized-Opt**( $j$ )

**if**  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
         $\text{Compute-Memoized-Opt}(j-1))$

**return**  $M[j]$



# Weighted interval scheduling: memoization

**Input:**  $n$ ,  $s[1..n]$ ,  $f[1..n]$ ,  $v[1..n]$

**Sort** jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

**for**  $j=1$  **to**  $n$

$M[j] = \text{null}$

$M[0] = 0$ .

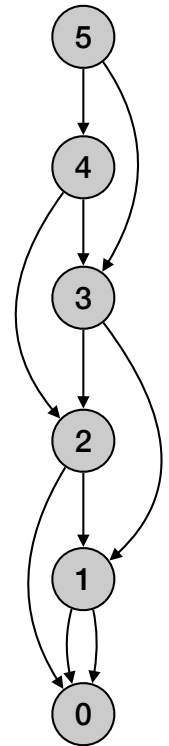
**Compute-Memoized-Opt**( $n$ )

**Compute-Memoized-Opt**( $j$ )

**if**  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
         $\text{Compute-Memoized-Opt}(j-1))$

**return**  $M[j]$



- Running time  $O(n \log n)$ :

# Weighted interval scheduling: memoization

**Input:**  $n$ ,  $s[1..n]$ ,  $f[1..n]$ ,  $v[1..n]$

**Sort** jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

**for**  $j=1$  **to**  $n$

$M[j] = \text{null}$

$M[0] = 0$ .

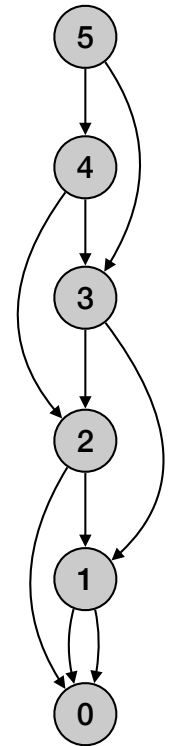
**Compute-Memoized-Opt**( $n$ )

**Compute-Memoized-Opt**( $j$ )

**if**  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
         $\text{Compute-Memoized-Opt}(j-1))$

**return**  $M[j]$



- Running time  $O(n \log n)$ :
  - Sorting takes  $O(n \log n)$  time.

# Weighted interval scheduling: memoization

**Input:**  $n$ ,  $s[1..n]$ ,  $f[1..n]$ ,  $v[1..n]$

**Sort** jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

**for**  $j=1$  **to**  $n$

$M[j] = \text{null}$

$M[0] = 0$ .

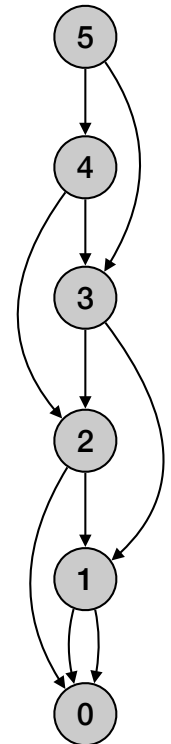
**Compute-Memoized-Opt**( $n$ )

**Compute-Memoized-Opt**( $j$ )

**if**  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                   $\text{Compute-Memoized-Opt}(j-1))$

**return**  $M[j]$



- Running time  $O(n \log n)$ :
  - Sorting takes  $O(n \log n)$  time.
  - Computing  $p(n)$ :  $O(n \log n)$  - use  $\log n$  time to find each  $p(i)$ .

# Weighted interval scheduling: memoization

```
Input: n, s[1..n], f[1..n], v[1..n]
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

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

```
for j=1 to n
```

```
    M[j] = null
```

```
M[0] = 0.
```

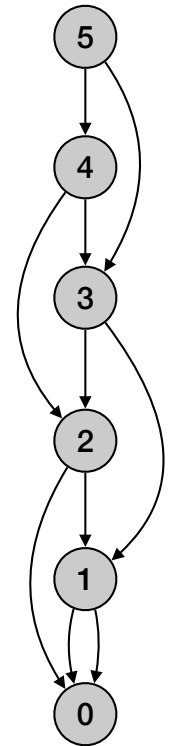
```
Compute-Memoized-Opt(n)
```

```
Compute-Memoized-Opt(j)
```

```
if M[j] is empty
```

```
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),  
              Compute-Memoized-Opt(j-1))
```

```
return M[j]
```



- Running time  $O(n \log n)$ :
  - Sorting takes  $O(n \log n)$  time.
  - Computing  $p(n)$ :  $O(n \log n)$  - use  $\log n$  time to find each  $p(i)$ .
  - Each subproblem solved once.

# Weighted interval scheduling: memoization

```
Input: n, s[1..n], f[1..n], v[1..n]
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

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

```
for j=1 to n
```

```
    M[j] = null
```

```
M[0] = 0.
```

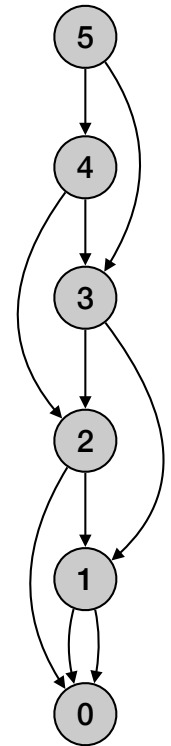
```
Compute-Memoized-Opt(n)
```

```
Compute-Memoized-Opt(j)
```

```
if M[j] is empty
```

```
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),  
              Compute-Memoized-Opt(j-1))
```

```
return M[j]
```



- Running time  $O(n \log n)$ :
  - Sorting takes  $O(n \log n)$  time.
  - Computing  $p(n)$ :  $O(n \log n)$  - use  $\log n$  time to find each  $p(i)$ .
  - Each subproblem solved once.
  - Time to solve a subproblem constant.



# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

**Sort** jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

**for**  $j=1$  **to**  $n$

$M[j] = \text{null}$

$M[0] = 0.$

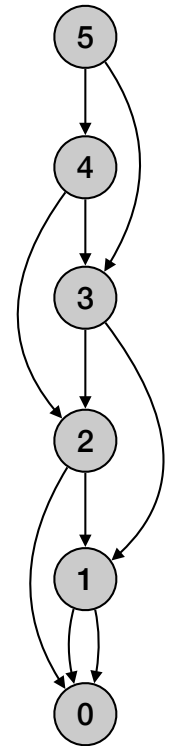
**Compute-Memoized-Opt**( $n$ )

**Compute-Memoized-Opt**( $j$ )

**if**  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
         $\text{Compute-Memoized-Opt}(j-1))$

**return**  $M[j]$



- Running time  $O(n \log n)$ :
  - Sorting takes  $O(n \log n)$  time.
  - Computing  $p(n)$ :  $O(n \log n)$  - use  $\log n$  time to find each  $p(i)$ .
  - Each subproblem solved once.
  - Time to solve a subproblem constant.
- Space  $O(n)$

# Weighted interval scheduling: memoization

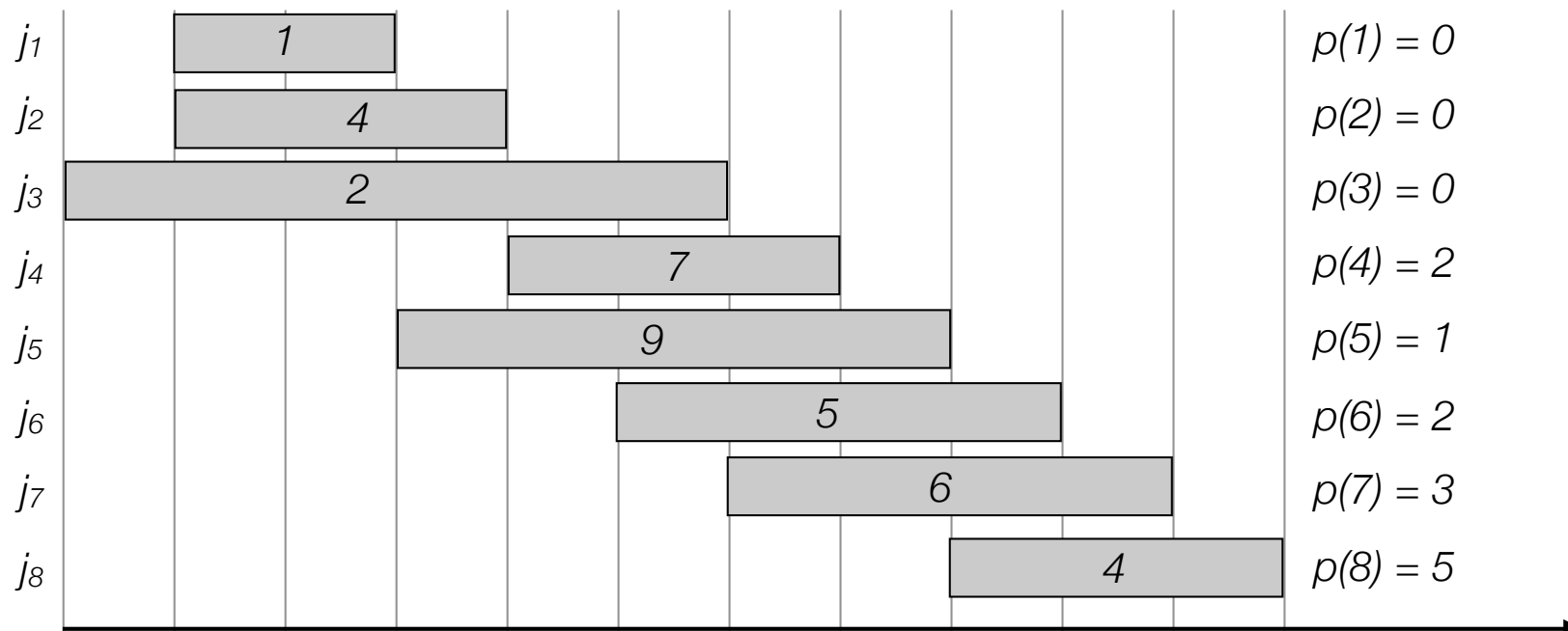
```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
Compute  $p[1], p[2], \dots, p[n]$ 

for j=1 to n
    M[j] = empty
M[0] = 0.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
              Compute-Memoized-Opt(j-1))
return M[j]
    
```



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

# Weighted interval scheduling: memoization

8

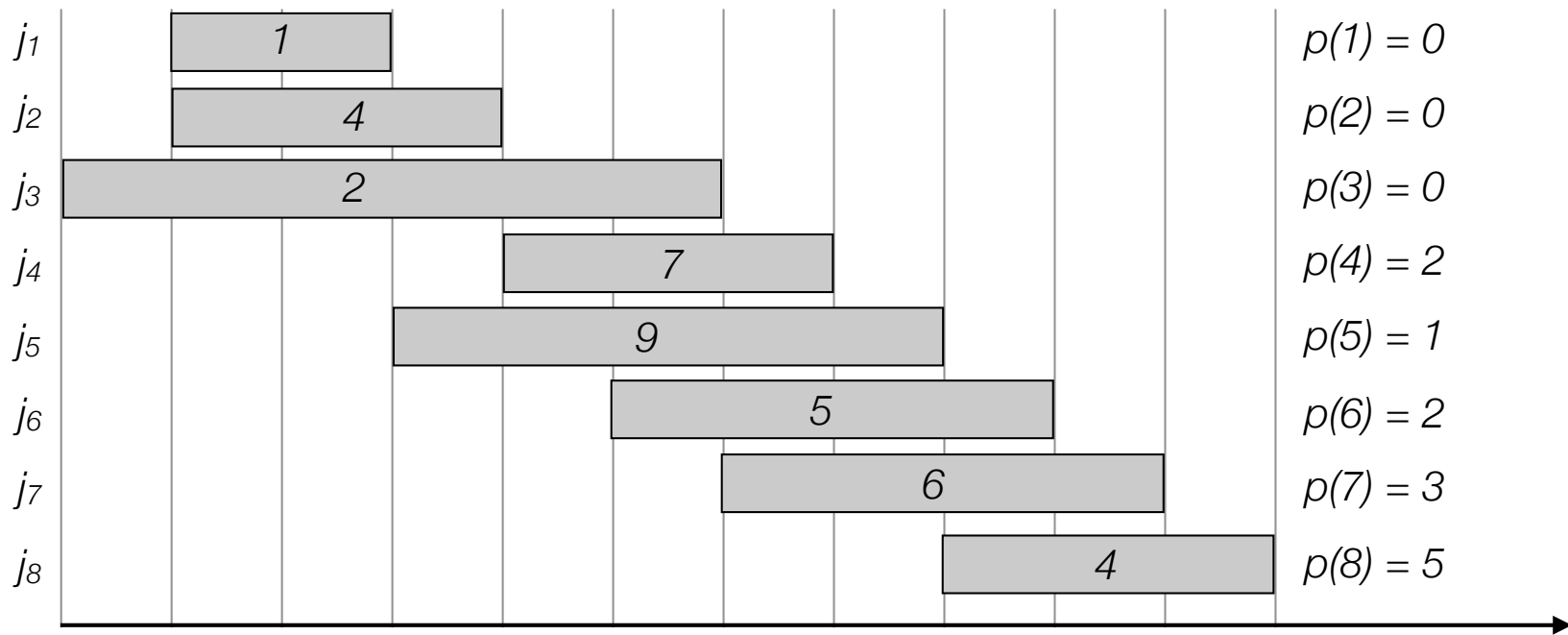
```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time 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.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
               Compute-Memoized-Opt(j-1))
return M[j]
    
```



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

# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

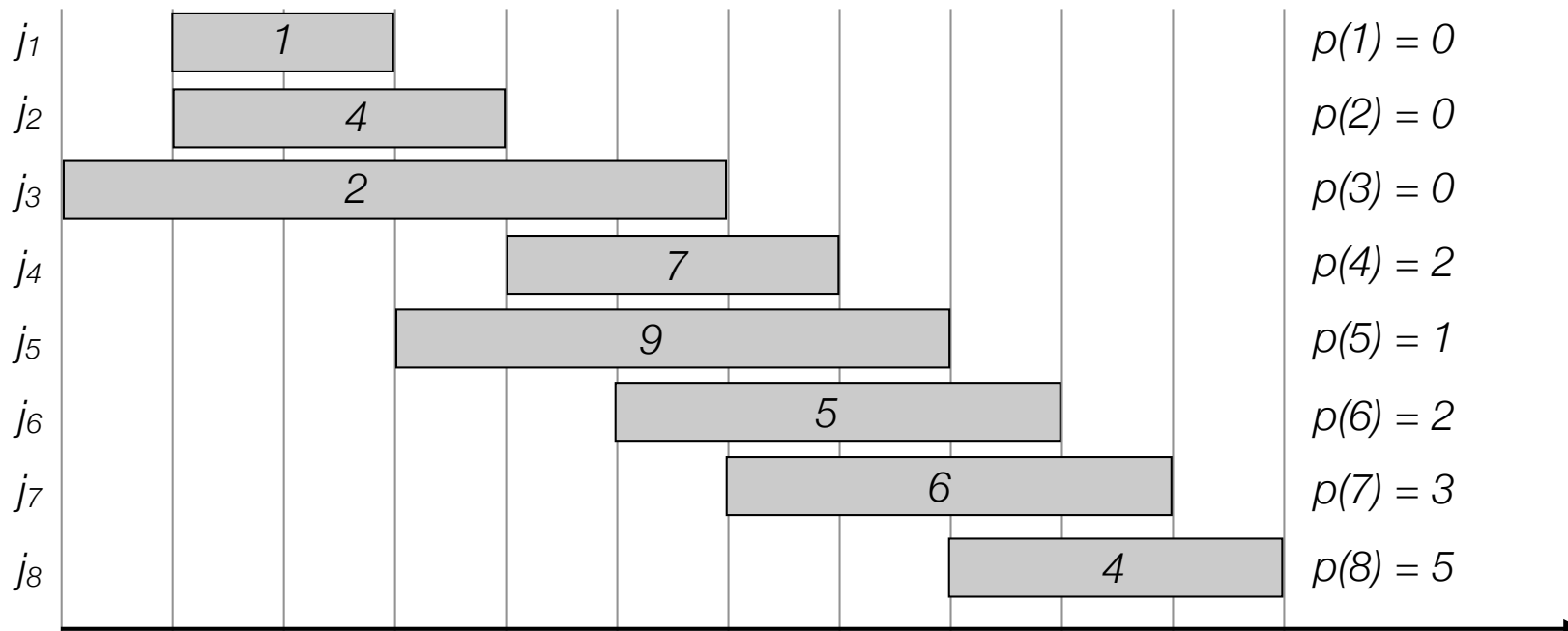
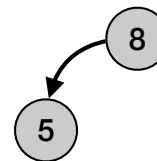
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

    if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

    return  $M[j]$



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

# Weighted interval scheduling: memoization

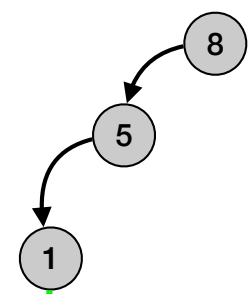
```

Input: n, s[1..n], f[1..n], v[1..n]

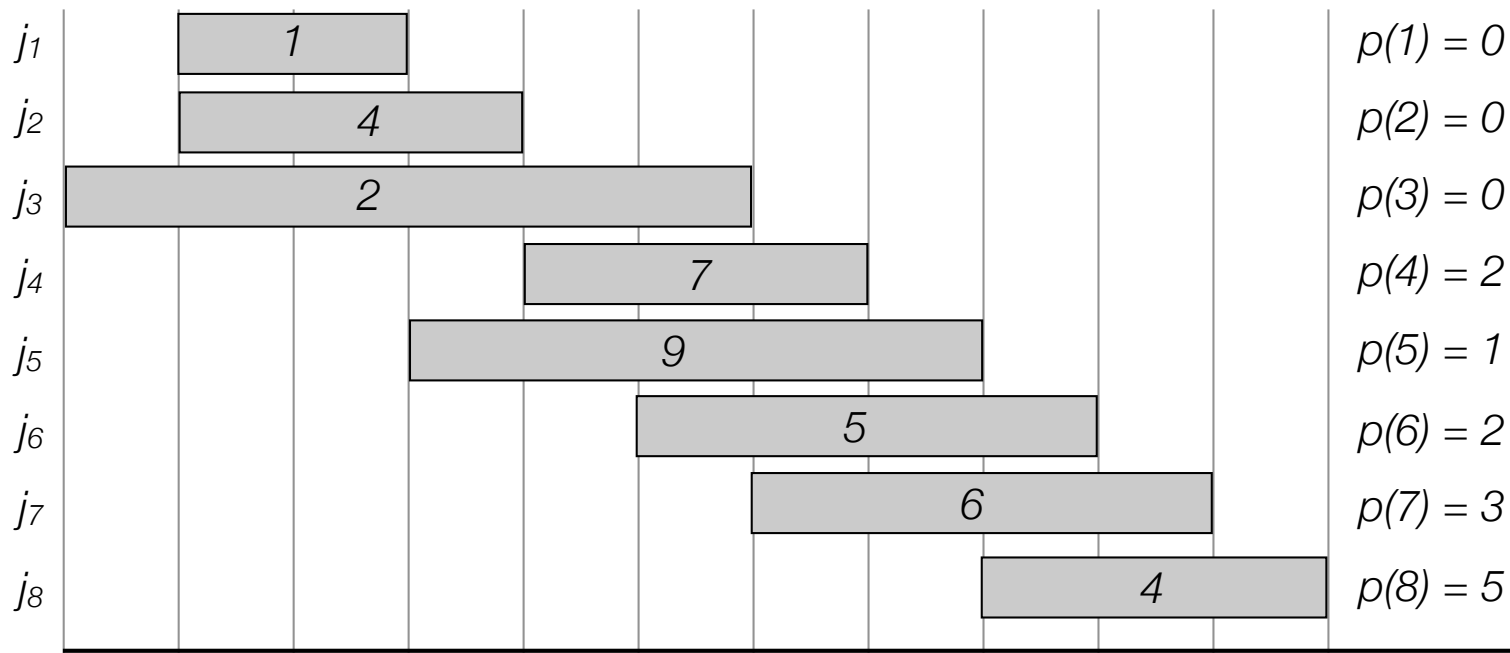
Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]

for j=1 to n
    M[j] = empty
M[0] = 0.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
               Compute-Memoized-Opt(j-1))
return M[j]
    
```



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



# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

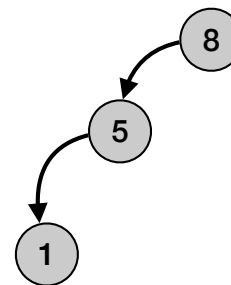
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

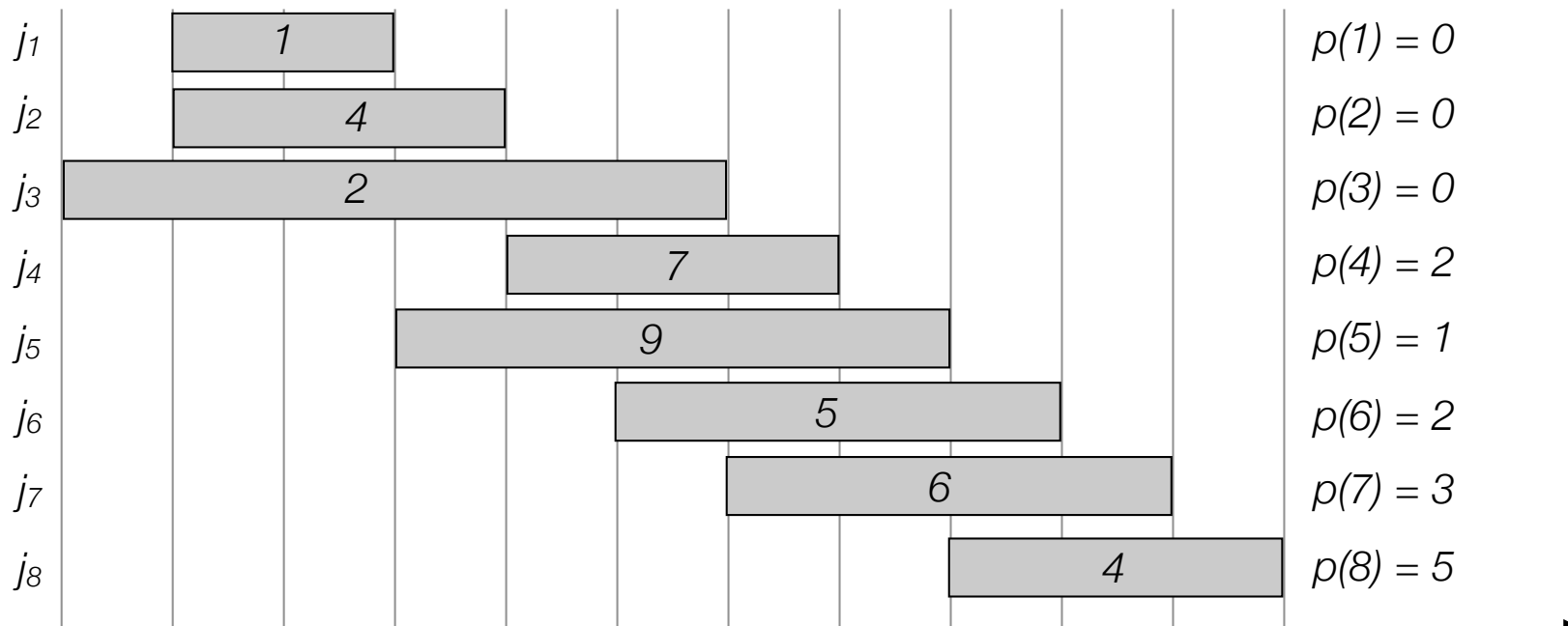
        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



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



# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

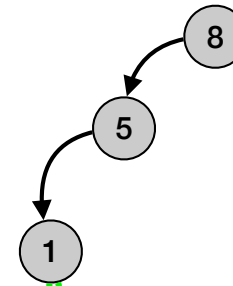
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

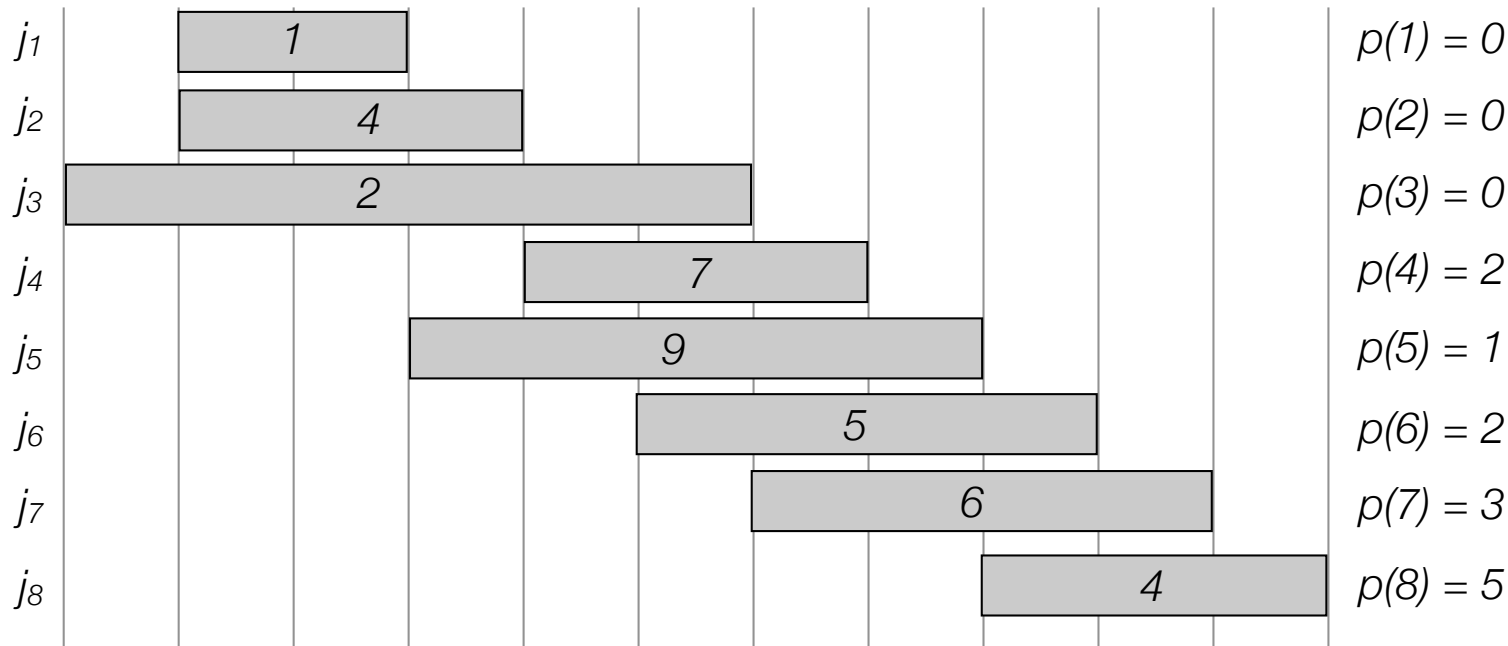
        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



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



# Weighted interval scheduling: memoization

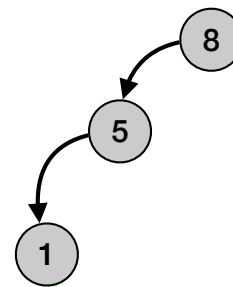
```

Input: n, s[1..n], f[1..n], v[1..n]

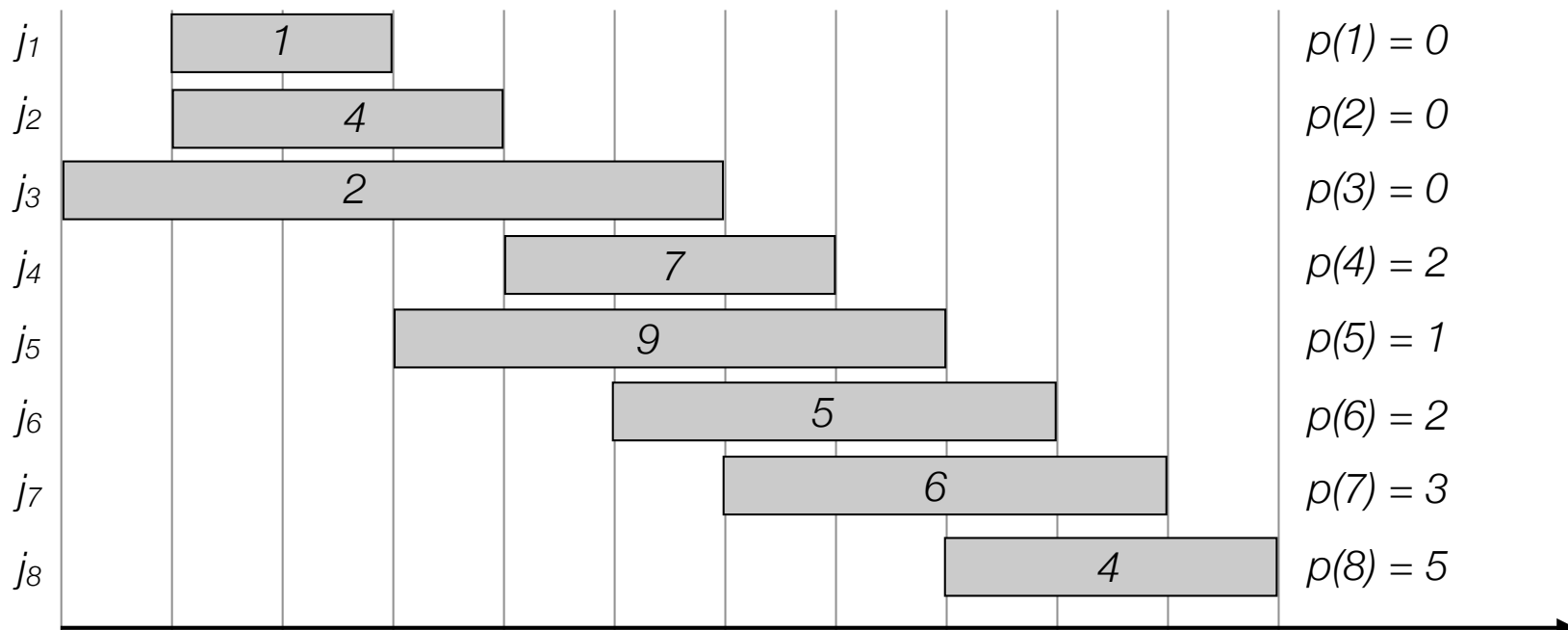
Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]

for j=1 to n
    M[j] = empty
M[0] = 0.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
               Compute-Memoized-Opt(j-1))
return M[j]
    
```



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





# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

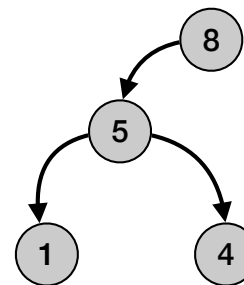
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

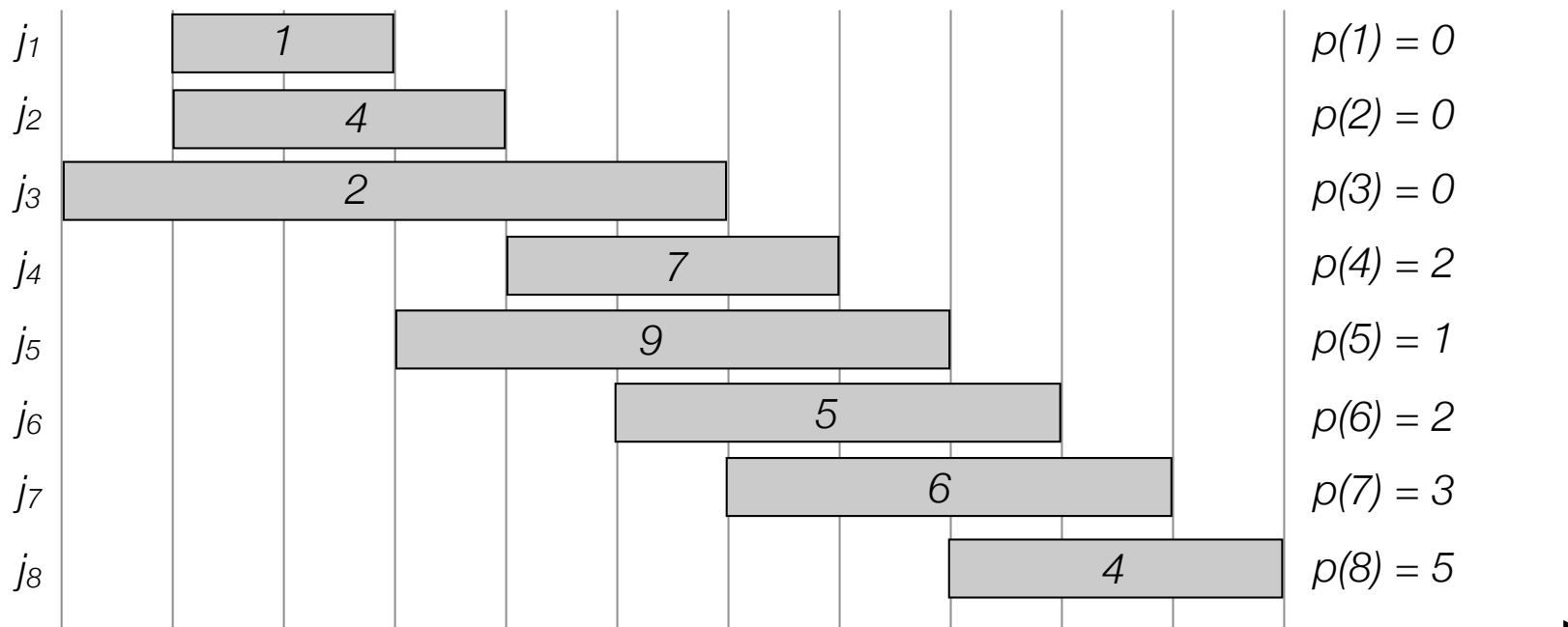
        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



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



# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

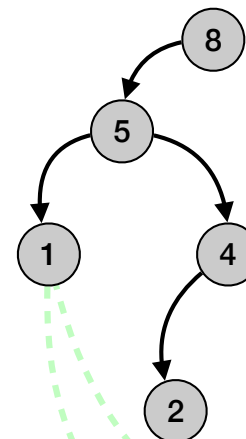
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

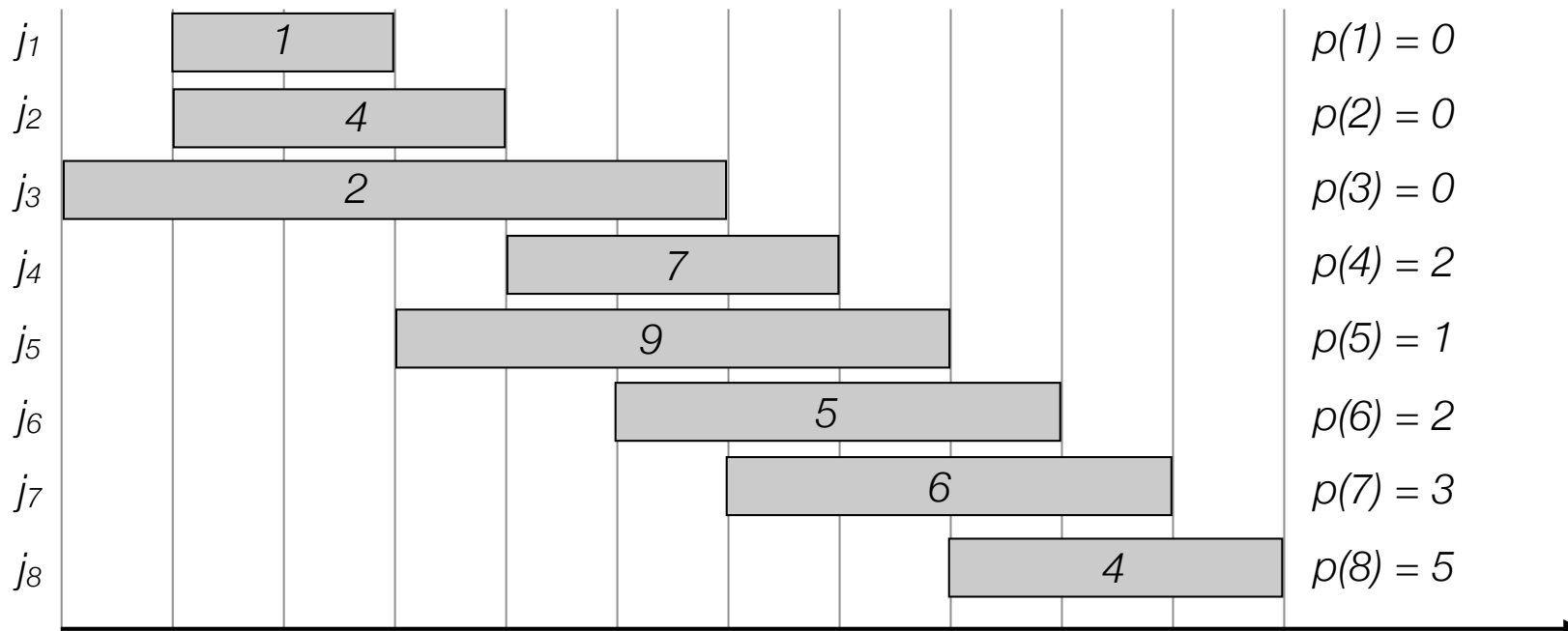
        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



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



# Weighted interval scheduling: memoization

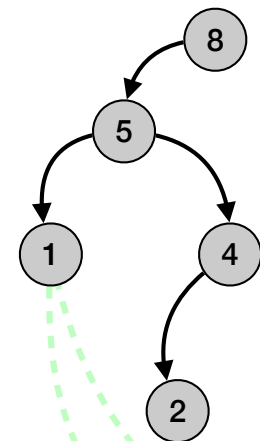
```

Input: n, s[1..n], f[1..n], v[1..n]

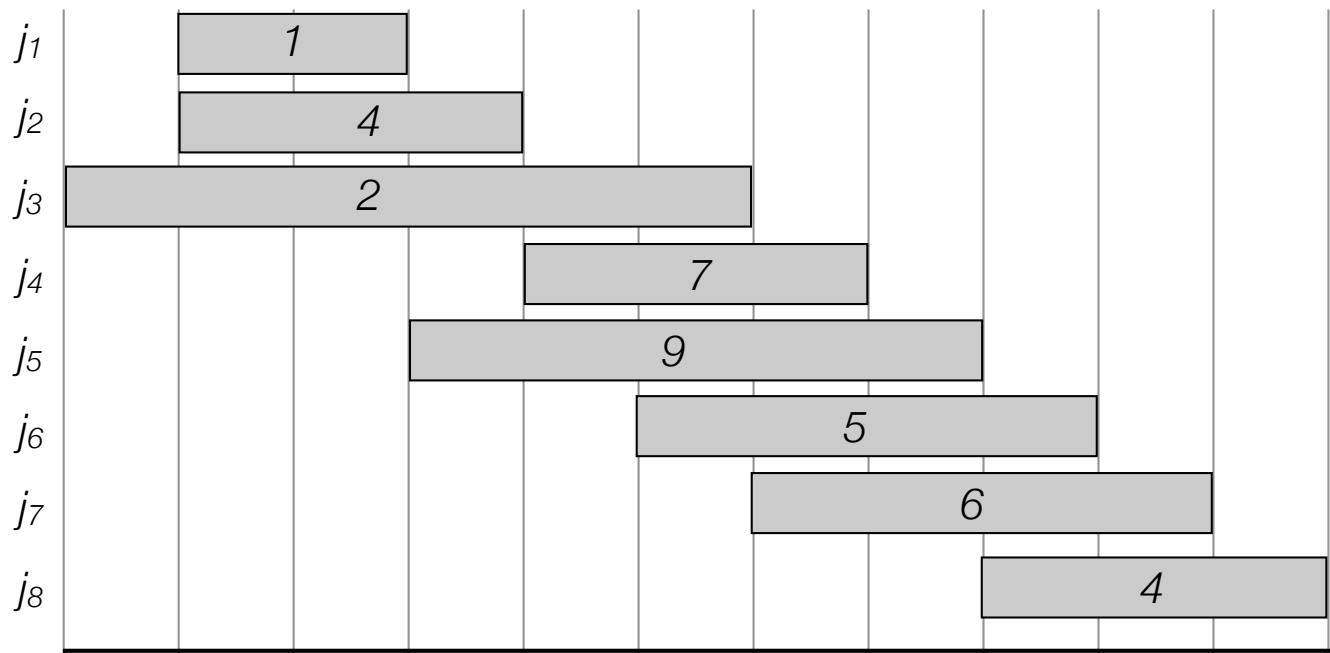
Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]

for j=1 to n
    M[j] = empty
M[0] = 0.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
              Compute-Memoized-Opt(j-1))
return M[j]
    
```



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



- $p(1) = 0$
- $p(2) = 0$
- $p(3) = 0$
- $p(4) = 2$
- $p(5) = 1$
- $p(6) = 2$
- $p(7) = 3$
- $p(8) = 5$

# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

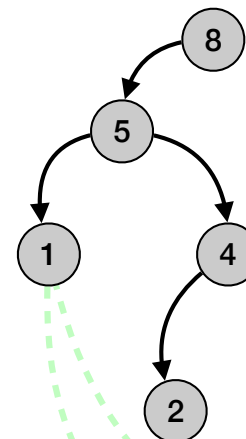
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

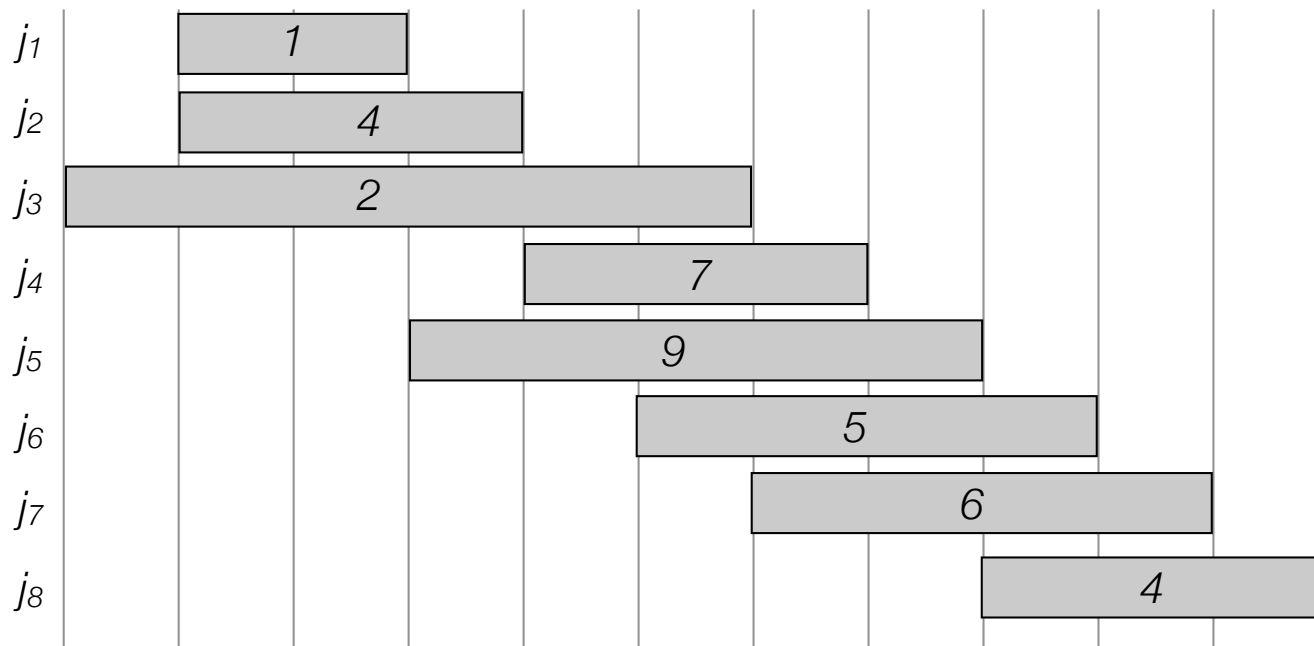
        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



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



$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$

# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

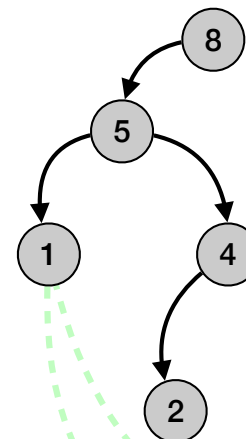
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

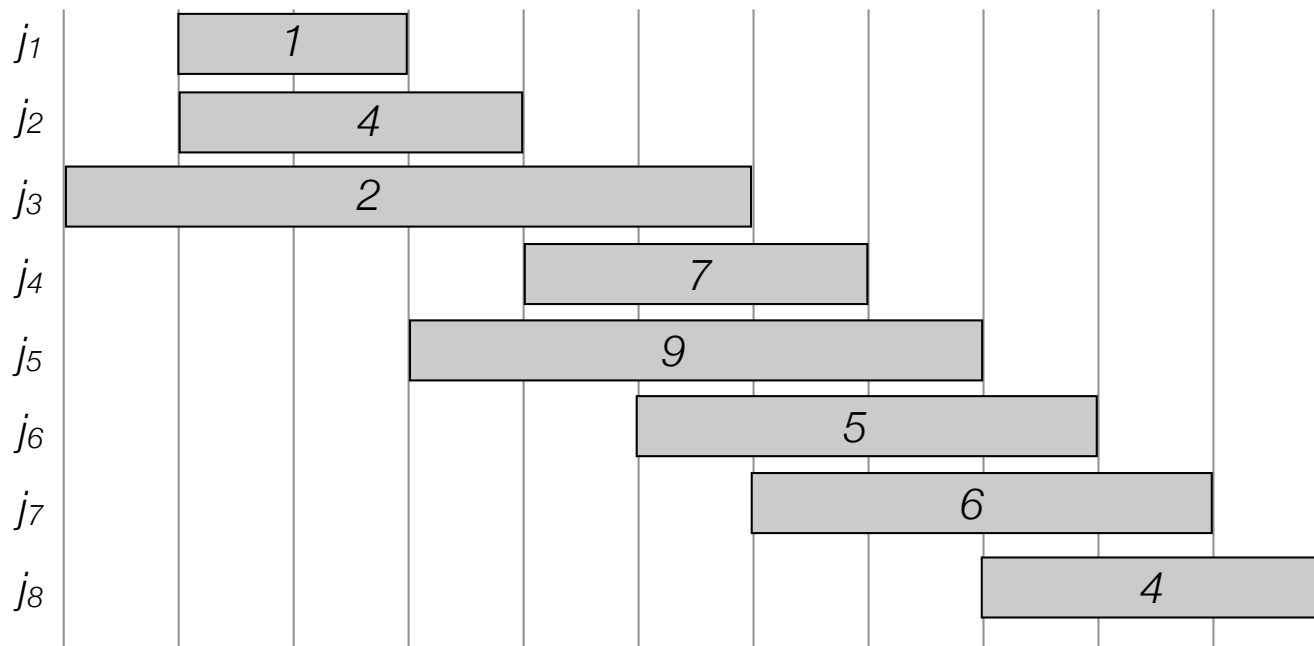
        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



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



$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$

# Weighted interval scheduling: memoization

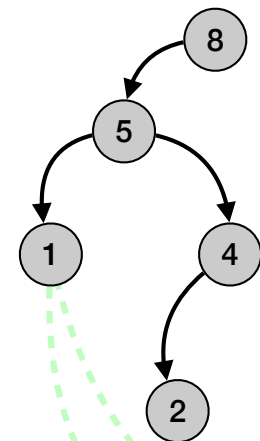
```

Input: n, s[1..n], f[1..n], v[1..n]

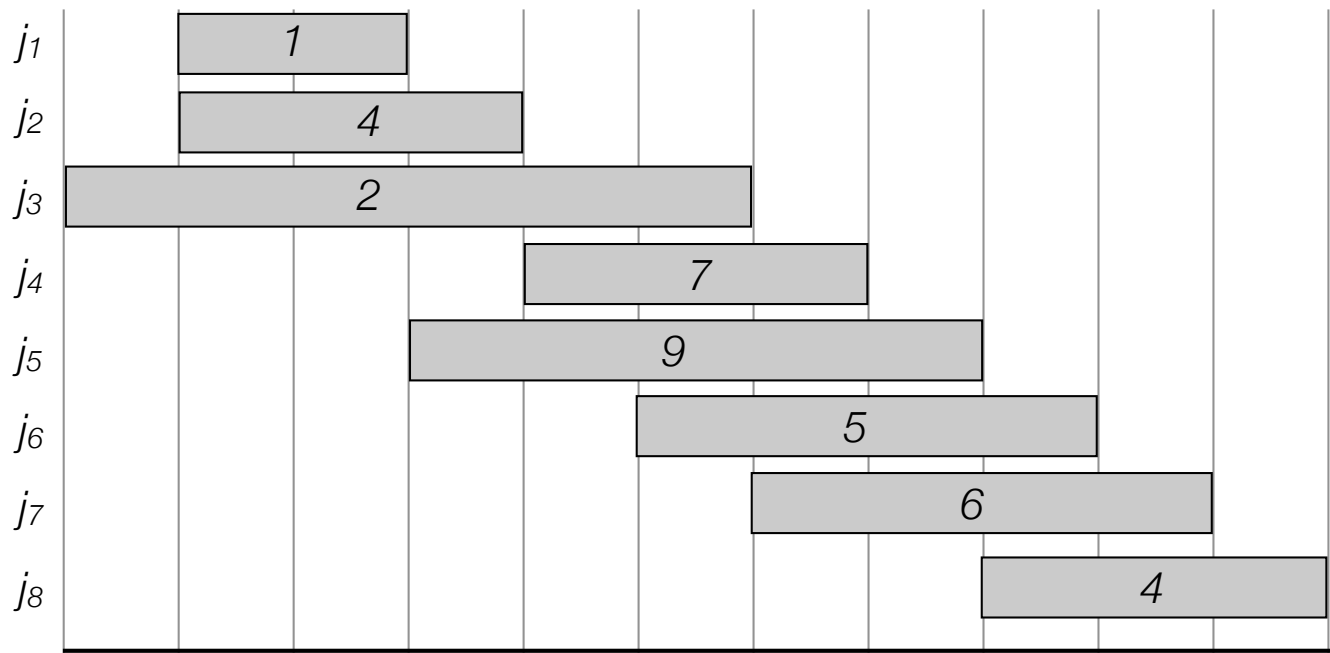
Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]

for j=1 to n
    M[j] = empty
M[0] = 0.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
              Compute-Memoized-Opt(j-1))
return M[j]
    
```



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



- $p(1) = 0$
- $p(2) = 0$
- $p(3) = 0$
- $p(4) = 2$
- $p(5) = 1$
- $p(6) = 2$
- $p(7) = 3$
- $p(8) = 5$

# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

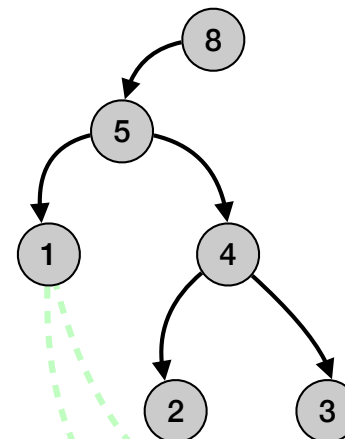
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

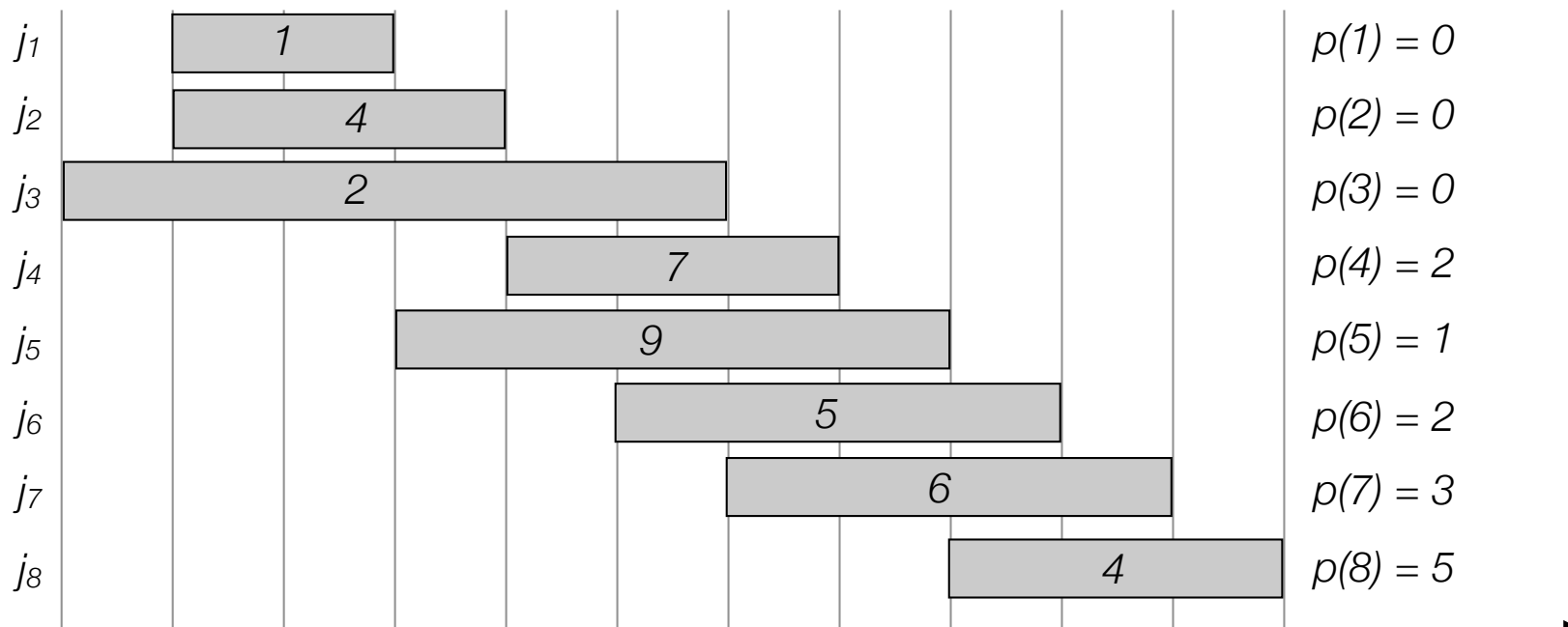
        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



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



# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

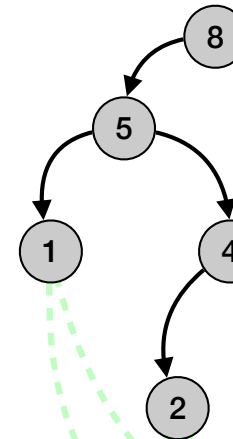
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

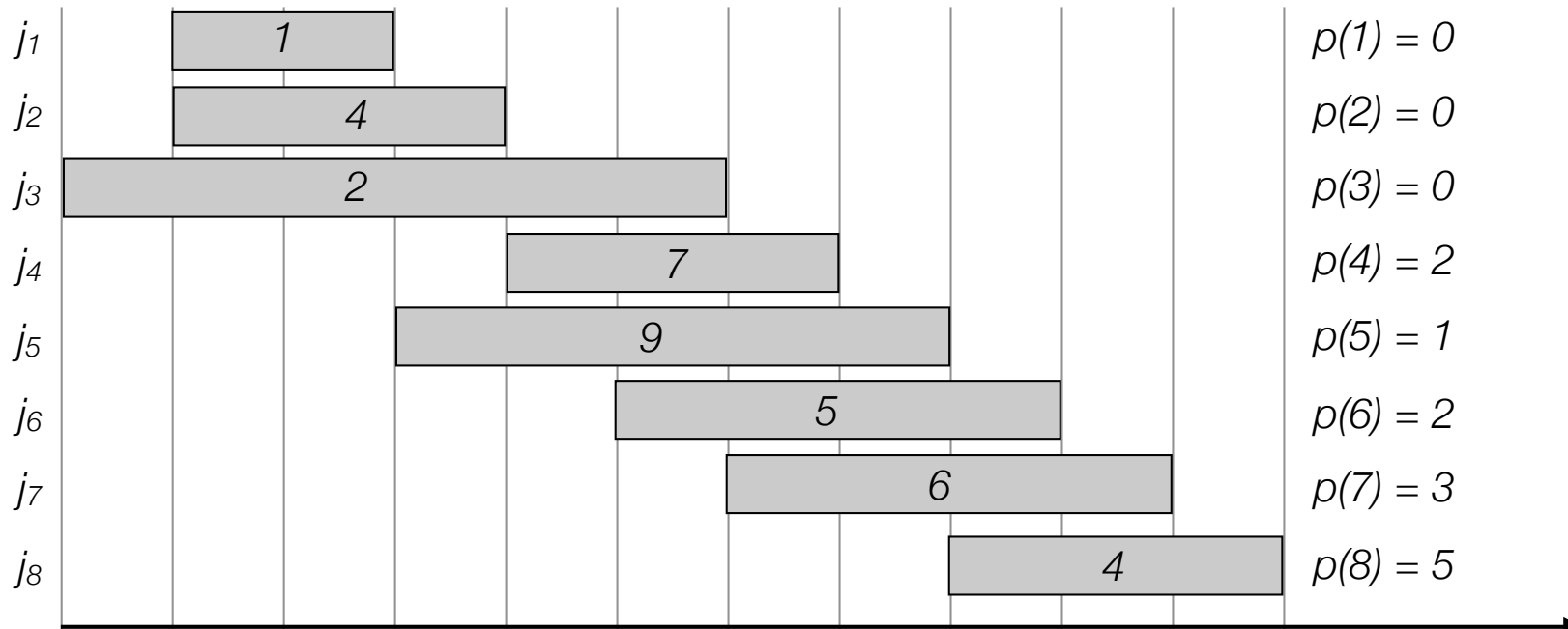
        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



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



$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$



# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

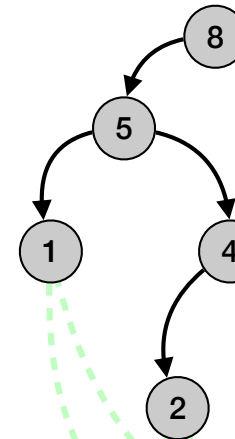
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

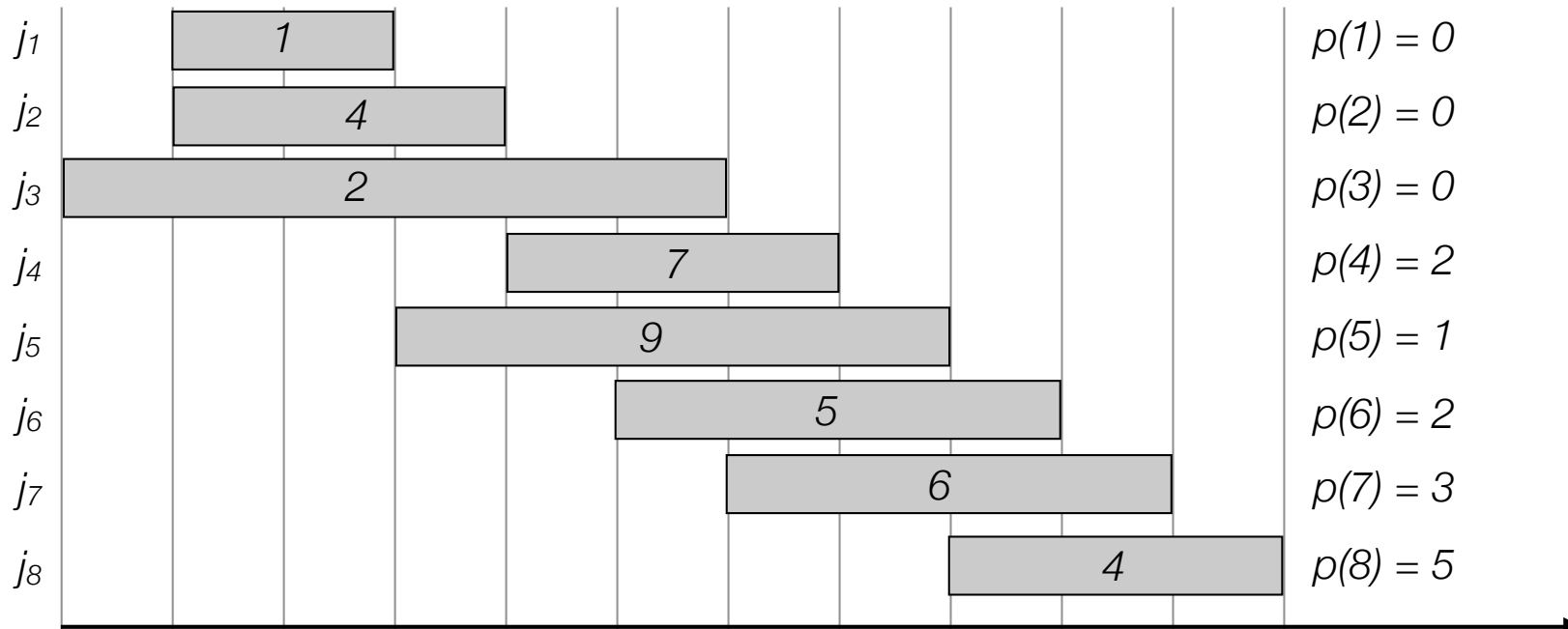
        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



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



$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$

# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

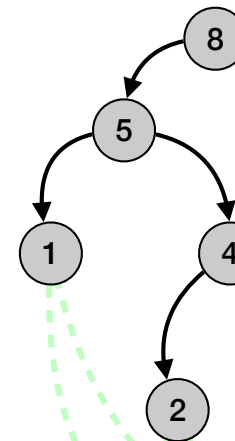
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

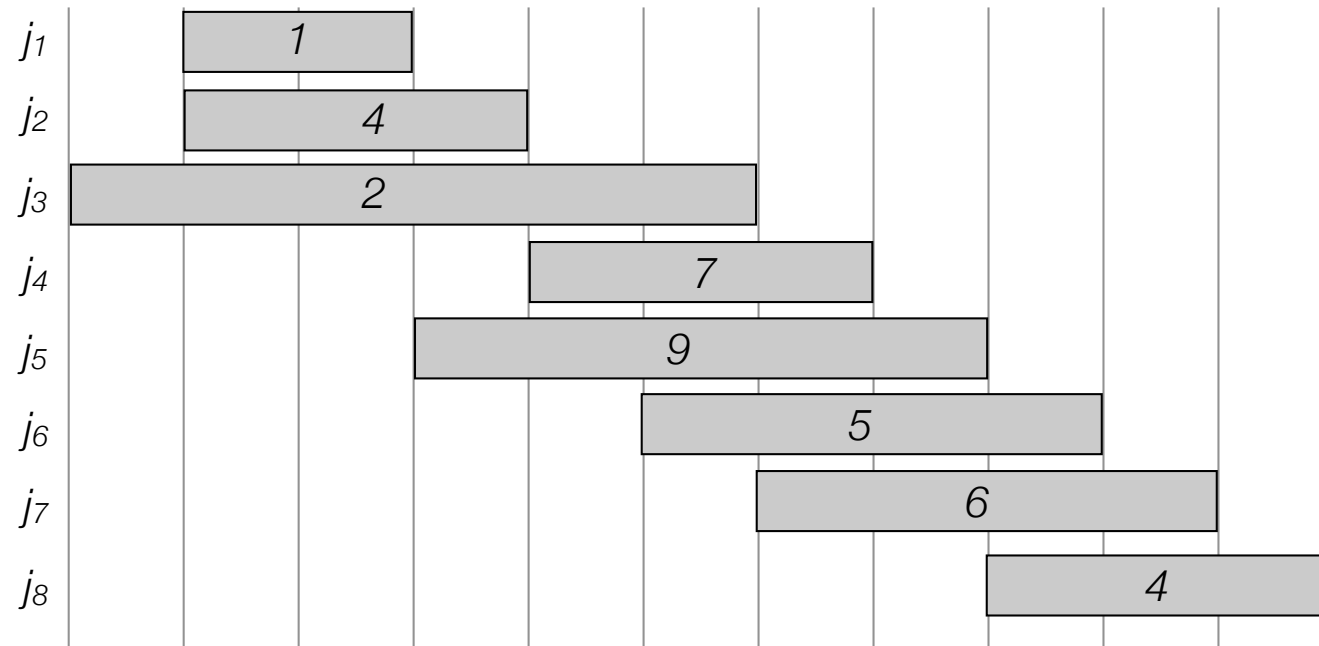
        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



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



$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$

# Weighted interval scheduling: memoization

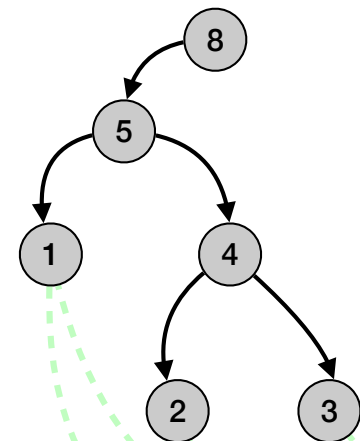
```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]

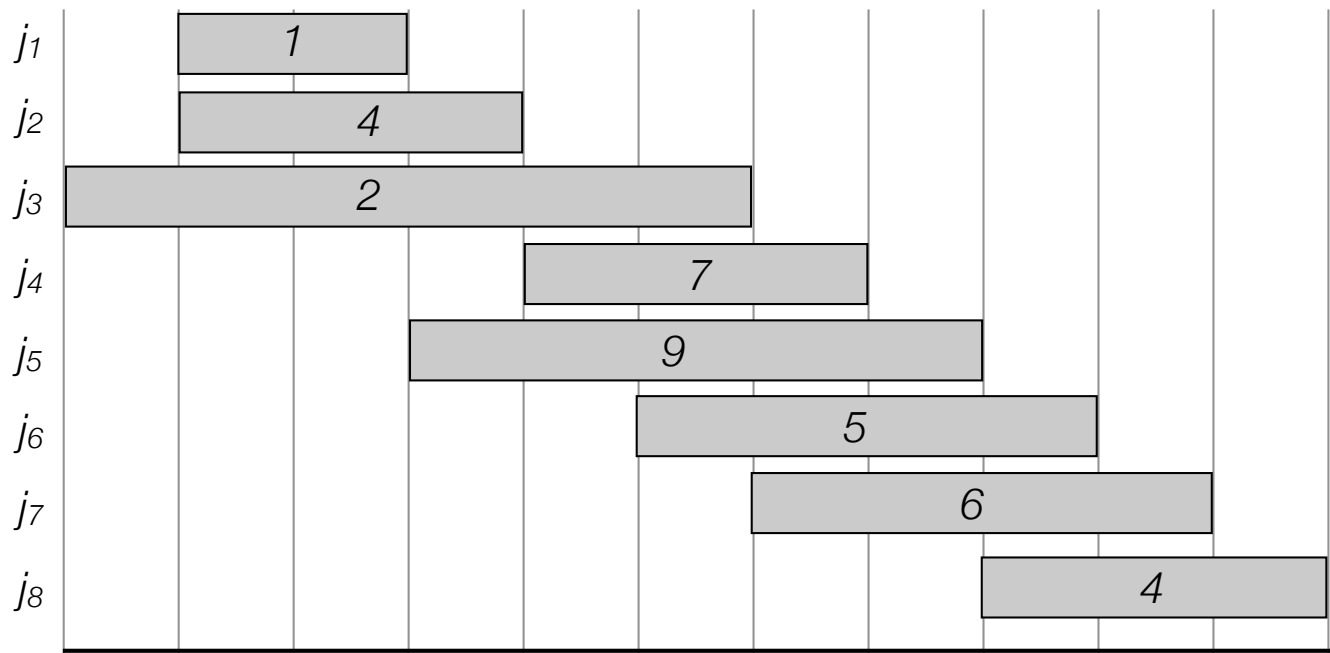
for j=1 to n
    M[j] = empty
M[0] = 0.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
              Compute-Memoized-Opt(j-1))
return M[j]
    
```



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

- $p(1) = 0$
- $p(2) = 0$
- $p(3) = 0$
- $p(4) = 2$
- $p(5) = 1$
- $p(6) = 2$
- $p(7) = 3$
- $p(8) = 5$



# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

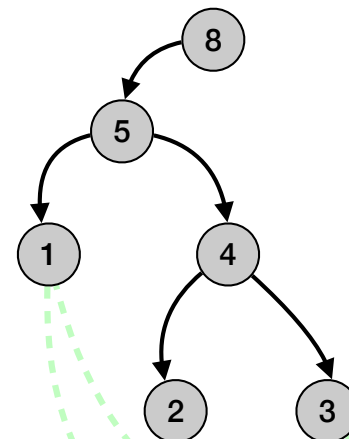
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

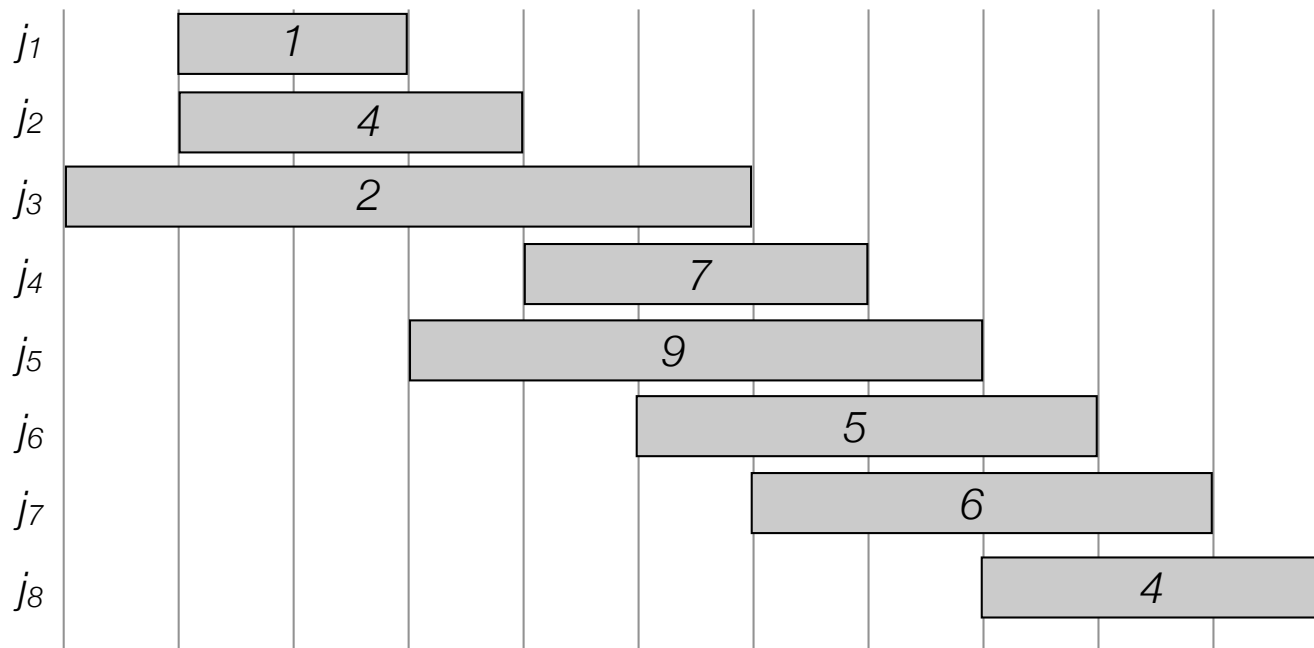
    if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

    return  $M[j]$



$i$	$M[i]$
0	0
1	1
2	4
3	4
4	11
5	
6	
7	
8	



$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$

# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

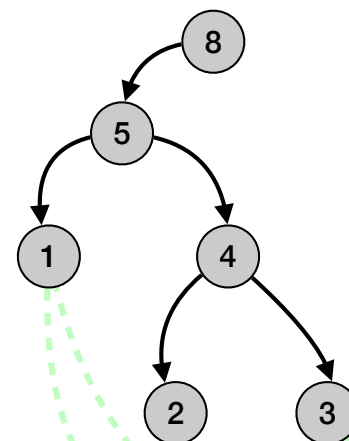
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

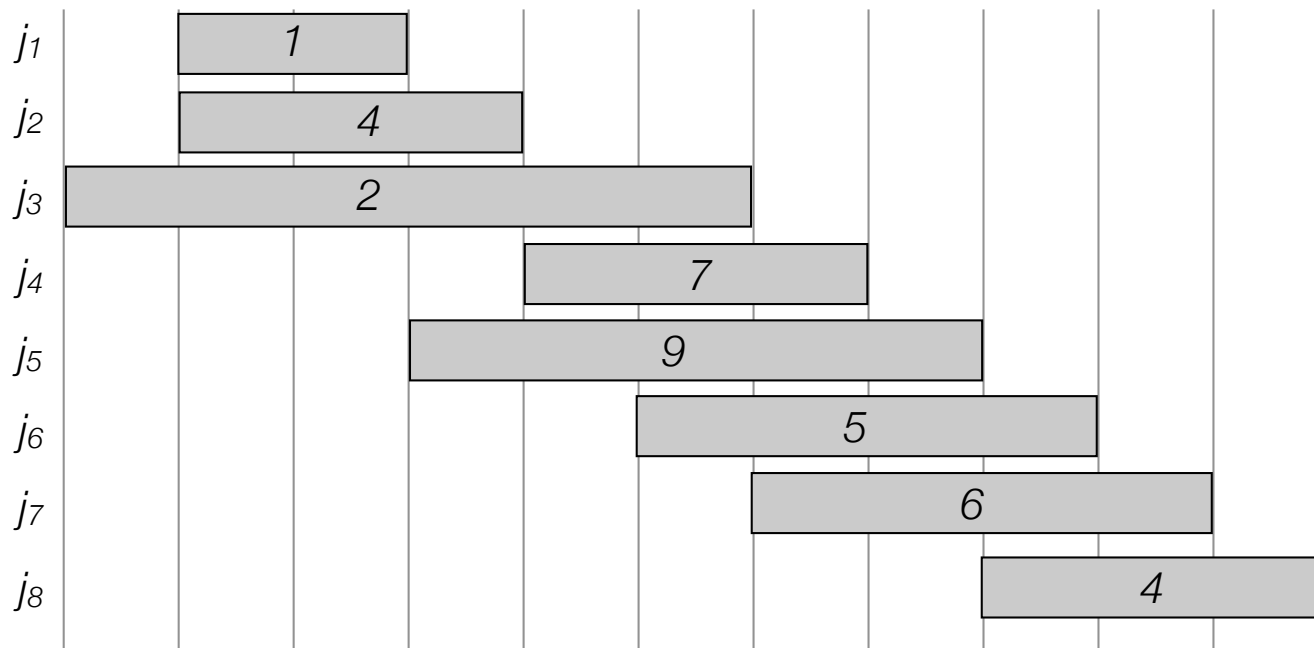
    if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

    return  $M[j]$



$i$	$M[i]$
0	0
1	1
2	4
3	4
4	11
5	11
6	
7	
8	



$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$

# Weighted interval scheduling: memoization

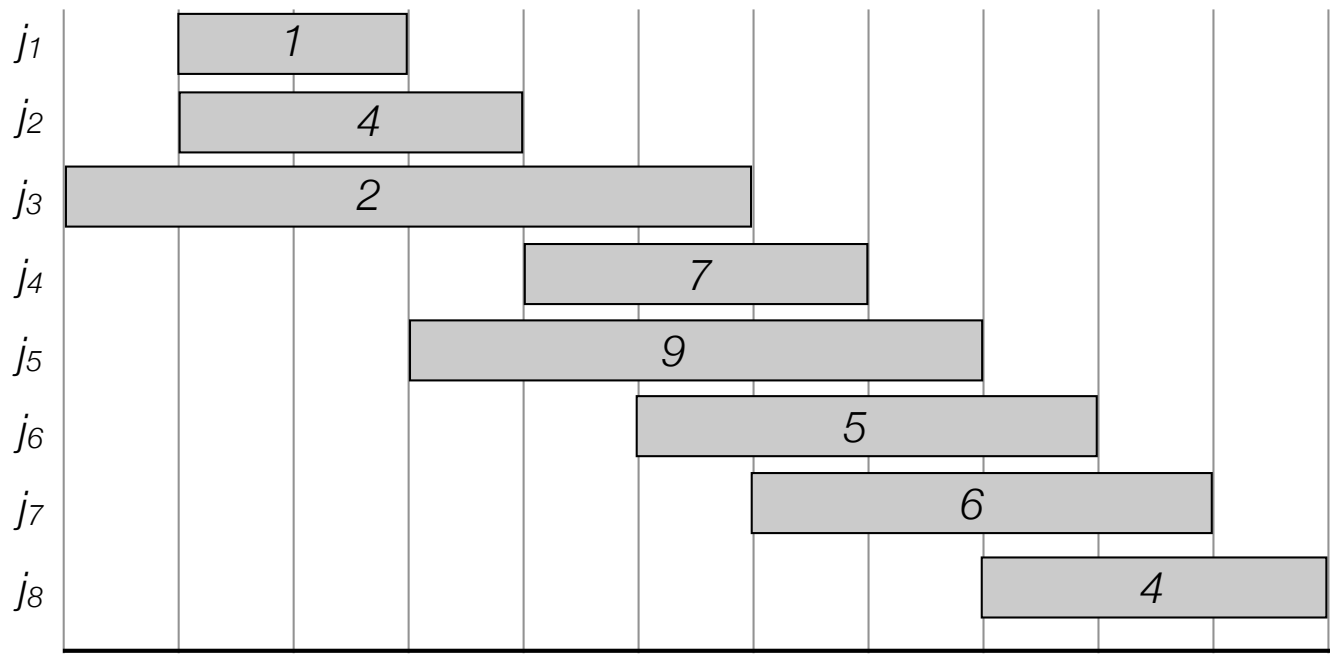
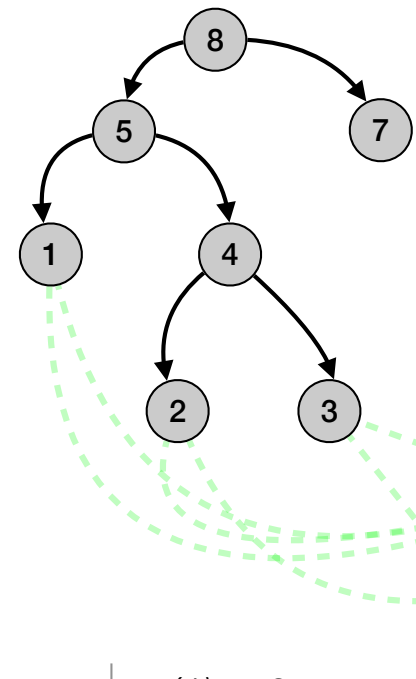
```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time 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.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
                Compute-Memoized-Opt(j-1))
return M[j]
    
```



- $p(1) = 0$
- $p(2) = 0$
- $p(3) = 0$
- $p(4) = 2$
- $p(5) = 1$
- $p(6) = 2$
- $p(7) = 3$
- $p(8) = 5$

i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	
7	
8	

# Weighted interval scheduling: memoization

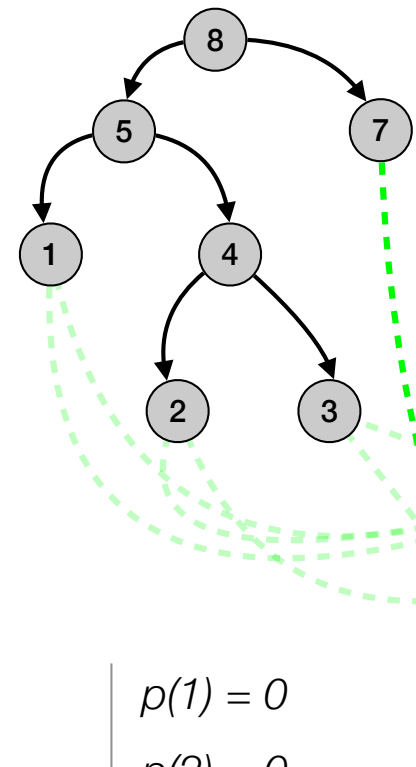
```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]

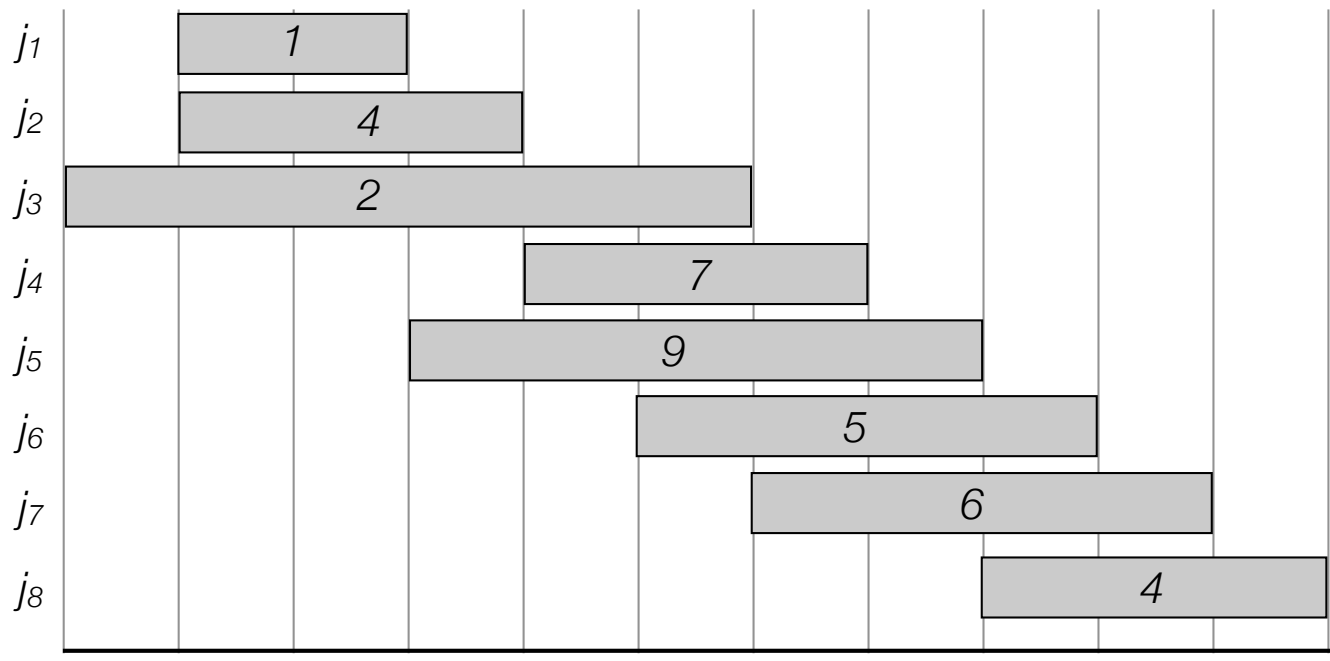
for j=1 to n
    M[j] = empty
M[0] = 0.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
              Compute-Memoized-Opt(j-1))
return M[j]
    
```



i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	
7	
8	

- $p(1) = 0$
- $p(2) = 0$
- $p(3) = 0$
- $p(4) = 2$
- $p(5) = 1$
- $p(6) = 2$
- $p(7) = 3$
- $p(8) = 5$



# Weighted interval scheduling: memoization

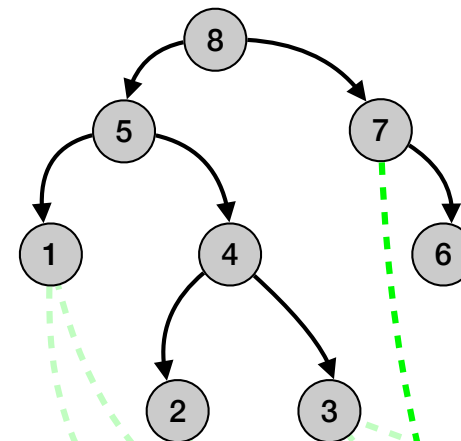
```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]

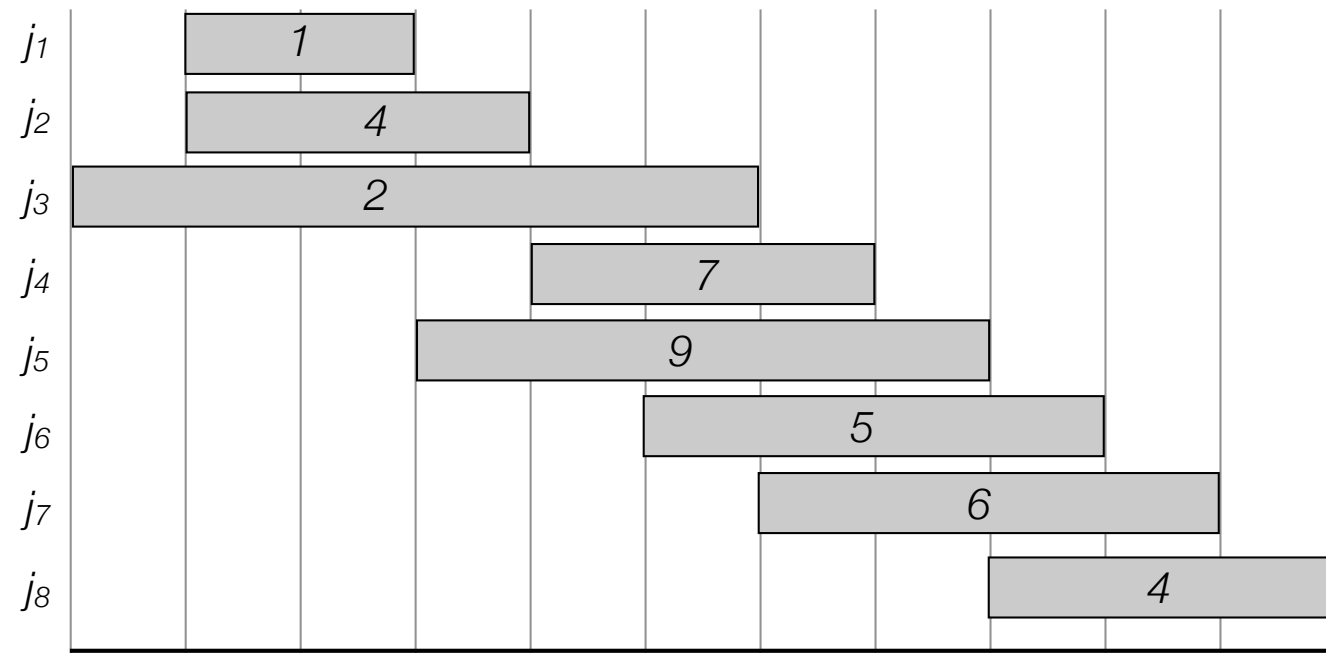
for j=1 to n
    M[j] = empty
M[0] = 0.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
               Compute-Memoized-Opt(j-1))
return M[j]
    
```



i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	
7	
8	

- $p(1) = 0$
- $p(2) = 0$
- $p(3) = 0$
- $p(4) = 2$
- $p(5) = 1$
- $p(6) = 2$
- $p(7) = 3$
- $p(8) = 5$





# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

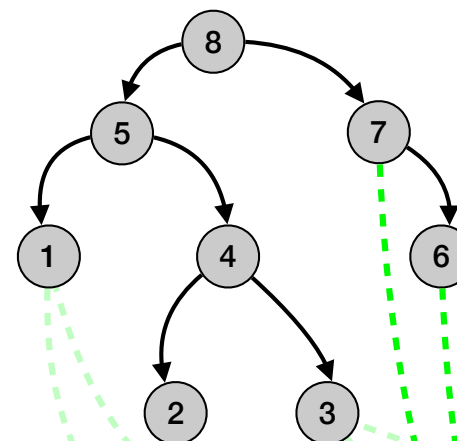
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



$i$	$M[i]$
0	0
1	1
2	4
3	4
4	11
5	11
6	
7	
8	

$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

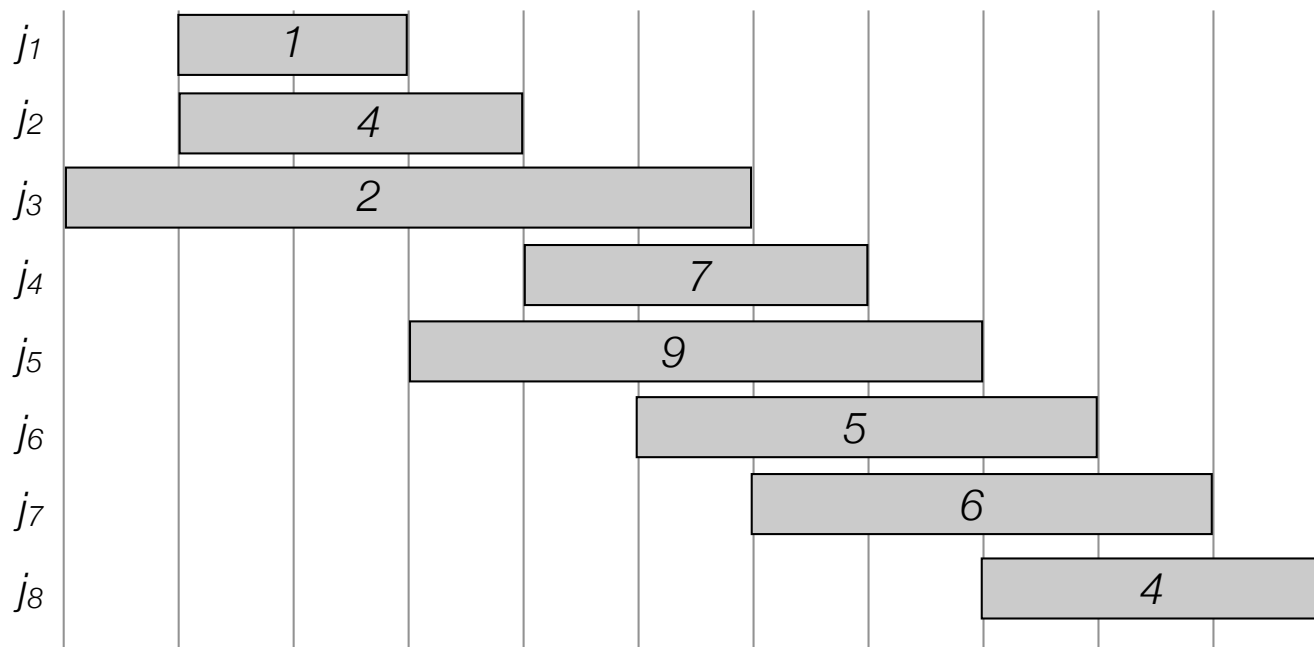
$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$



# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

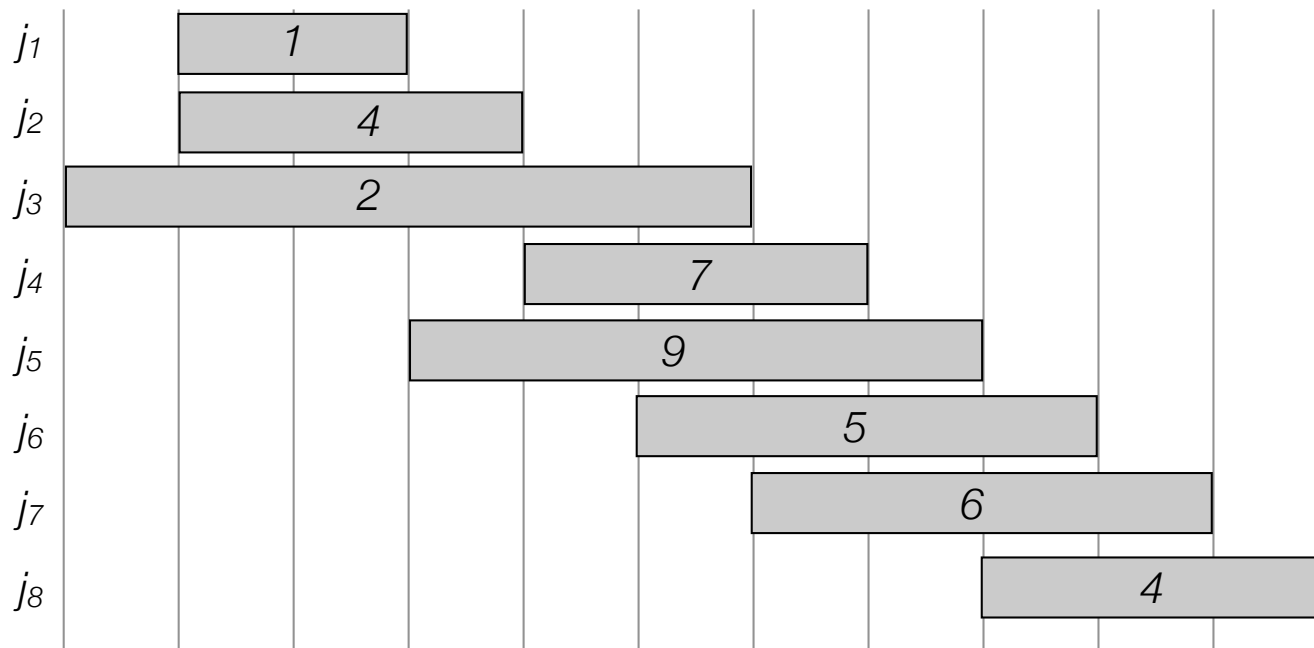
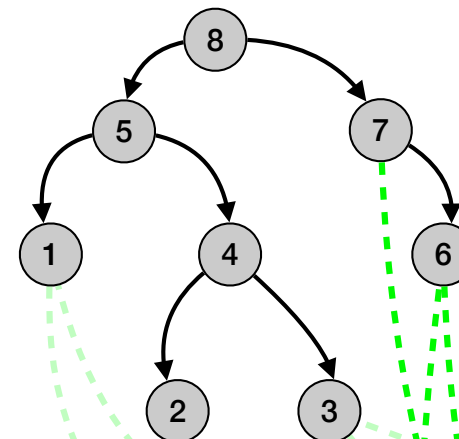
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$

$i$	$M[i]$
0	0
1	1
2	4
3	4
4	11
5	11
6	
7	
8	

# Weighted interval scheduling: memoization

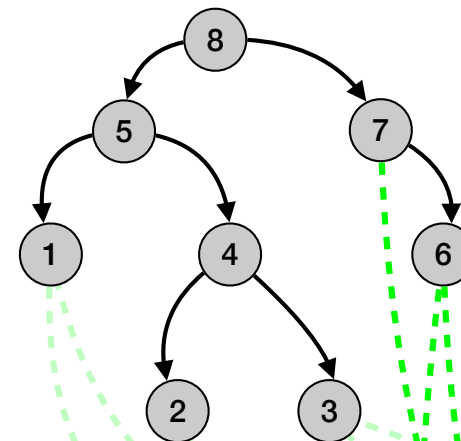
```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n]
Compute p[1], p[2], ..., p[n]

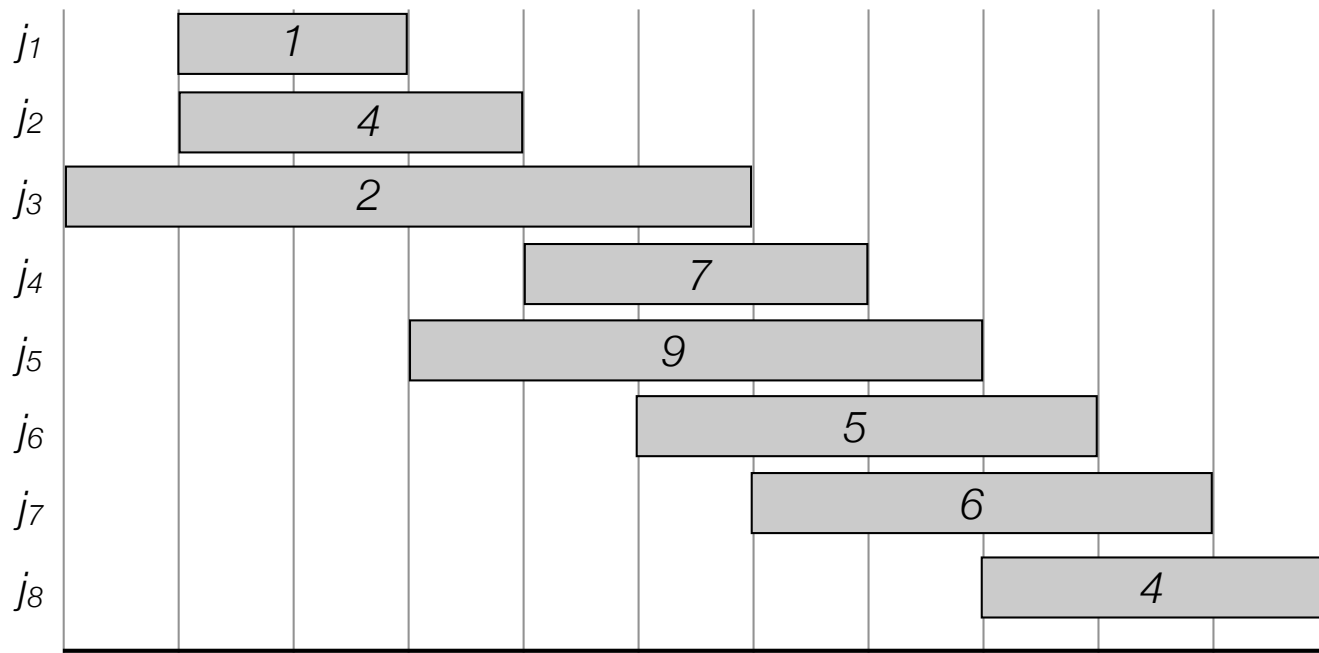
for j=1 to n
    M[j] = empty
M[0] = 0.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
              Compute-Memoized-Opt(j-1))
return M[j]
    
```



i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	
8	

- $p(1) = 0$
- $p(2) = 0$
- $p(3) = 0$
- $p(4) = 2$
- $p(5) = 1$
- $p(6) = 2$
- $p(7) = 3$
- $p(8) = 5$



# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

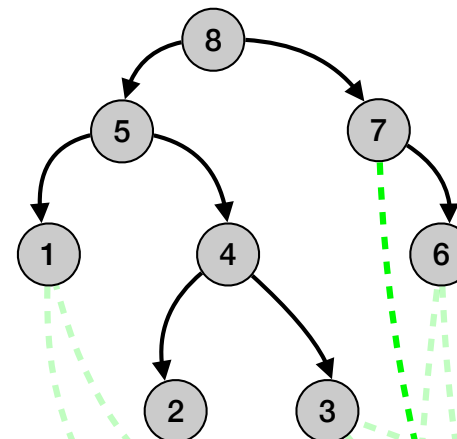
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                      $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



$i$	$M[i]$
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	
8	

$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

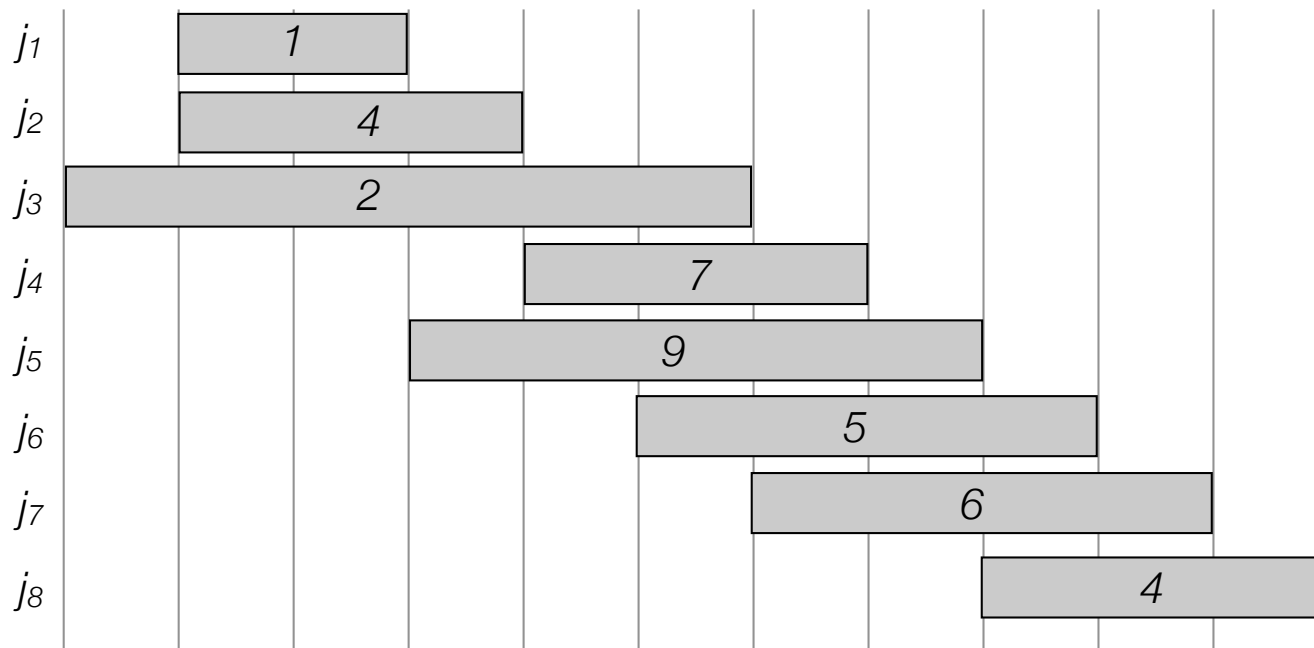
$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$



# Weighted interval scheduling: memoization

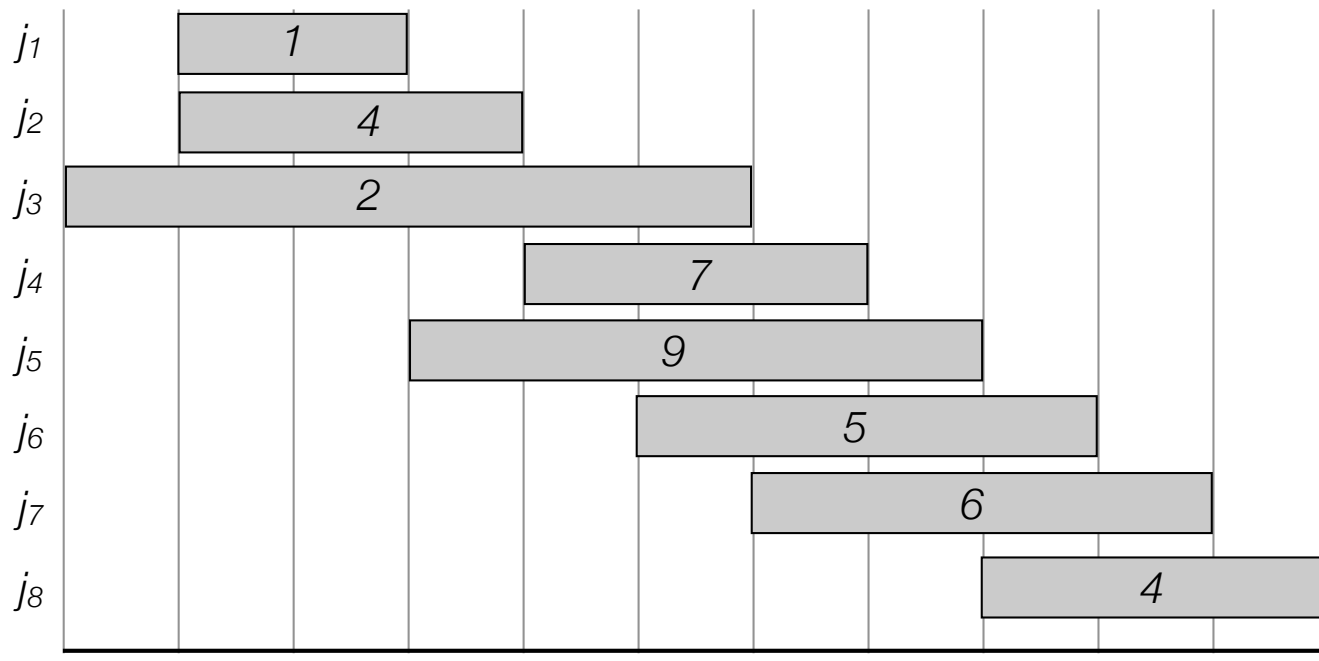
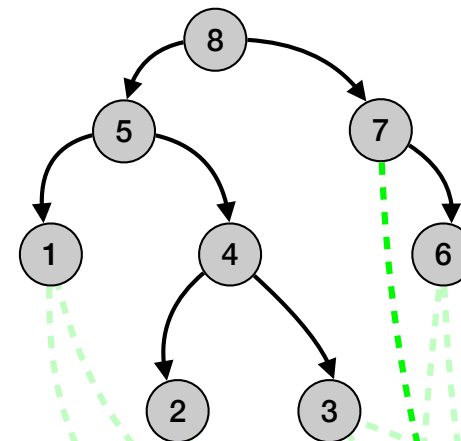
```

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time 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.
Compute-Memoized-Opt(n)

Compute-Memoized-Opt(j)
if M[j] is empty
    M[j] = max(v[j] + Compute-Memoized-Opt(p[j]),
               Compute-Memoized-Opt(j-1))
return M[j]
    
```



- $p(1) = 0$
- $p(2) = 0$
- $p(3) = 0$
- $p(4) = 2$
- $p(5) = 1$
- $p(6) = 2$
- $p(7) = 3$
- $p(8) = 5$

i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	

# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

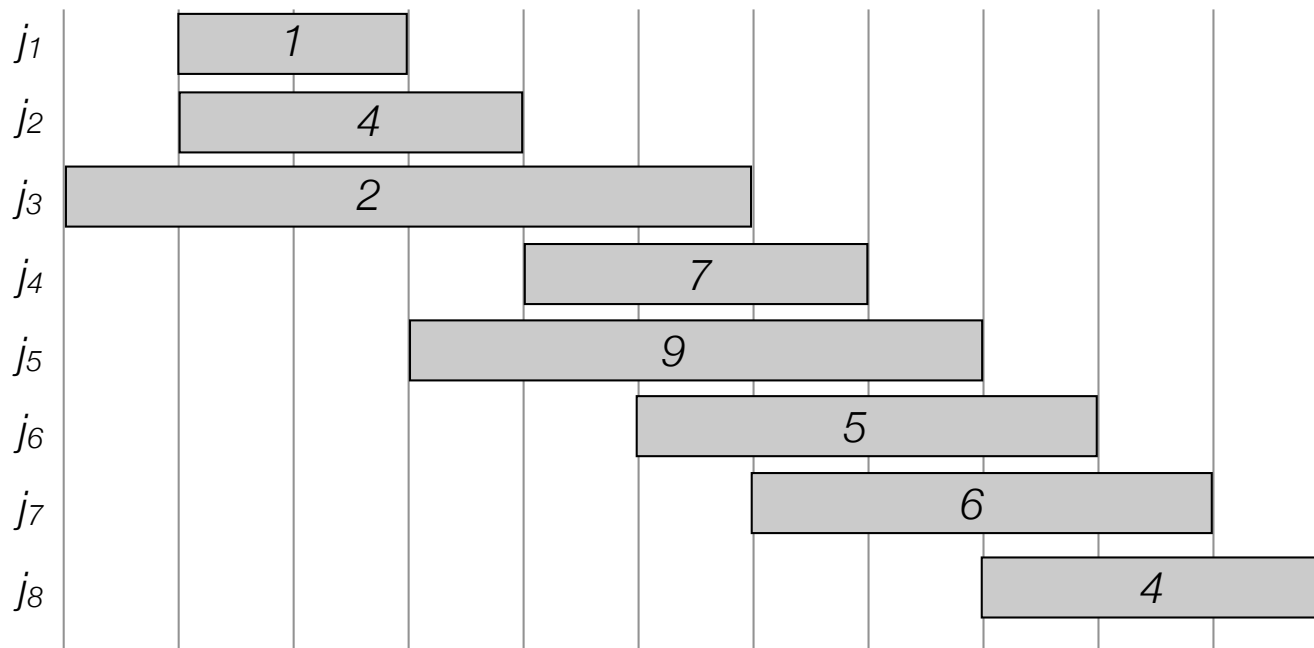
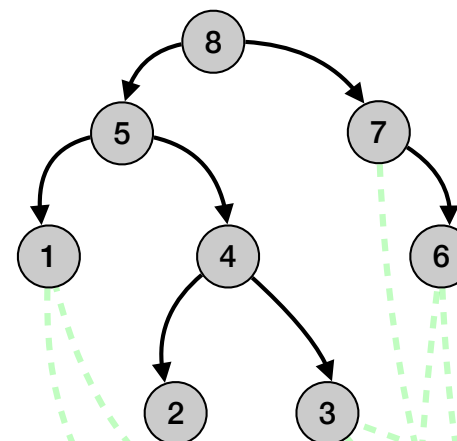
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                              $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$

$i$	$M[i]$
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	

# Weighted interval scheduling: memoization

**Input:**  $n, s[1..n], f[1..n], v[1..n]$

Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$

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

for  $j=1$  to  $n$

$M[j] = \text{empty}$

$M[0] = 0.$

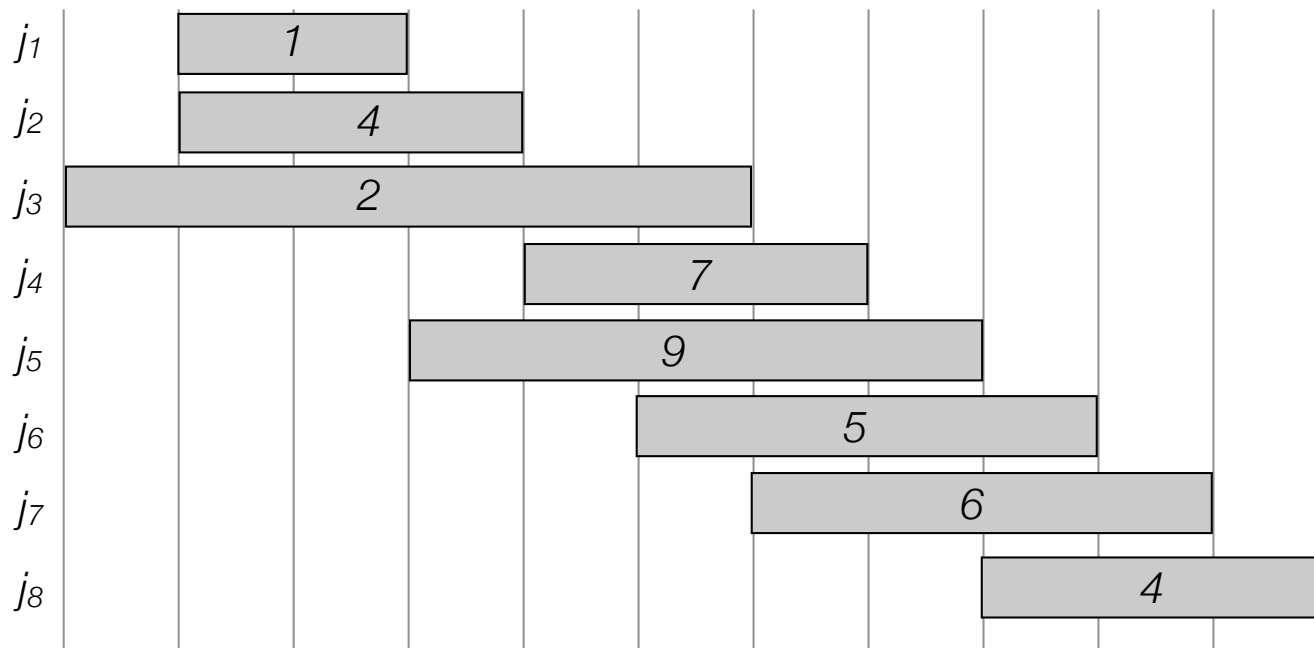
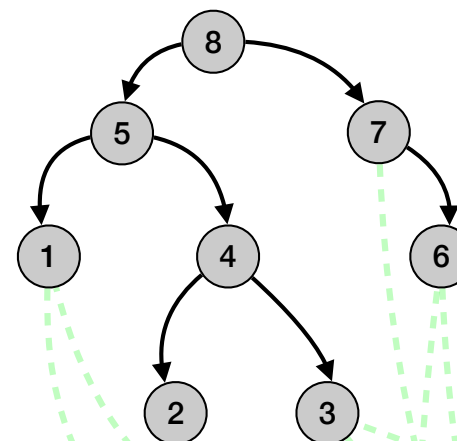
    Compute-Memoized-Opt( $n$ )

    Compute-Memoized-Opt( $j$ )

        if  $M[j]$  is empty

$M[j] = \max(v[j] + \text{Compute-Memoized-Opt}(p[j]),$   
                              $\text{Compute-Memoized-Opt}(j-1))$

        return  $M[j]$



$$p(1) = 0$$

$$p(2) = 0$$

$$p(3) = 0$$

$$p(4) = 2$$

$$p(5) = 1$$

$$p(6) = 2$$

$$p(7) = 3$$

$$p(8) = 5$$

i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: bottom-up

---

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

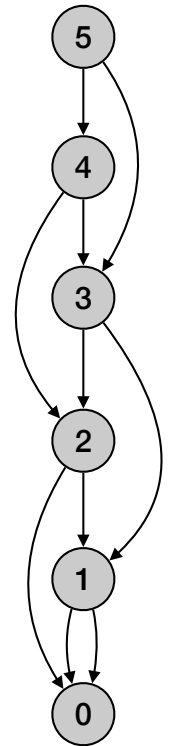
```
Compute  $p[1], p[2], \dots, p[n]$ 
```

```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```





# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

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

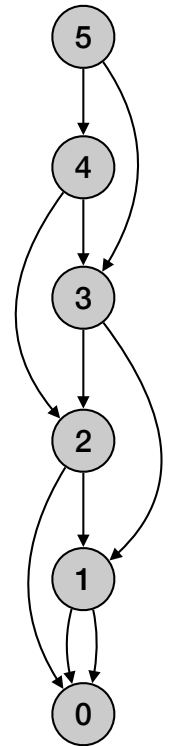
```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```

- Running time  $O(n \log n)$ :



# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

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

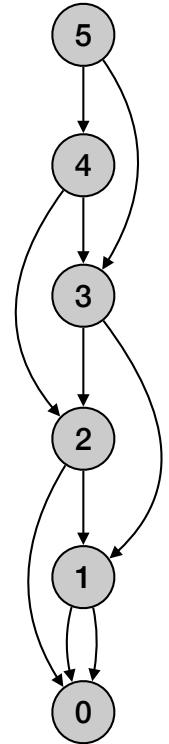
```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```

- Running time  $O(n \log n)$ :
  - Sorting takes  $O(n \log n)$  time.



# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

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

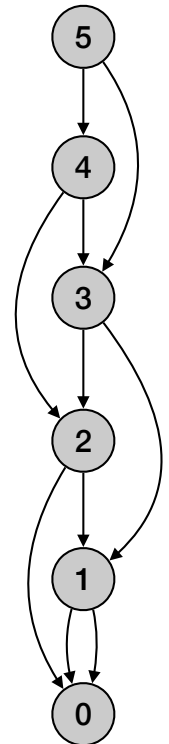
```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```

- Running time  $O(n \log n)$ :
  - Sorting takes  $O(n \log n)$  time.
  - Computing  $p(n)$ :  $O(n \log n)$



# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

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

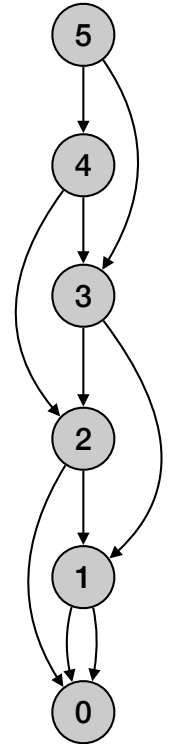
```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```

- Running time  $O(n \log n)$ :
  - Sorting takes  $O(n \log n)$  time.
  - Computing  $p(n)$ :  $O(n \log n)$
  - For loop:  $O(n)$  time



# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

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

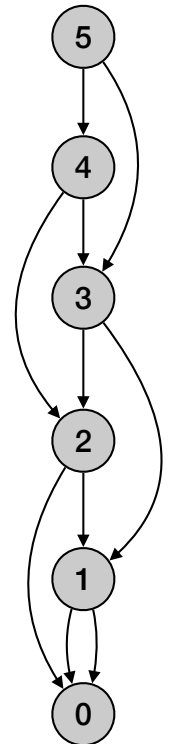
```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```

- Running time  $O(n \log n)$ :
  - Sorting takes  $O(n \log n)$  time.
  - Computing  $p(n)$ :  $O(n \log n)$
  - For loop:  $O(n)$  time
    - Each iteration takes constant time.



# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

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

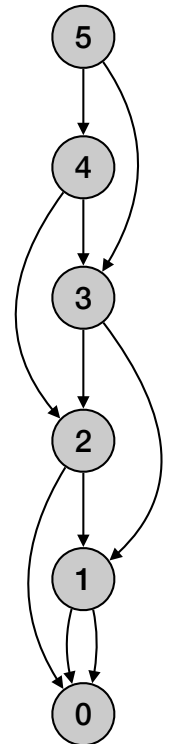
```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```

- Running time  $O(n \log n)$ :
  - Sorting takes  $O(n \log n)$  time.
  - Computing  $p(n)$ :  $O(n \log n)$
  - For loop:  $O(n)$  time
    - Each iteration takes constant time.
- Space  $O(n)$



# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

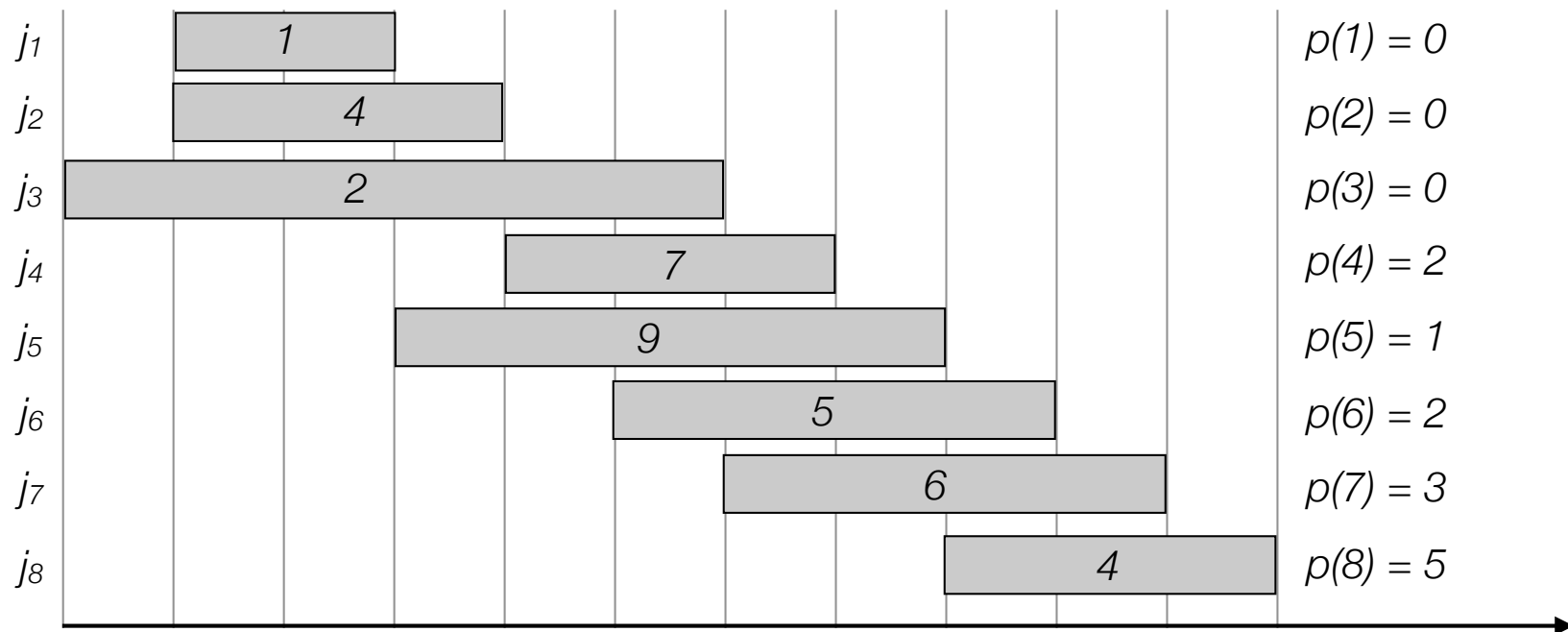
```
Compute  $p[1], p[2], \dots, p[n]$ 
```

```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```



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

# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

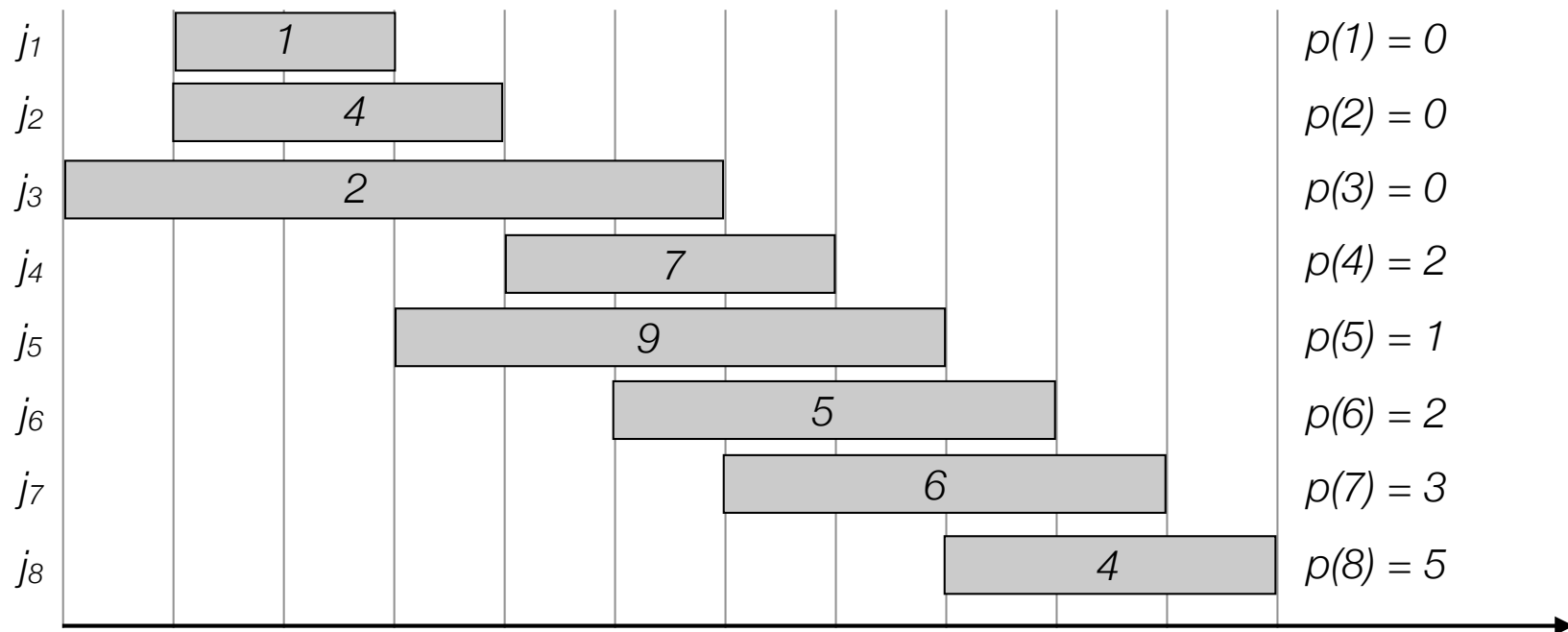
```
Compute  $p[1], p[2], \dots, p[n]$ 
```

```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```



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



# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

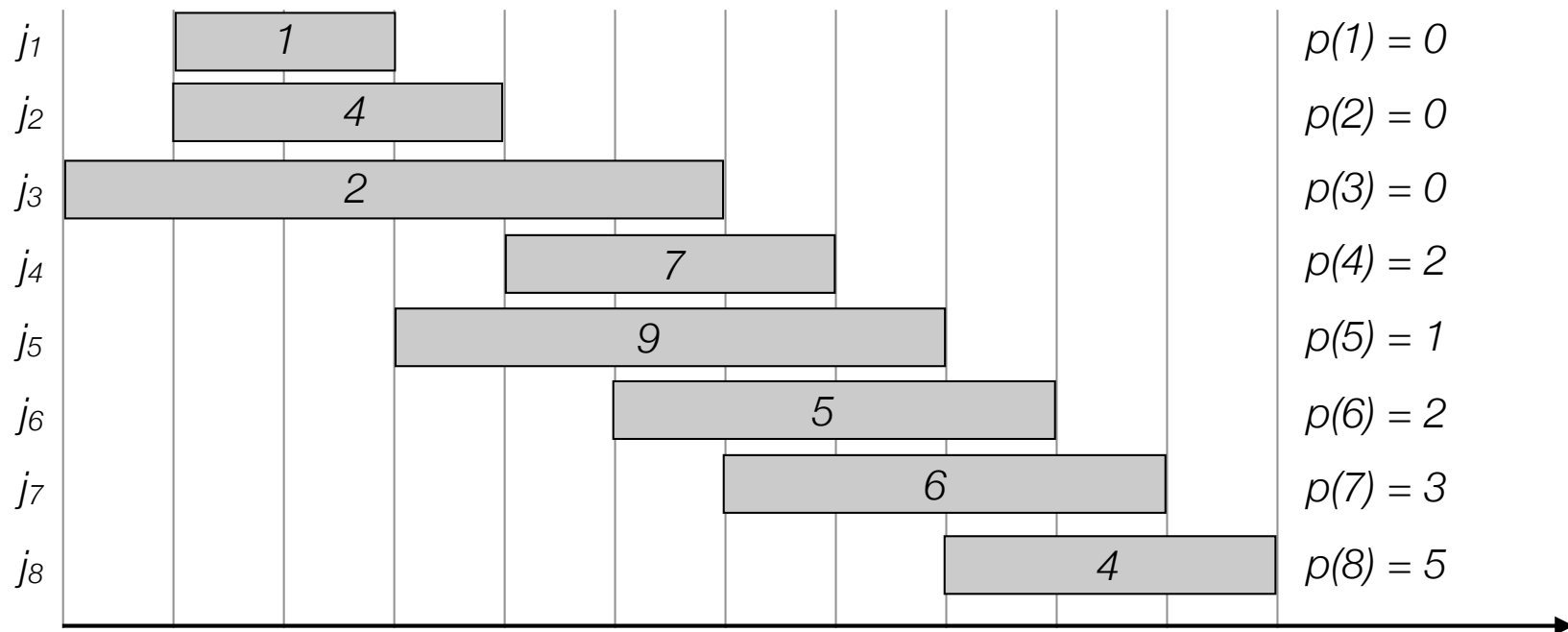
```
Compute  $p[1], p[2], \dots, p[n]$ 
```

```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```



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

# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

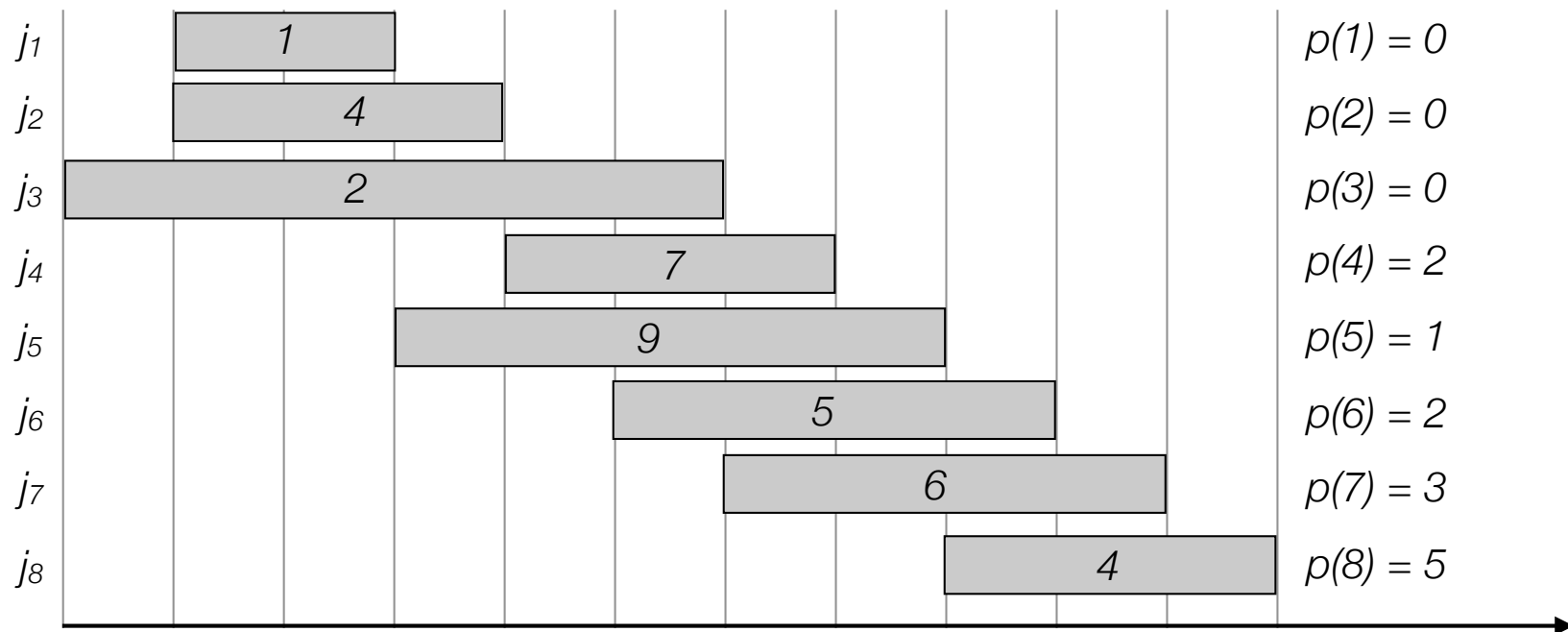
```
Compute  $p[1], p[2], \dots, p[n]$ 
```

```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```



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

# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

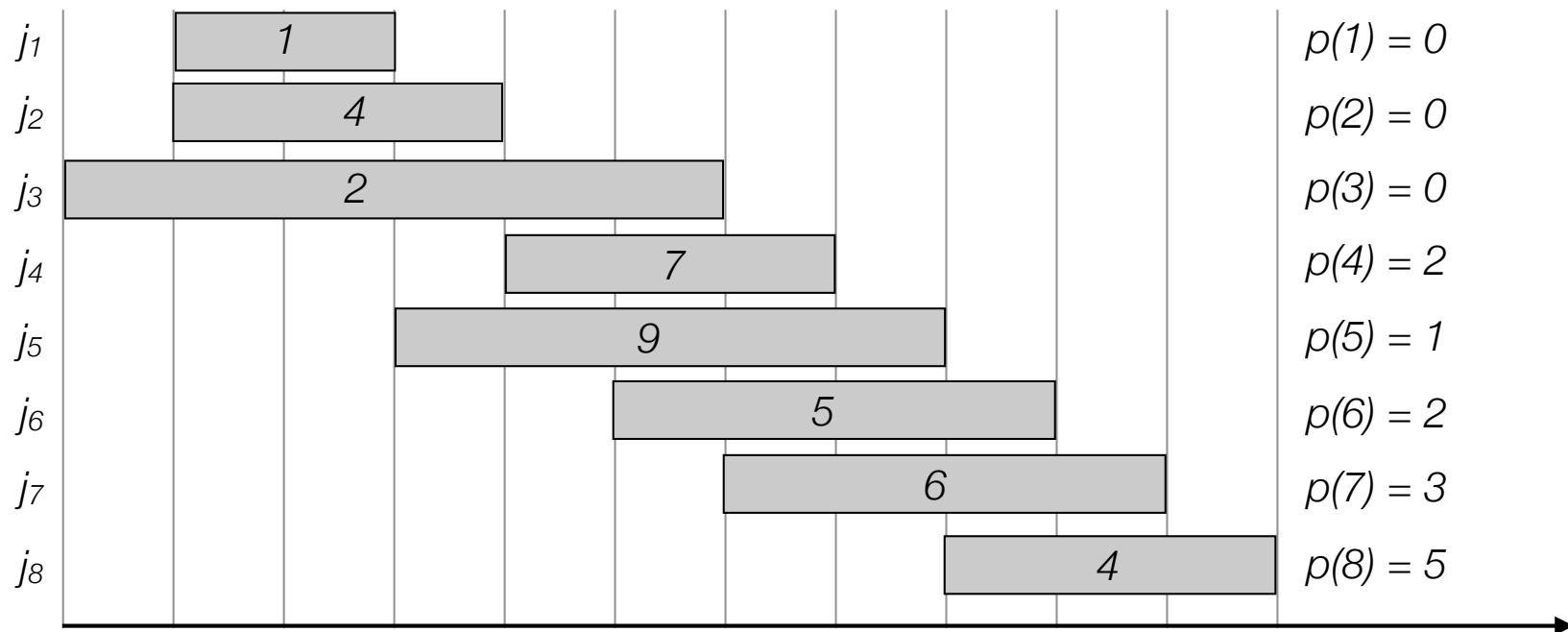
```
Compute  $p[1], p[2], \dots, p[n]$ 
```

```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```



$i$	$M[i]$
0	0
1	1
2	4
3	4
4	11
5	
6	
7	
8	

# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

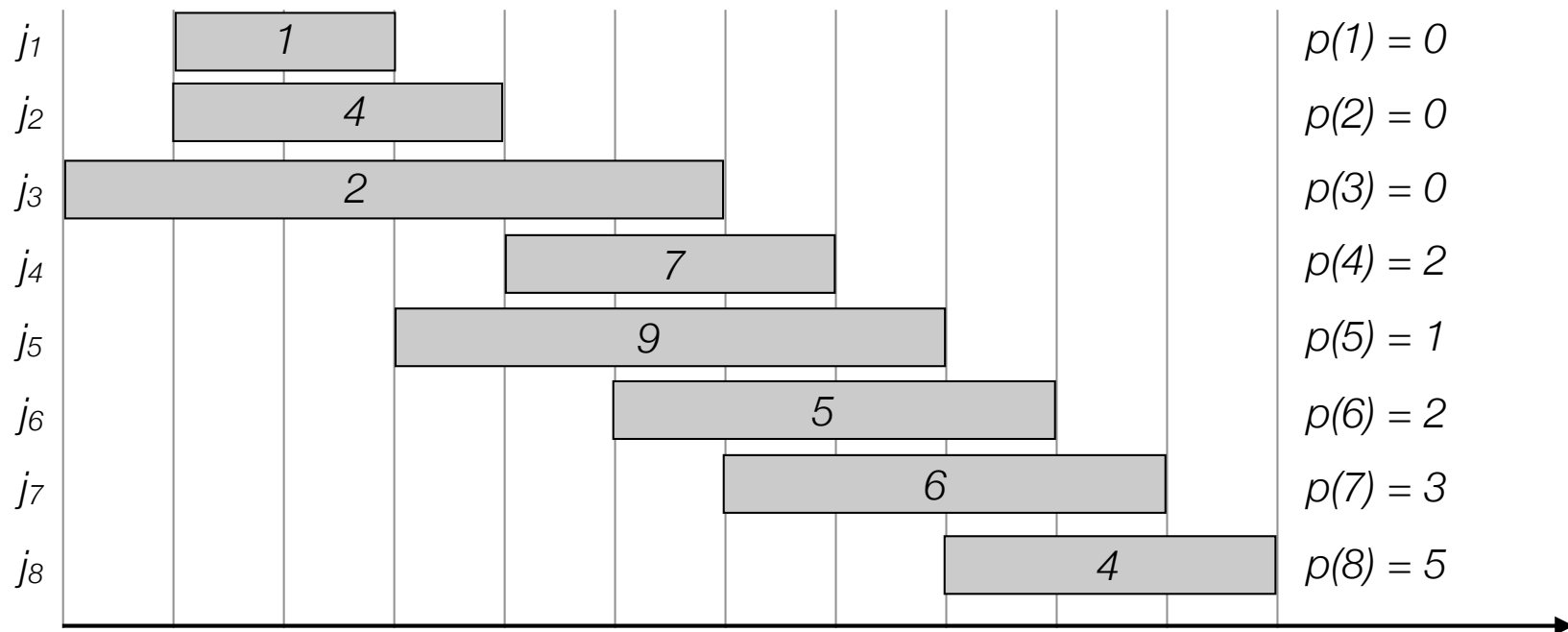
```
Compute  $p[1], p[2], \dots, p[n]$ 
```

```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```



$i$	$M[i]$
0	0
1	1
2	4
3	4
4	11
5	11
6	
7	
8	

# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

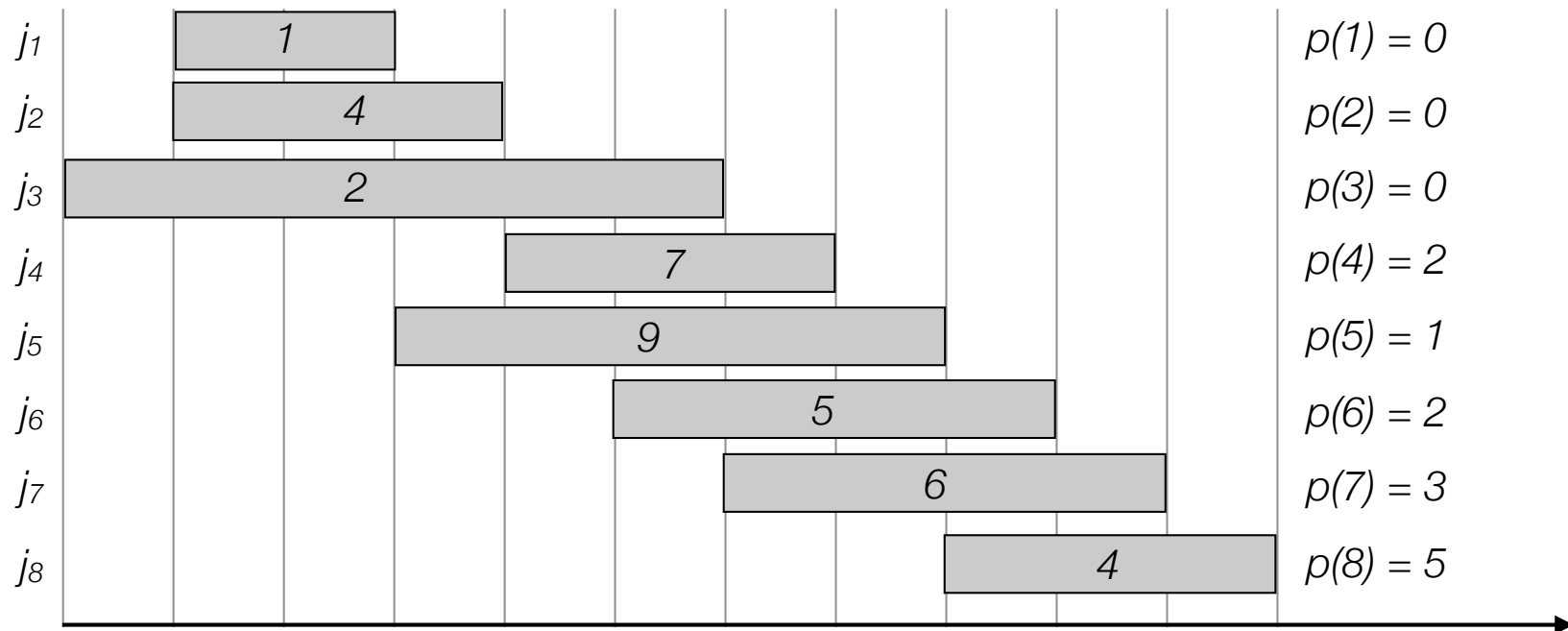
```
Compute  $p[1], p[2], \dots, p[n]$ 
```

```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```



$i$	$M[i]$
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	
8	

# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

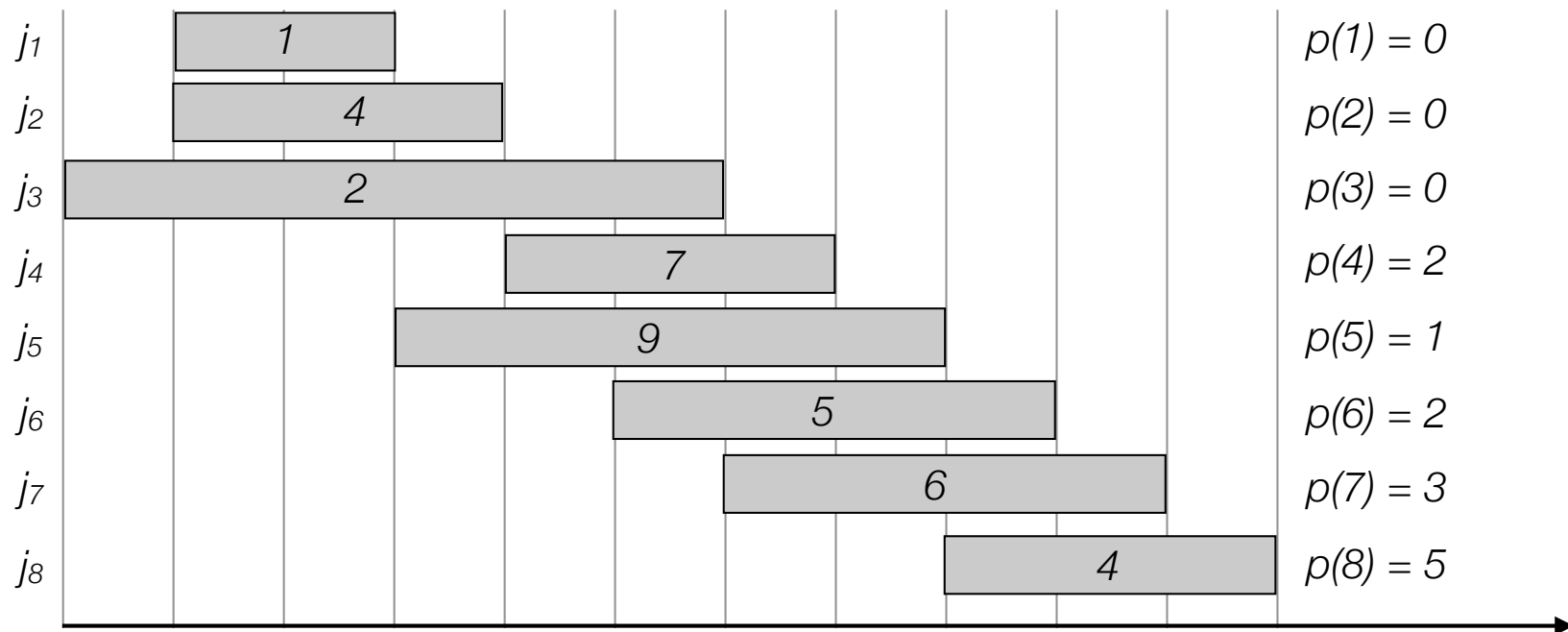
```
Compute  $p[1], p[2], \dots, p[n]$ 
```

```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```



i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	

# Weighted interval scheduling: bottom-up

```
Compute-Bottom-Up-Opt( $n, s[1..n], f[1..n], v[1..n]$ )
```

```
Sort jobs by finish time so that  $f[1] \leq f[2] \leq \dots \leq f[n]$ 
```

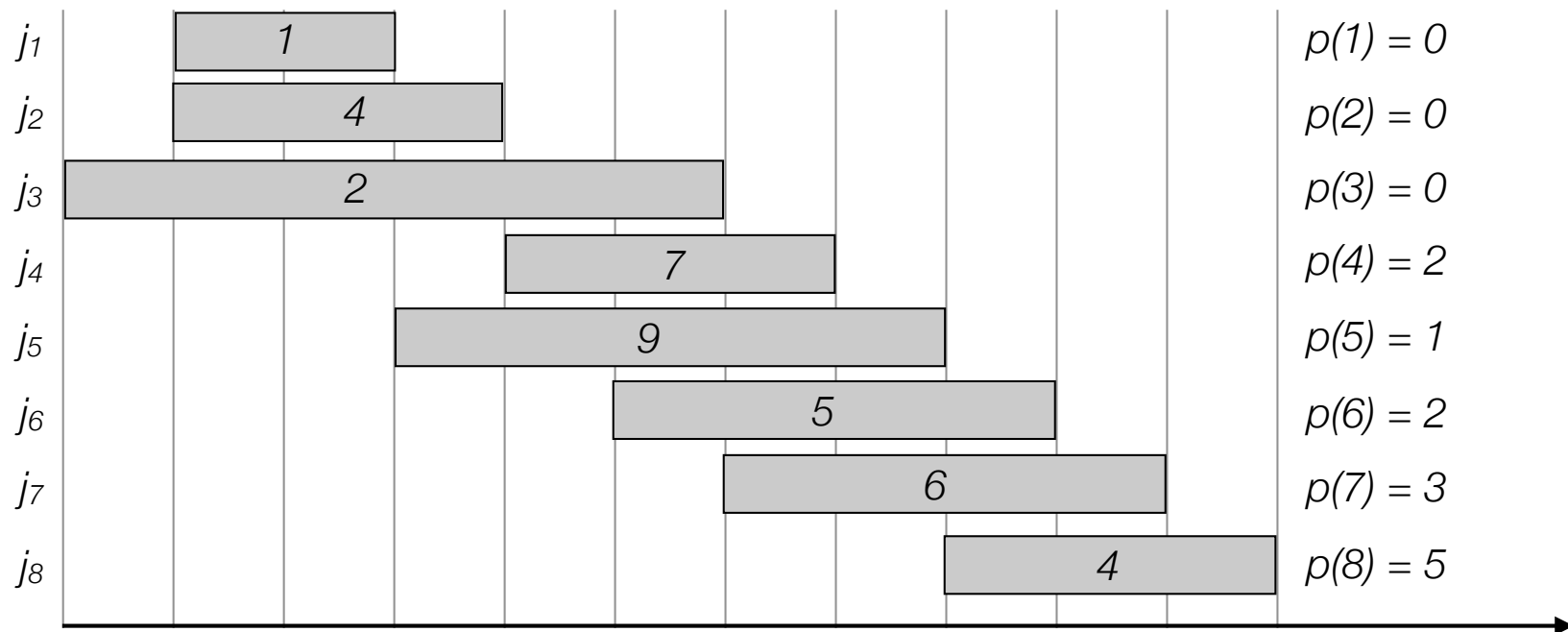
```
Compute  $p[1], p[2], \dots, p[n]$ 
```

```
 $M[0] = 0.$ 
```

```
for  $j=1$  to  $n$ 
```

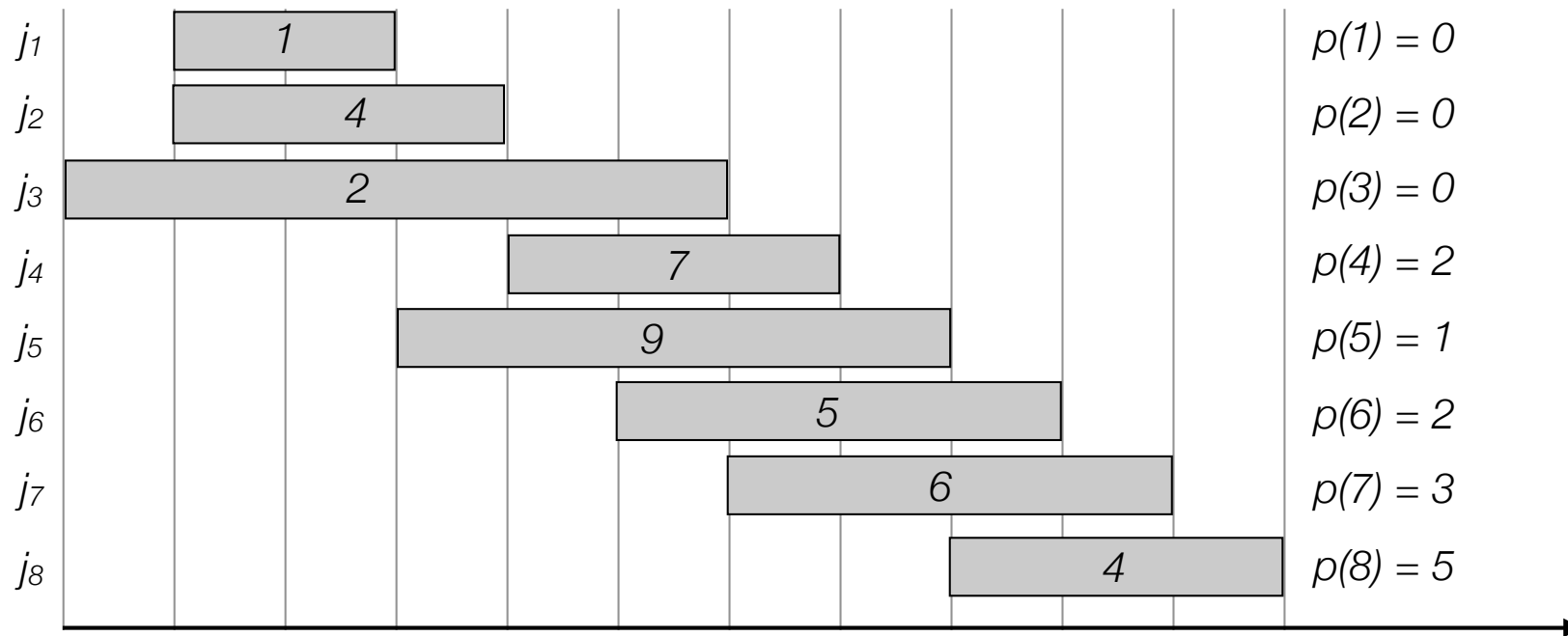
```
     $M[j] = \max(v[j] + M(p[j]), M[j-1])$ 
```

```
return  $M[n]$ 
```



$i$	$M[i]$
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

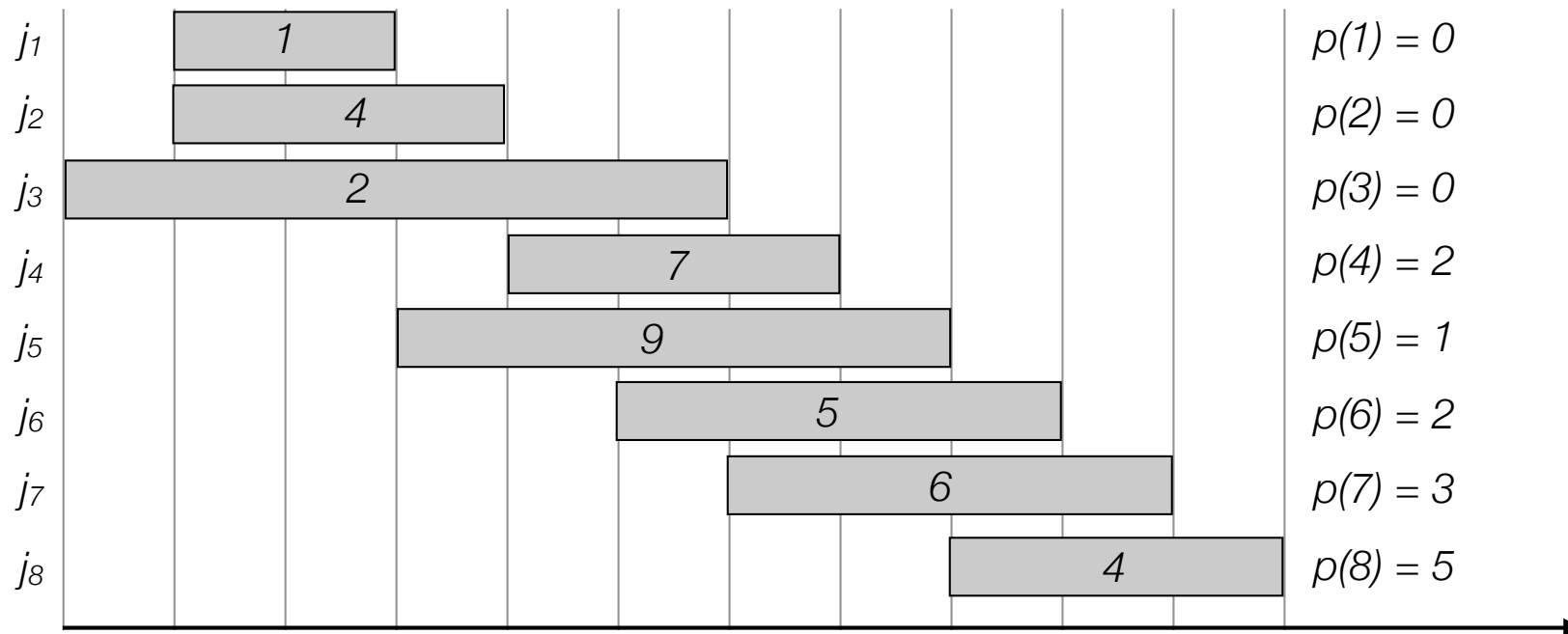


i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15



# Weighted interval scheduling: find solution

```
Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} U Find-Solution(p[j])
else
  return Find-Solution(j-1)
```



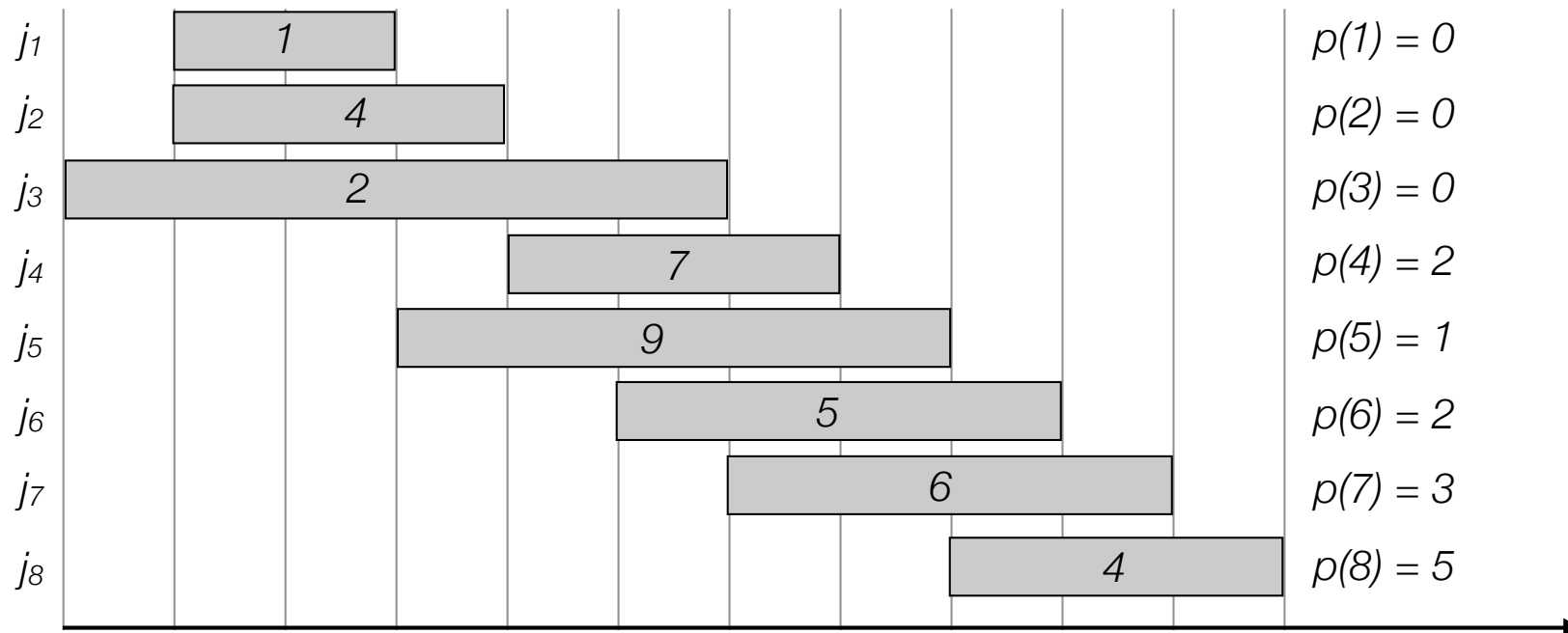
i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```

Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} ∪ Find-Solution(p[j])
else
  return Find-Solution(j-1)
    
```

**Solution =**



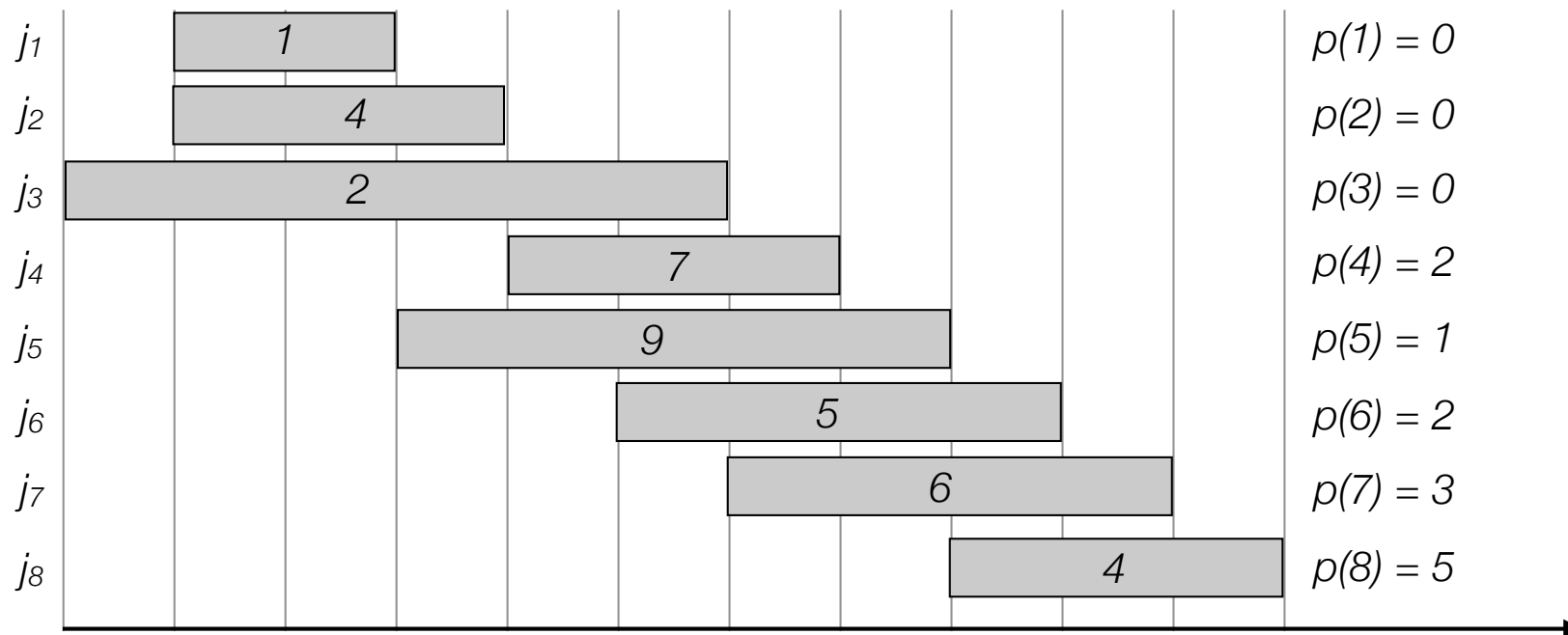
i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```

Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} ∪ Find-Solution(p[j])
else
  return Find-Solution(j-1)
    
```

Solution =



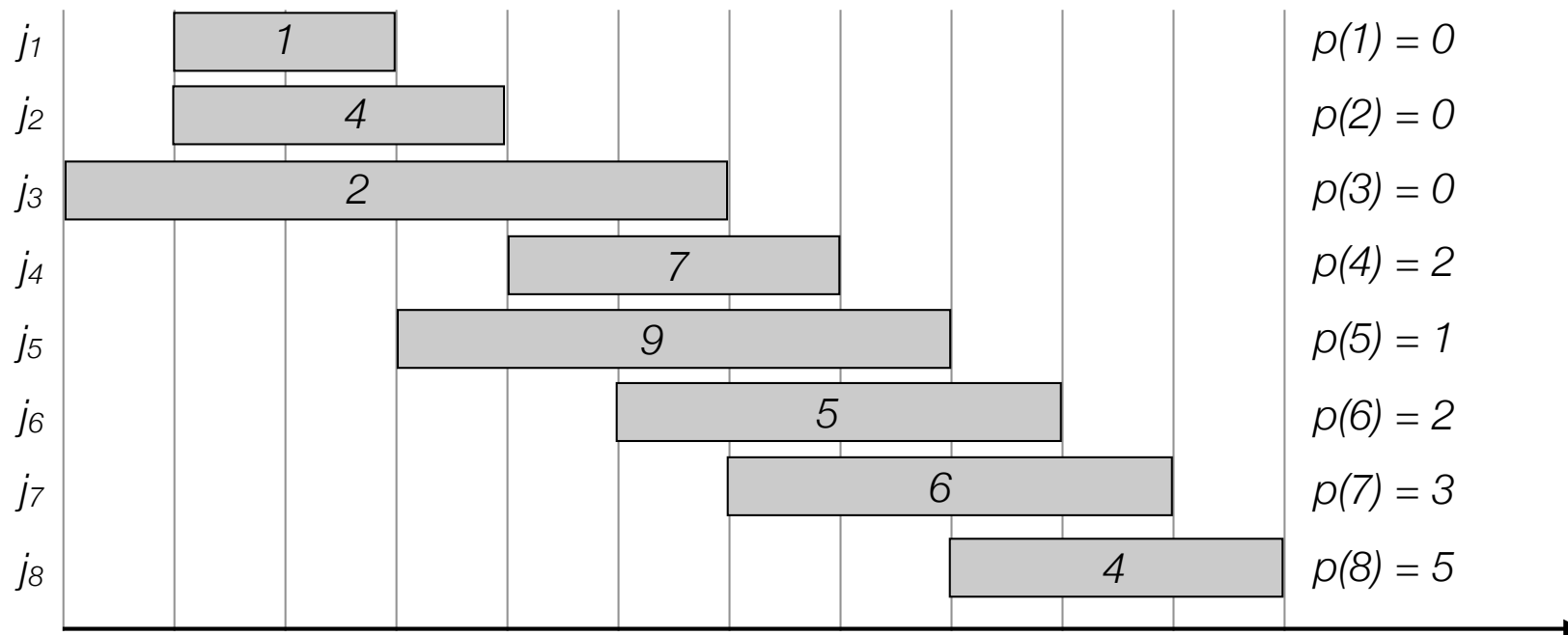
i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```

Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} ∪ Find-Solution(p[j])
else
  return Find-Solution(j-1)
    
```

Solution =



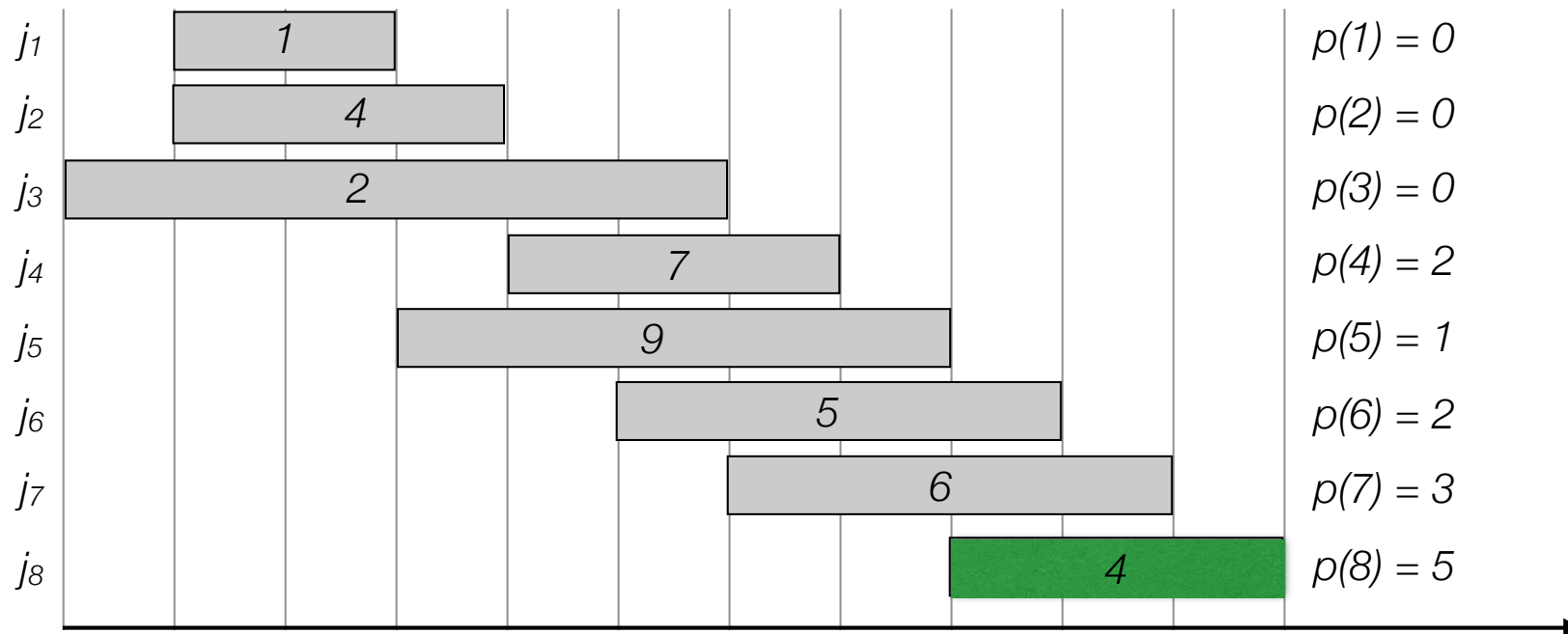
i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```

Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} ∪ Find-Solution(p[j])
else
  return Find-Solution(j-1)
    
```

**Solution = 8**

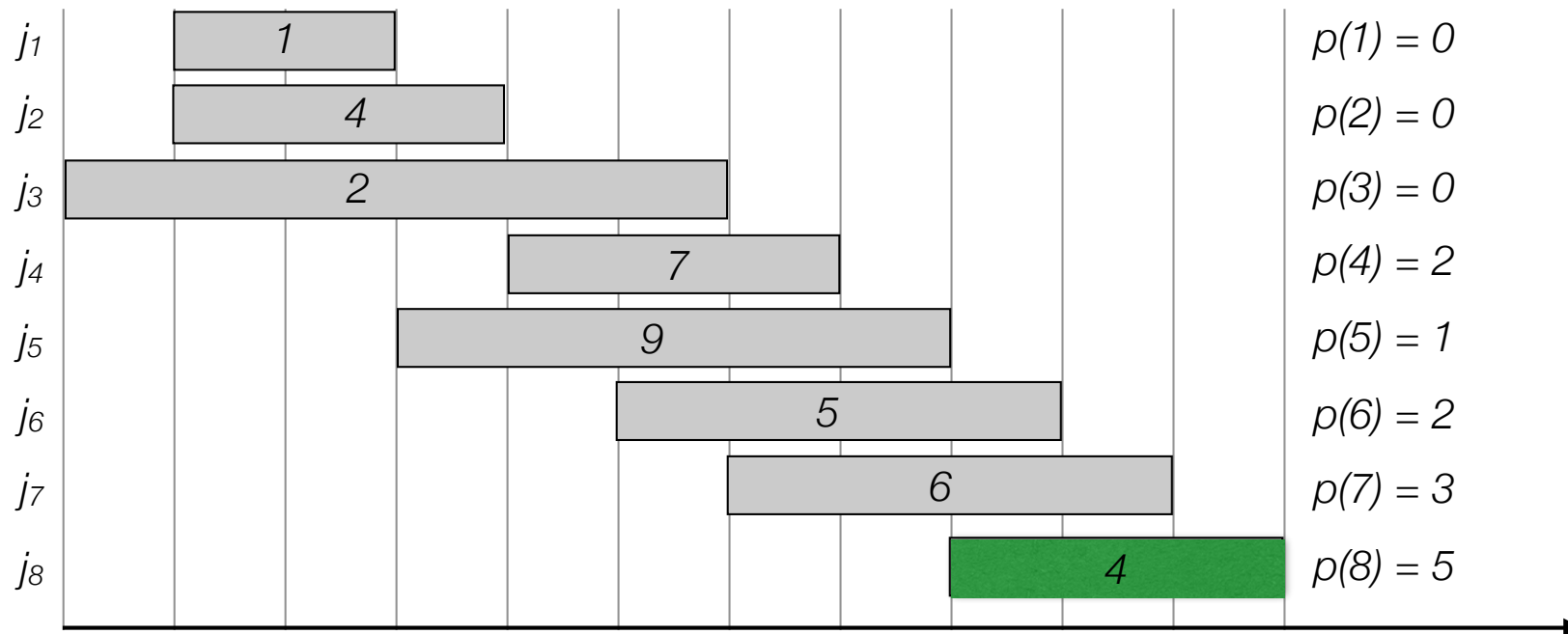


i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```
Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} ∪ Find-Solution(p[j])
else
  return Find-Solution(j-1)
```

**Solution = 8**



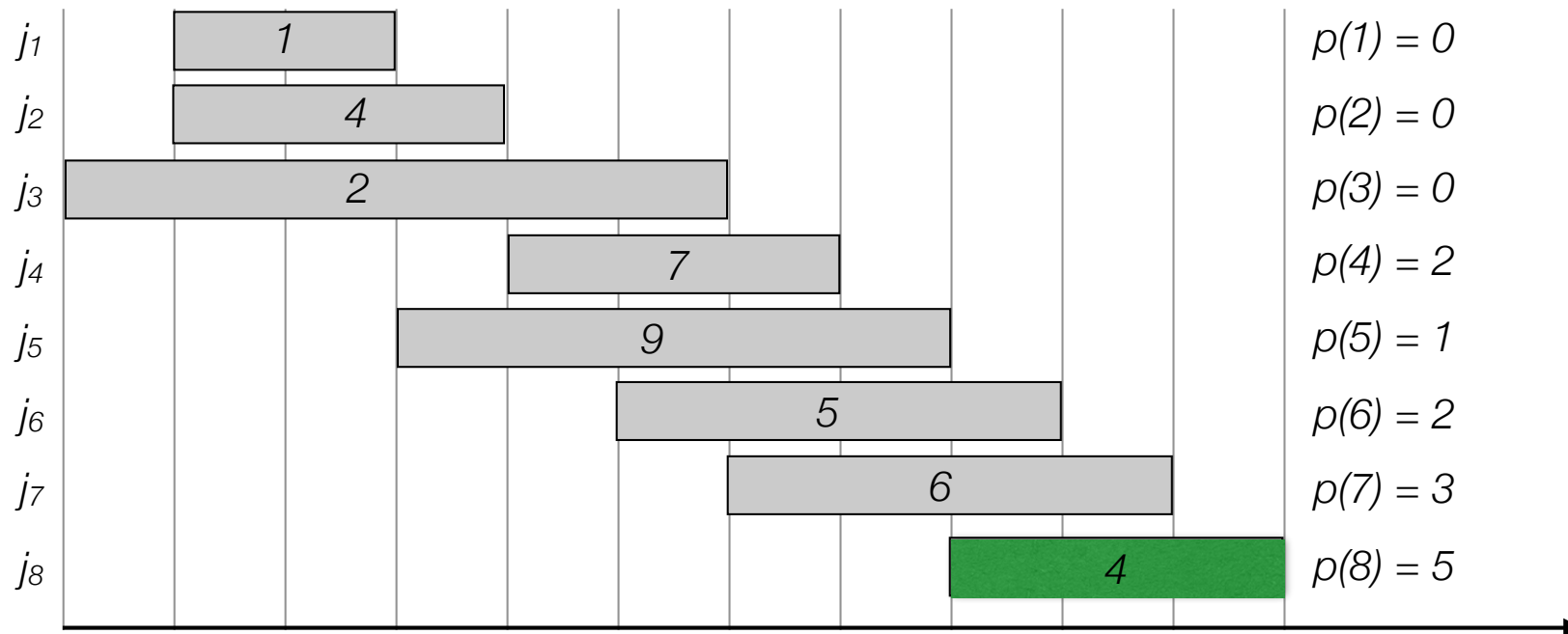
i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```

Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} ∪ Find-Solution(p[j])
else
  return Find-Solution(j-1)
    
```

**Solution = 8**

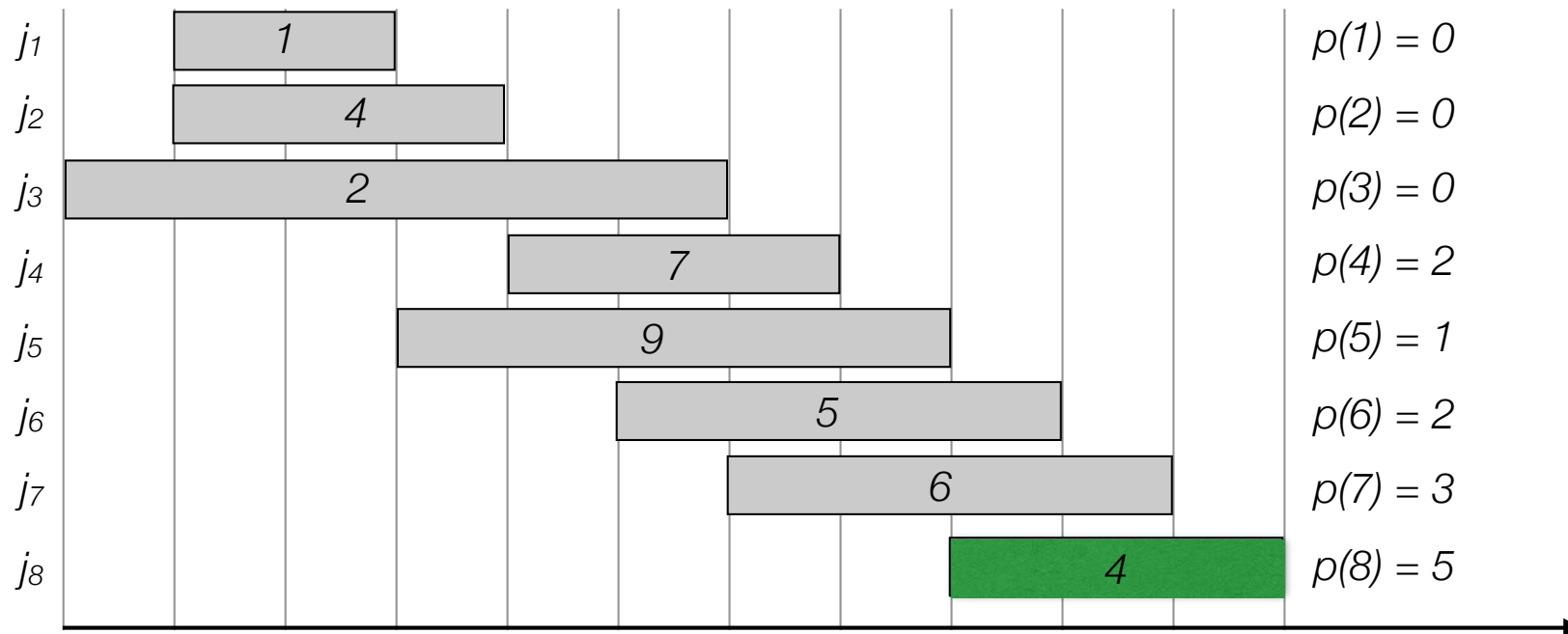


i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```
Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} U Find-Solution(p[j])
else
  return Find-Solution(j-1)
```

**Solution = 8**



$p(1) = 0$   
 $p(2) = 0$   
 $p(3) = 0$   
 $p(4) = 2$   
 $p(5) = 1$   
 $p(6) = 2$   
 $p(7) = 3$   
 $p(8) = 5$

i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

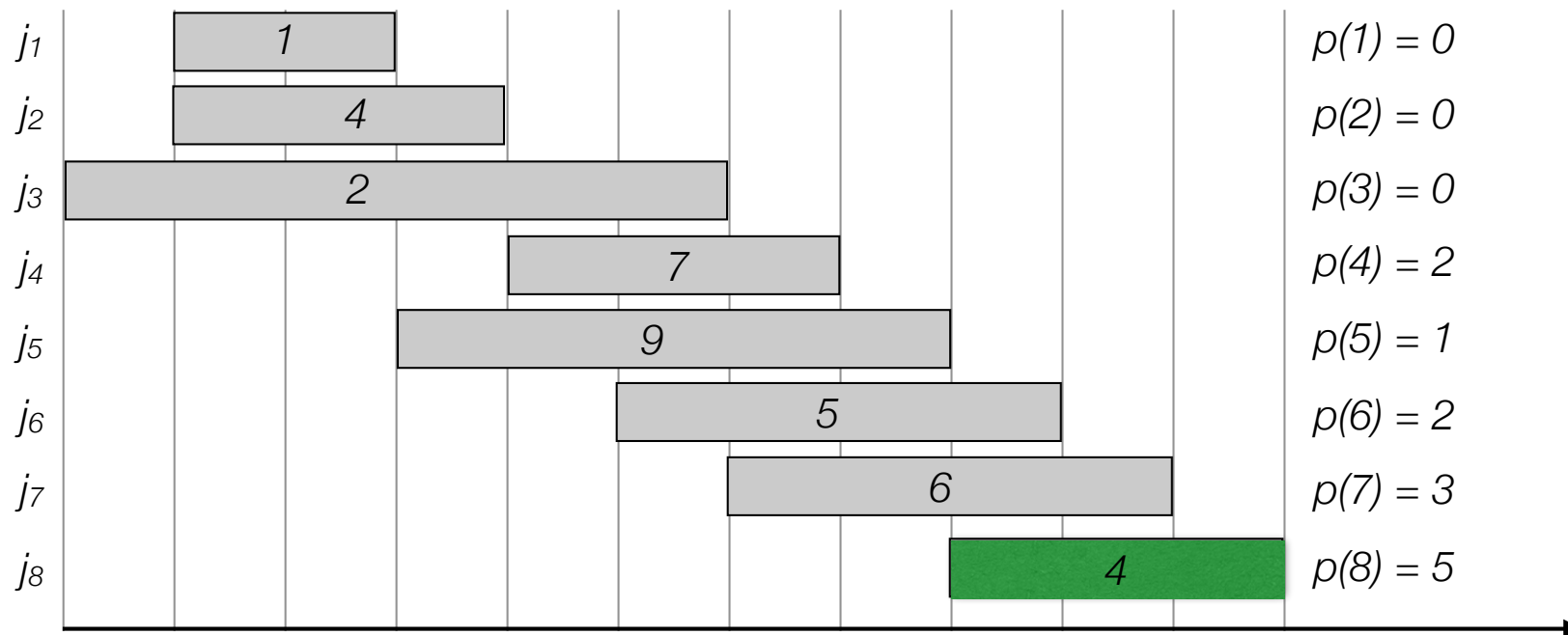


# Weighted interval scheduling: find solution

```

Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} ∪ Find-Solution(p[j])
else
  return Find-Solution(j-1)
    
```

**Solution = 8**



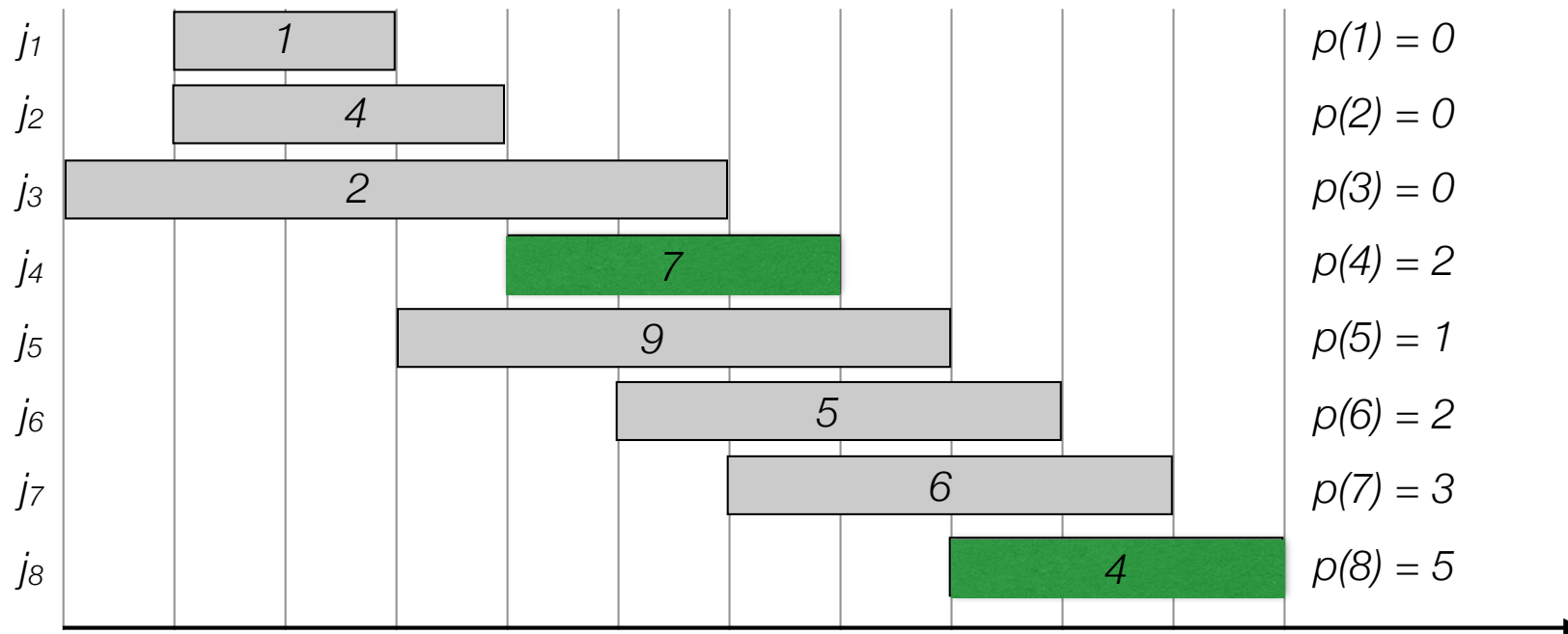
i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```

Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} ∪ Find-Solution(p[j])
else
  return Find-Solution(j-1)
    
```

**Solution = 8, 4**



$p(1) = 0$   
 $p(2) = 0$   
 $p(3) = 0$   
 $p(4) = 2$   
 $p(5) = 1$   
 $p(6) = 2$   
 $p(7) = 3$   
 $p(8) = 5$

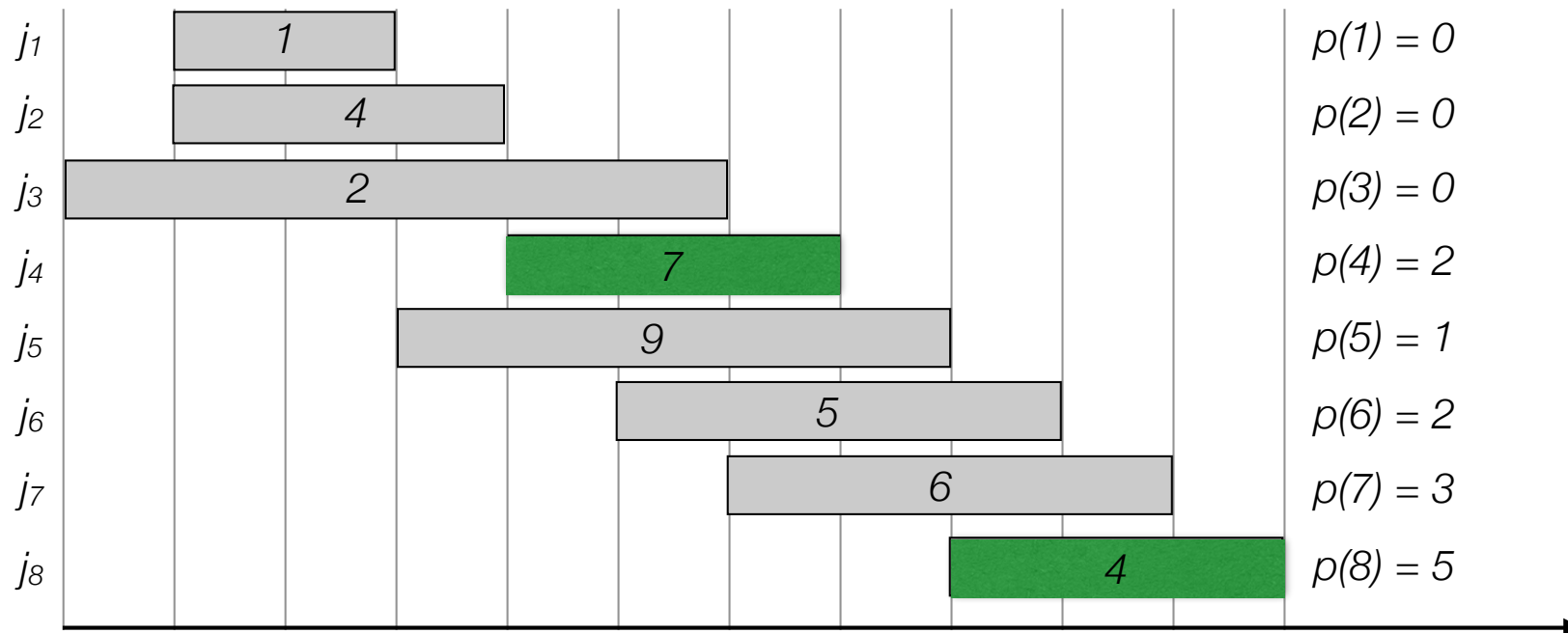
i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```

Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} U Find-Solution(p[j])
else
  return Find-Solution(j-1)
    
```

**Solution = 8, 4**



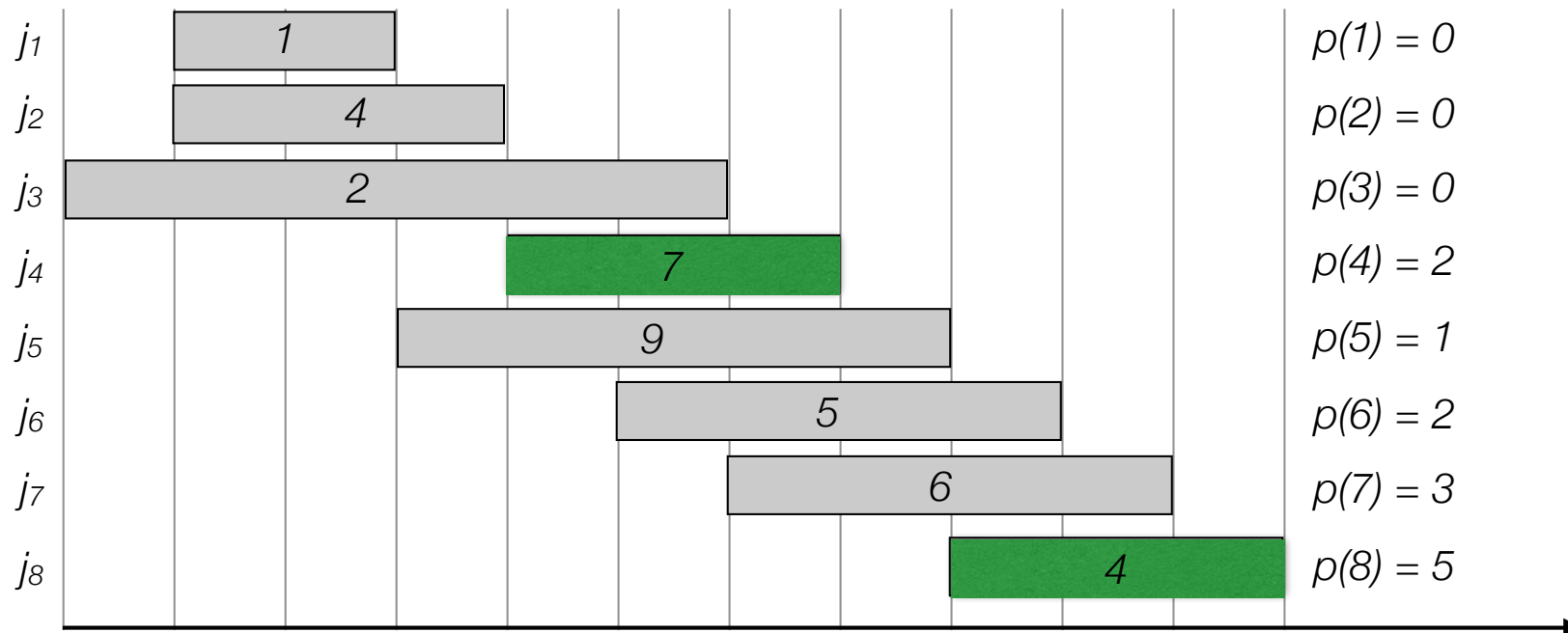
i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```

Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} U Find-Solution(p[j])
else
  return Find-Solution(j-1)
    
```

**Solution = 8, 4**



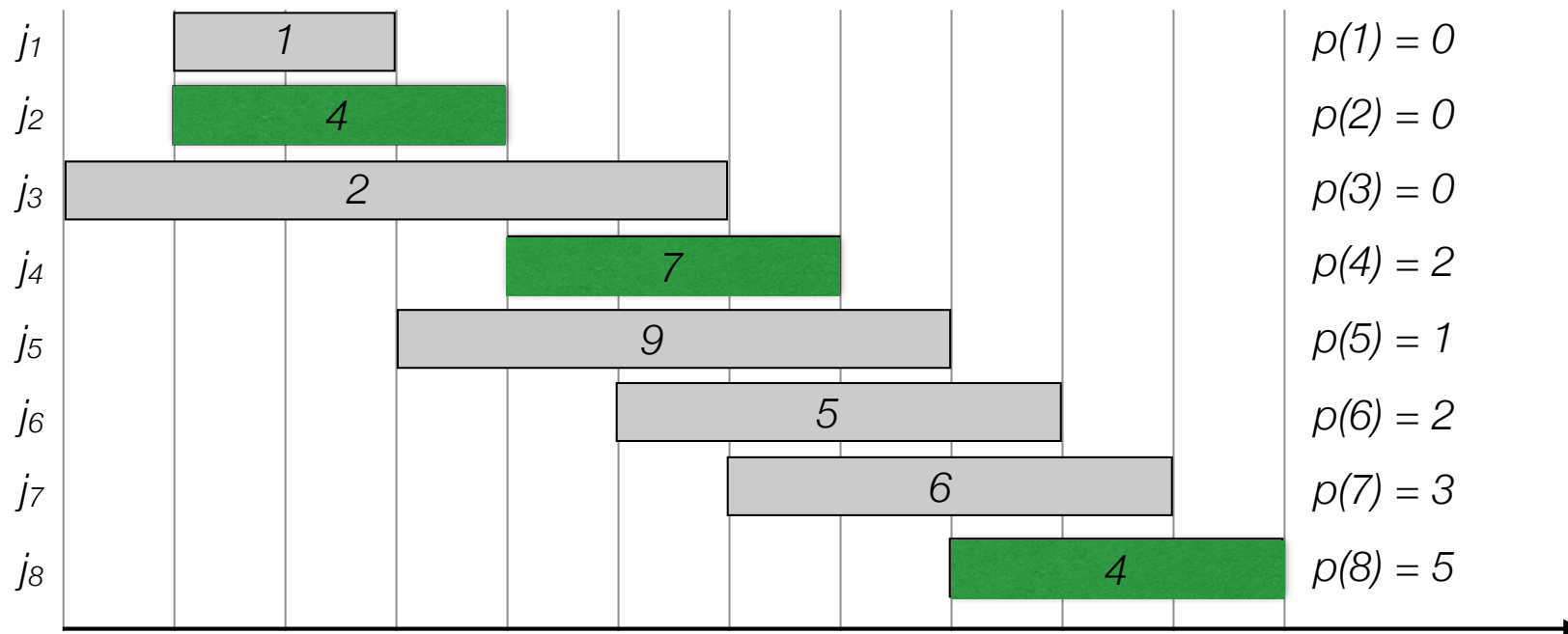
i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```

Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} U Find-Solution(p[j])
else
  return Find-Solution(j-1)
    
```

**Solution = 8, 4, 2**



$p(1) = 0$   
 $p(2) = 0$   
 $p(3) = 0$   
 $p(4) = 2$   
 $p(5) = 1$   
 $p(6) = 2$   
 $p(7) = 3$   
 $p(8) = 5$

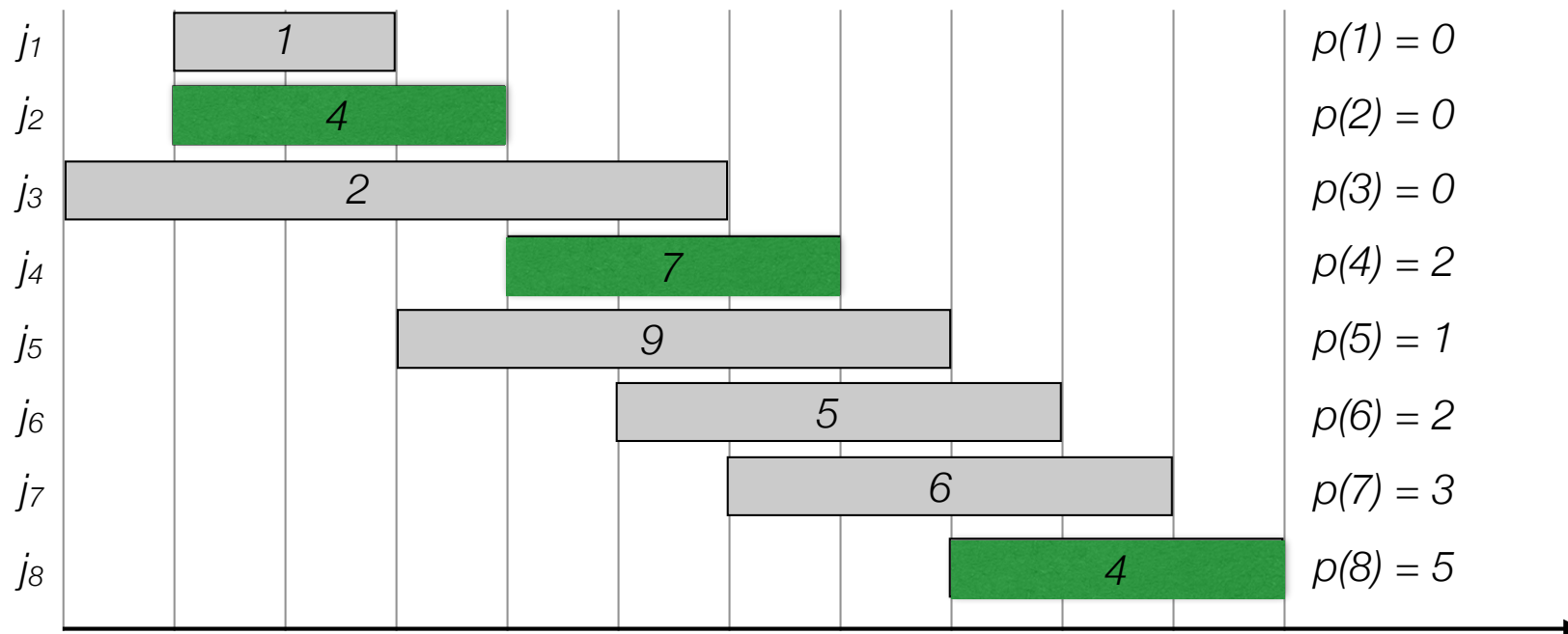
i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Weighted interval scheduling: find solution

```

Find-Solution(j)
if j=0
  Return emptyset
else if M[j] > M[j-1]
  return {j} U Find-Solution(p[j])
else
  return Find-Solution(j-1)
    
```

**Solution = 8, 4, 2**



$p(1) = 0$   
 $p(2) = 0$   
 $p(3) = 0$   
 $p(4) = 2$   
 $p(5) = 1$   
 $p(6) = 2$   
 $p(7) = 3$   
 $p(8) = 5$

i	M[i]
0	0
1	1
2	4
3	4
4	11
5	11
6	11
7	11
8	15

# Subset Sum and Knapsack

# Subset Sum

---

- Subset Sum

- Given  $n$  items  $\{1, \dots, n\}$
- Item  $i$  has weight  $w_i$
- Bound  $W$
- Goal: Select maximum weight subset  $S$  of items so that

$$\sum_{i \in S} w_i \leq W$$

- Example

- $\{2, 5, 8, 9, 12, 18\}$  and  $W = 25$ .
- Solution:  $5 + 8 + 12 = 25$ .



# Subset Sum

---

- $\mathcal{O}$  = optimal solution
- Consider element  $n$ .
  - Either in  $\mathcal{O}$  or not.
    - $n \notin \mathcal{O}$  : Optimal solution using items  $\{1, \dots, n - 1\}$  is equal to  $\mathcal{O}$ .
    - $n \in \mathcal{O}$  : Value of  $\mathcal{O} = w_n +$  weight of optimal solution on  $\{1, \dots, n - 1\}$  with capacity  $W - w_n$ .

- Recurrence

- $\text{OPT}(i, w)$  = optimal solution on  $\{1, \dots, i\}$  with capacity  $w$ .
- From above:

$$\text{OPT}(n, W) = \max(\text{OPT}(n - 1, W), w_n + \text{OPT}(n - 1, W - w_n))$$

- If  $w_n > W$ :

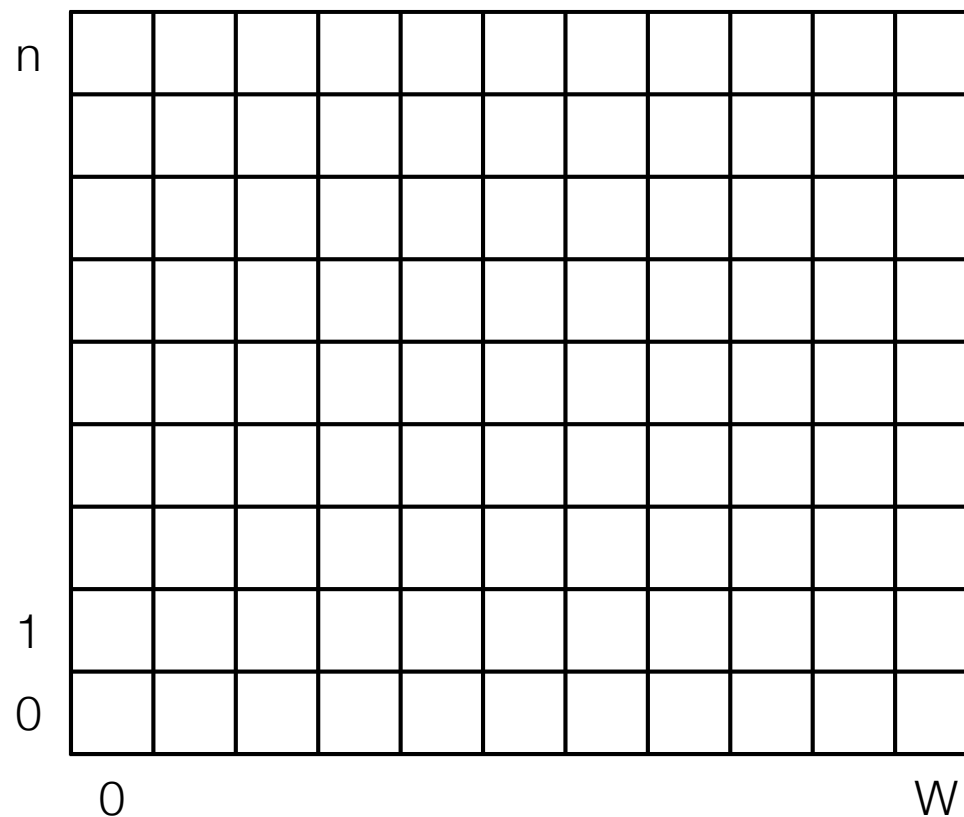
$$\text{OPT}(n, W) = \text{OPT}(n - 1, W)$$

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

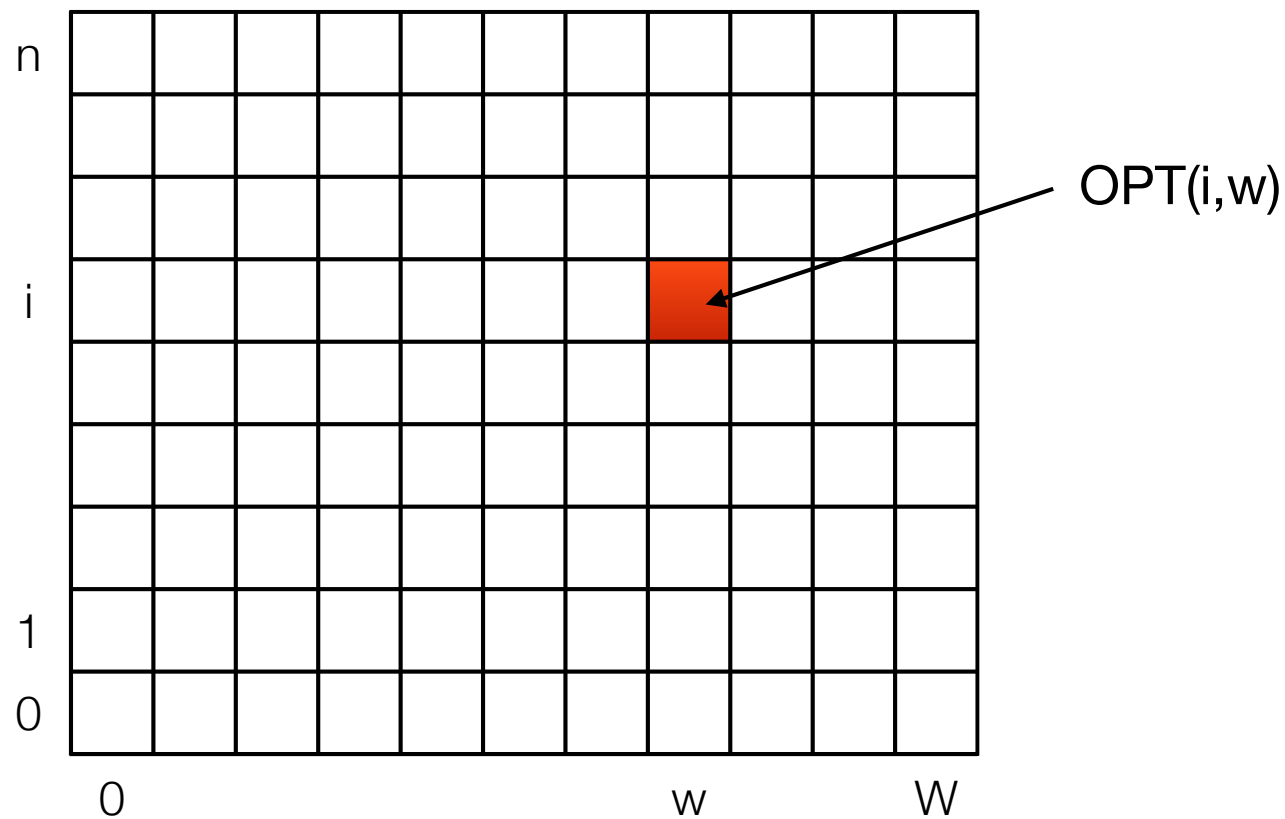


# Subset Sum

---

- Recurrence:

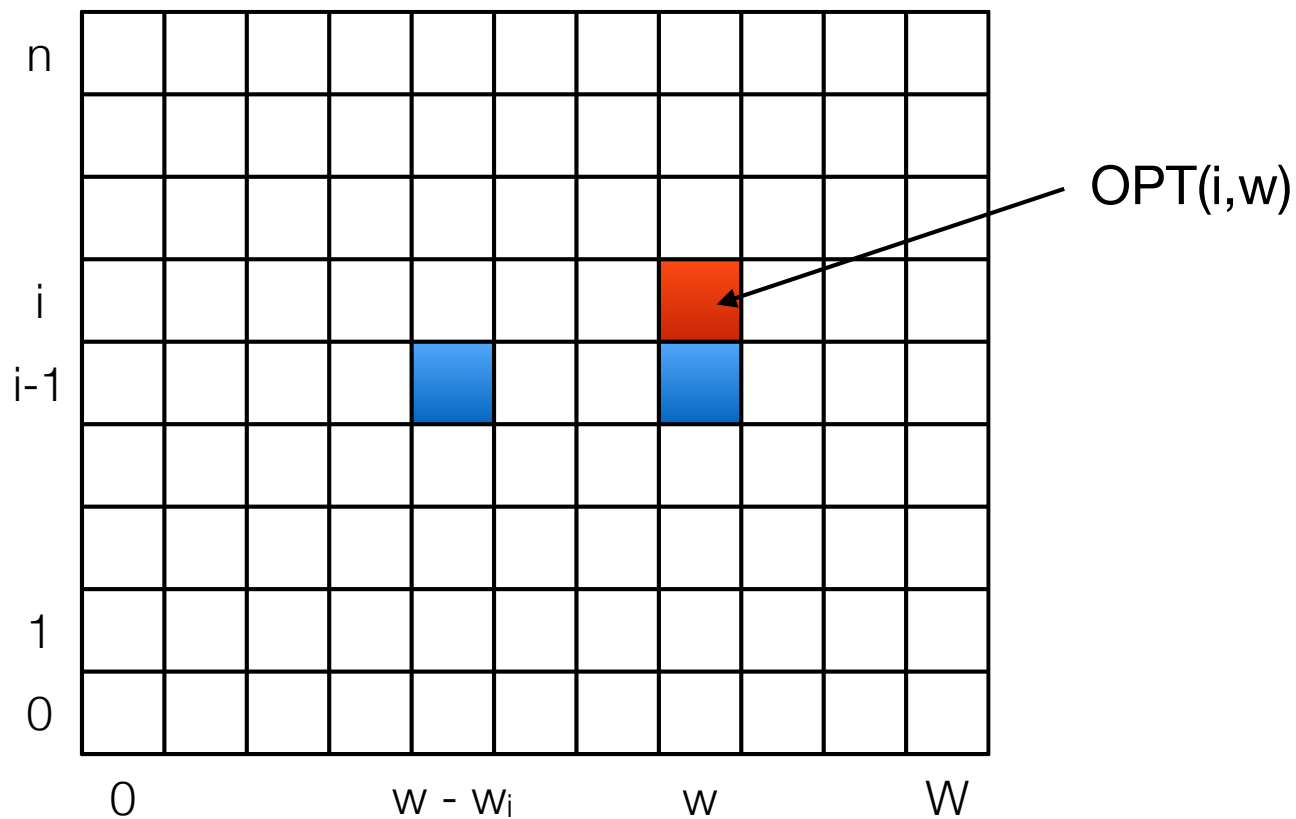
$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$



# Subset Sum

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$



# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

```
Subset-Sum(n, W)
```

```
  Array M[0...n, 0...W]
```

```
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
```

```
  for i = 1 to n
```

```
    for w = 0 to W
```

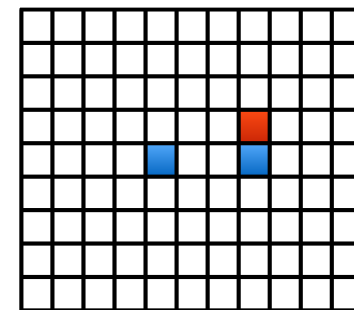
```
      if w < wi
```

```
        M[i, w] = M[i-1, w]
```

```
      else
```

```
        M[i, w] = max(M[i-1, w], wi + M[i-1, w-wi])
```

```
  return M[n, W]
```



# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2													
1	1													
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2													
1	1	0												
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2													
1	1	0												
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12



# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2													
1	1	0	1											
-	0	0	0	0	0	0	0	0	0	0	0	0	0	
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2													
1	1	0	1	1										
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2													
1	1	0	1	1	1	1	1	1	1	1	1	1	1	
-	0	0	0	0	0	0	0	0	0	0	0	0	0	
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2	0												
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2	0	1											
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2	0	1											
1	1	0	1	1	1	1	1	1	1	1	1	1	1	
-	0	0	0	0	0	0	0	0	0	0	0	0	0	
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2	0	1	2										
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2	0	1	2	3									
1	1	0	1	1	1	1	1	1	1	1	1	1	1	
-	0	0	0	0	0	0	0	0	0	0	0	0	0	
		0	1	2	3	4	5	6	7	8	9	10	11	12



# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2	0	1	2	3	3								
1	1	0	1	1	1	1	1	1	1	1	1	1	1	
-	0	0	0	0	0	0	0	0	0	0	0	0	0	
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3													
2	2	0	1	2	3	3	3	3	3	3	3	3	3	
1	1	0	1	1	1	1	1	1	1	1	1	1	1	
-	0	0	0	0	0	0	0	0	0	0	0	0	0	
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3	0	1	2	3	3	5							
2	2	0	1	2	3	3	3	3	3	3	3	3	3	3
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4													
5	3	0	1	2	3	3	5	6						
2	2	0	1	2	3	3	3	3	3	3	3	3	3	3
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and W = 12

9	5													
8	4													
5	3	0	1	2	3	3	5	6	7	8	8	8	8	
2	2	0	1	2	3	3	3	3	3	3	3	3	3	
1	1	0	1	1	1	1	1	1	1	1	1	1	1	
-	0	0	0	0	0	0	0	0	0	0	0	0	0	
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and  $W = 12$

9	5													
8	4	0	1	2	3	4	5	6	7	8	9			
5	3	0	1	2	3	3	5	6	7	8	8	8	8	8
2	2	0	1	2	3	3	3	3	3	3	3	3	3	3
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Example

- {1, 2, 5, 8, 9} and W = 12

9	5	0	1	2	3	3	5	6	7	8	9	10	11	12
8	4	0	1	2	3	3	5	6	7	8	9	10	11	11
5	3	0	1	2	3	3	5	6	7	8	8	8	8	8
2	2	0	1	2	3	3	3	3	3	3	3	3	3	3
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	1	2	3	4	5	6	7	8	9	10	11	12

# Subset Sum

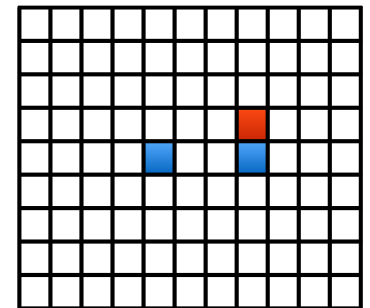
---

- Recurrence:

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), w_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Running time:

- Number of subproblems =  $nW$
- Constant time on each entry  $\Rightarrow O(nW)$
- *Pseudo-polynomial time.*
- Not polynomial in input size:
  - whole input can be described in  $O(n \log n + n \log w)$  bits, where  $w$  is the maximum weight (including  $W$ ) in the instance.





# Knapsack

- Knapsack






- Given  $n$  items  $\{1, \dots, n\}$
- Item  $i$  has weight  $w_i$  and value  $v_i$
- Bound  $W$
- Goal: Select maximum *value* subset  $S$  of items so that

$$\sum_{i \in S} w_i \leq W$$

Optimal solution:  
{3,4} has value  
40

- Example



value	1	6	18	22	28
					
weight	1	2	5	6	7

Capacity 11

# Knapsack

---

- $\mathcal{O}$  = optimal solution

- Consider element  $n$ .



- Either in  $\mathcal{O}$  or not.

- $n \notin \mathcal{O}$  : Optimal solution using items  $\{1, \dots, n - 1\}$  is equal to  $\mathcal{O}$ .

- $n \in \mathcal{O}$  : Value of  $\mathcal{O} = v_n +$  value on optimal solution on  $\{1, \dots, n - 1\}$  with capacity  $W - w_n$ .

- Recurrence

- $\text{OPT}(i, w)$  = optimal solution on  $\{1, \dots, i\}$  with capacity  $w$ .

$$\text{OPT}(i, w) = \begin{cases} \text{OPT}(i - 1, w) & \text{if } w < w_i \\ \max(\text{OPT}(i - 1, w), v_i + \text{OPT}(i - 1, w - w_i)) & \text{otherwise} \end{cases}$$

- Running time  $O(nW)$

# Dynamic programming

---

- **First formulate the problem recursively.**
  - Describe the *problem* recursively in a clear and precise way.
  - Give a recursive formula for the problem.
- **Bottom-up**
  - Identify all the subproblems.
  - Choose a memoization data structure.
  - Identify dependencies.
  - Find a good evaluation order.
- **Top-down**
  - Identify all the subproblems.
  - Choose a memoization data structure.
  - Identify base cases.