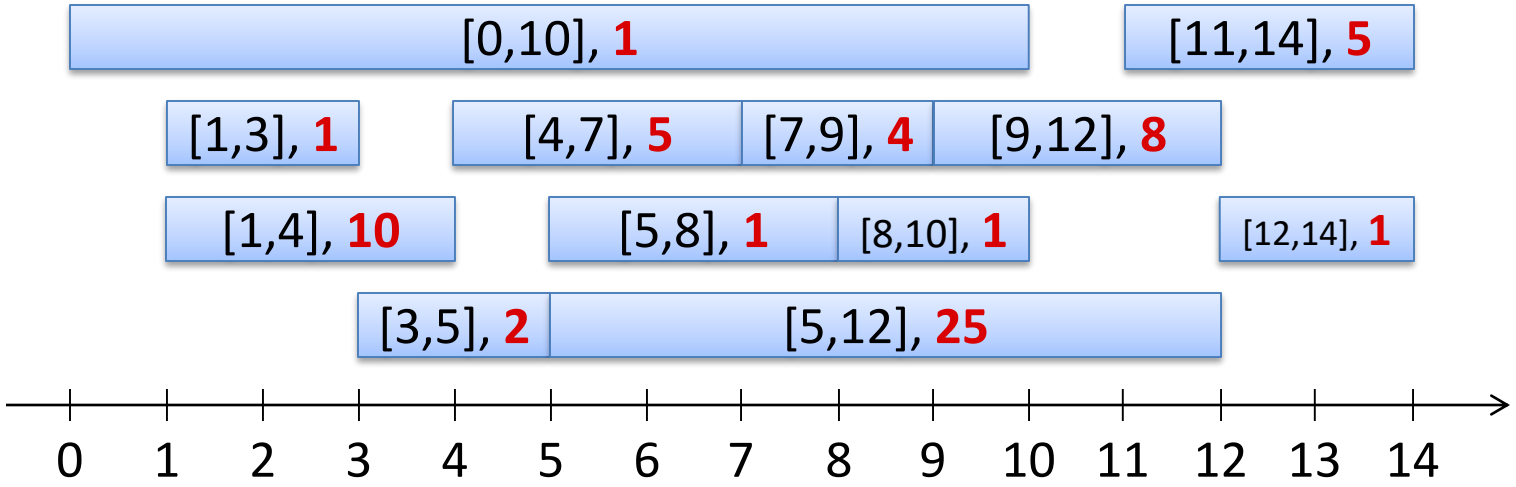
Chapter 3
Dynamic Programming

Algorithm Theory
WS 2016/17

Fabian Kuhn

Weighted Interval Scheduling

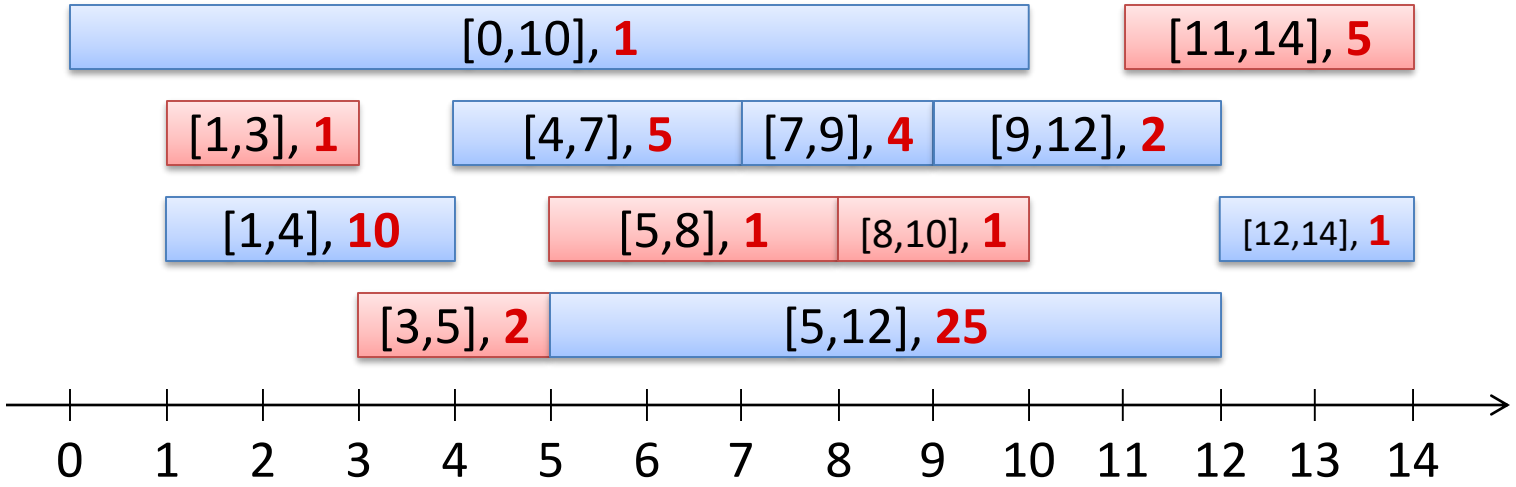
- **Given:** Set of intervals, e.g.
  [0,10],[1,3],[1,4],[3,5],[4,7],[5,8],[5,12],[7,9],[9,12],[8,10],[11,14],[12,14]
- Each interval has a **weight *w***



- **Goal:** Non-overlapping set of intervals of largest possible weight
  - Overlap at boundary ok, i.e., [4,7] and [7,9] are non-overlapping
- **Example:** Intervals are room requests of different importance

Greedy Algorithms

Choose available request with earliest finishing time:



- Algorithm is not optimal any more
  - It can even be arbitrarily bad…
- No greedy algorithm known that works

Solving Weighted Interval Scheduling

- Interval i: start time $s(i)$, finishing time: $f(i)$, weight: $w(i)$
- Assume intervals $1, \dots, n$ are sorted by increasing $f(i)$
  - $0 < f(1) \leq f(2) \leq \cdots \leq f(n)$, for convenience: $f(0) = 0$
- Simple observation:
  Opt. solution contains interval n or it doesn't contain interval n

Solving Weighted Interval Scheduling

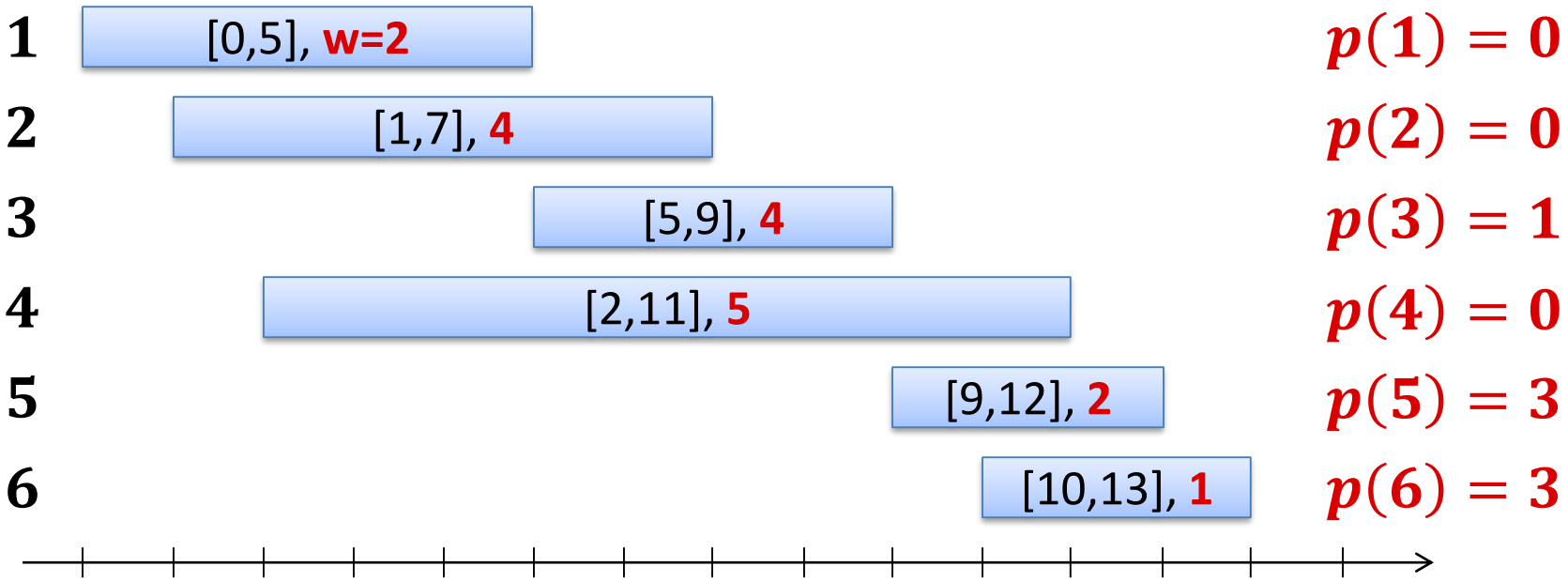
- Interval i: start time $s(i)$, finishing time: $f(i)$, weight: $w(i)$
- Assume intervals $1, \dots, n$ are sorted by increasing $f(i)$
  - $0 < f(1) \leq f(2) \leq \cdots \leq f(n)$, for convenience: $f(0) = 0$
- Simple observation:
  Opt. solution contains interval n or it doesn't contain interval n
- Weight of optimal solution for only intervals $1, \dots, k$: $W(k)$
  Define $p(k) := \max\{i \in \{0, \dots, k-1\} : f(i) \leq s(k)\}$

- Opt. solution does not contain interval n: $\boldsymbol{W(n) = W(n-1)}$

  Opt. solution contains interval n: $\boldsymbol{W(n) = w(n) + W(p(n))}$

Example

Interval:



Recursive Definition of Optimal Solution

- Recall:
  - $W(k)$: weight of optimal solution with intervals $1, \dots, k$
  - $p(k)$: last interval to finish before interval k starts

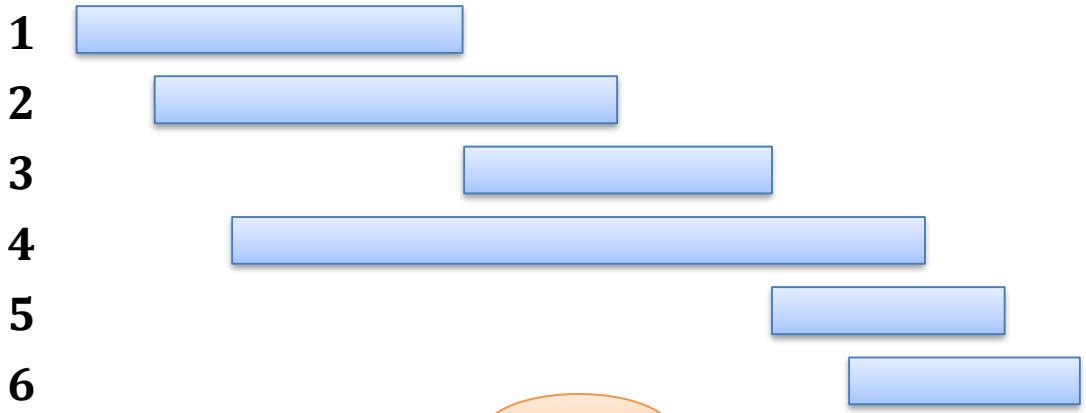
- Recursive definition of optimal weight:

$$\forall k > 1: W(k) = \max\{W(k-1), w(k) + W(p(k))\}$$
$$W(1) = w(1)$$

Immediately gives a simple, recursive algorithm

```
Compute p(k) values for all k
W(k):
    if k == 1:
        x = w(1)
    else:
        x = max{W(k-1), w(k) + W(p(k))}
    return x
```

Running Time of Recursive Algorithm



$p(1) = 0$

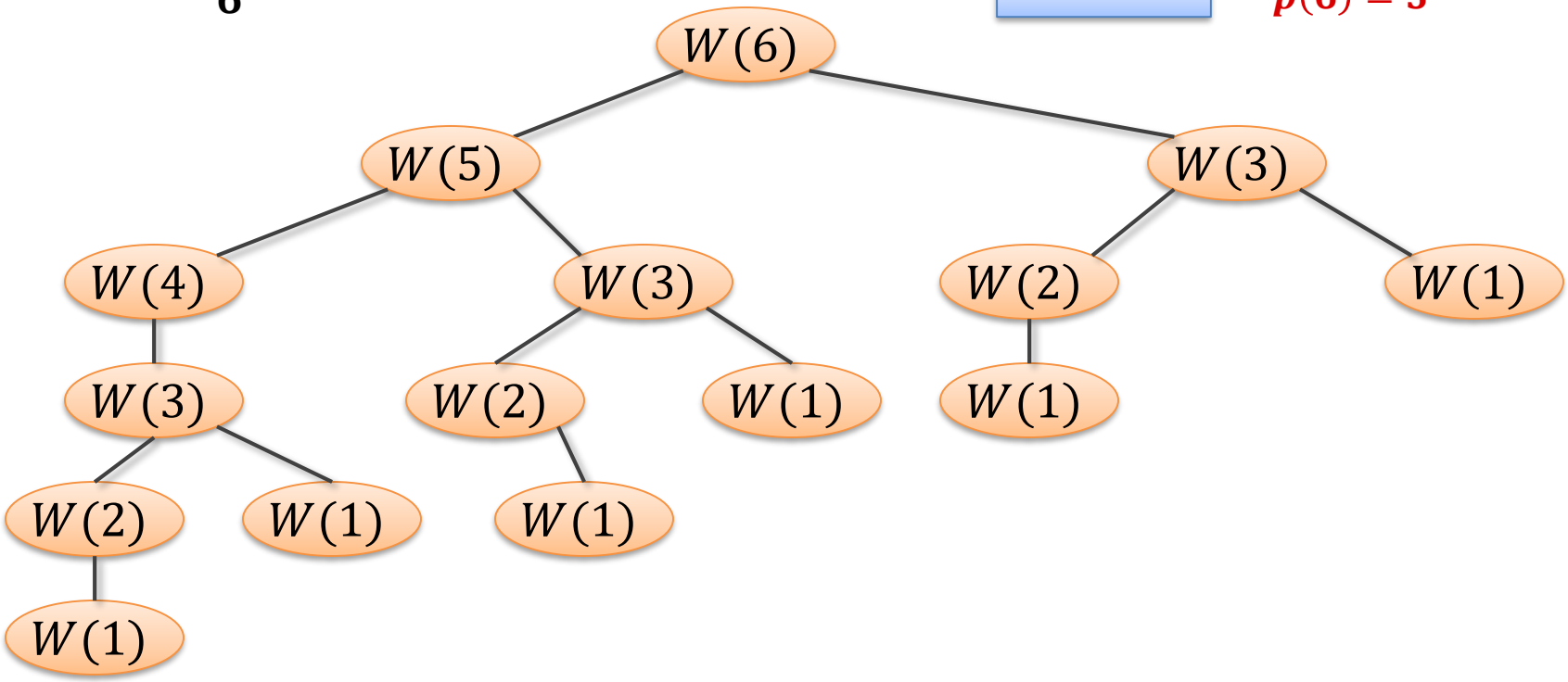
$p(2) = 0$

$p(3) = 1$

$p(4) = 0$

$p(5) = 3$

$p(6) = 3$



Memoizing the Recursion

- Running time of recursive algorithm: exponential!
- But, alg. only solves n different sub-problems: $W(1), \dots, W(n)$
- There is no need to compute them multiple times

Memoization: Store already computed values for future rec. calls

```
Compute p(k) for all k
memo = {};
W(k):
    if k in memo: return memo[k]
    if k == 1:
        x = w(1)
    else:
        x = max{W(k-1), w(k) + W(p(k))}
    memo[k] = x
    return x
```