Lecture

Greedy Algorithms

Wilson Rivera

Greedy algorithms

- A ***greedy algorithm*** always makes the choice that looks best at the moment
  - The hope: a locally optimal choice will lead to a globally optimal solution
  - For some problems, it works
- greedy algorithms **tend to be easier to code**

Activity Selection Problem

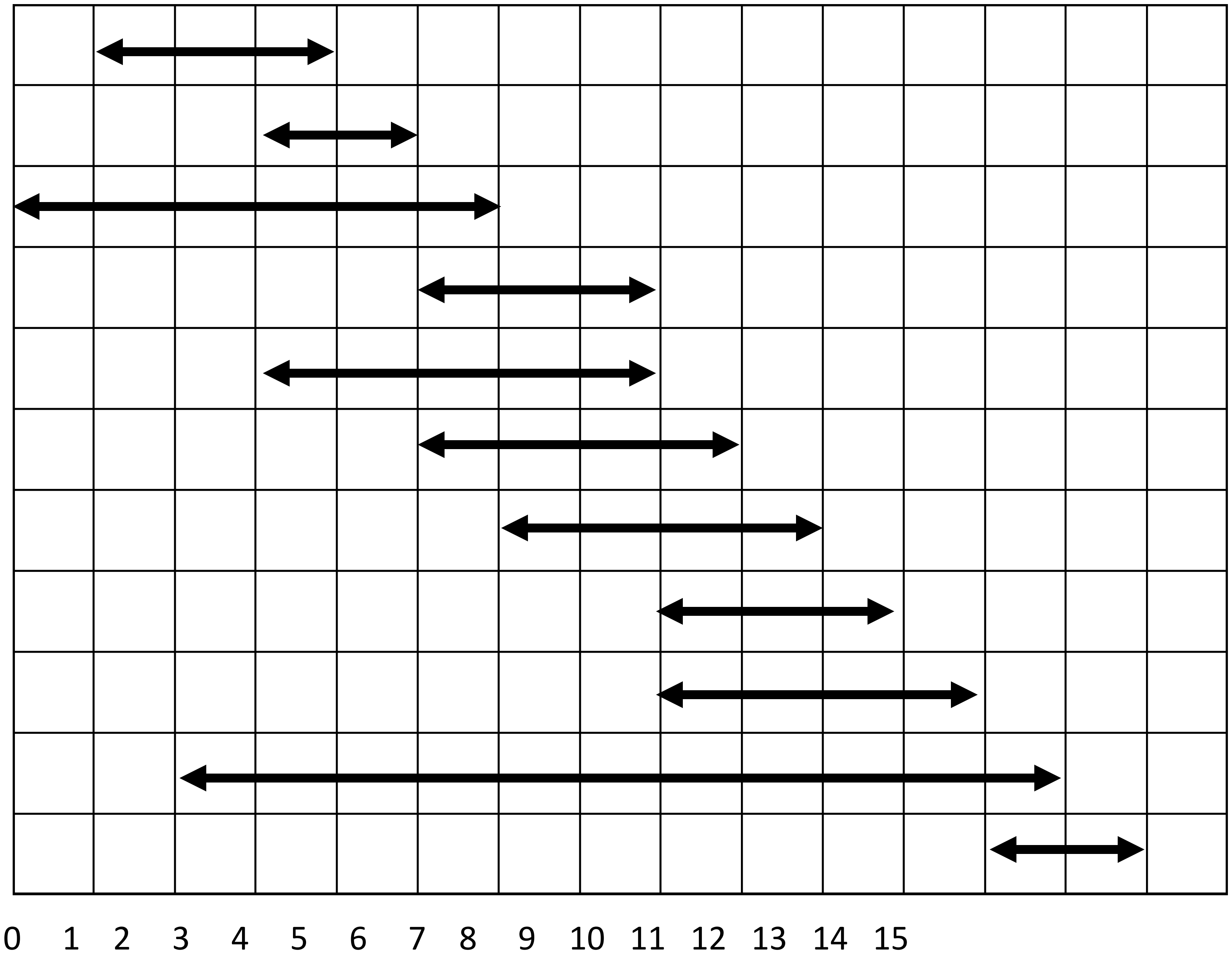
- **Input: A set of activities S = {a_1,…, a_n}**
- Each activity has start time and a finish time
    - $a_k=(s_k, f_k)$
- Two activities are compatible if and only if their interval does not overlap
- **Output: a maximum-size subset of mutually compatible activities**

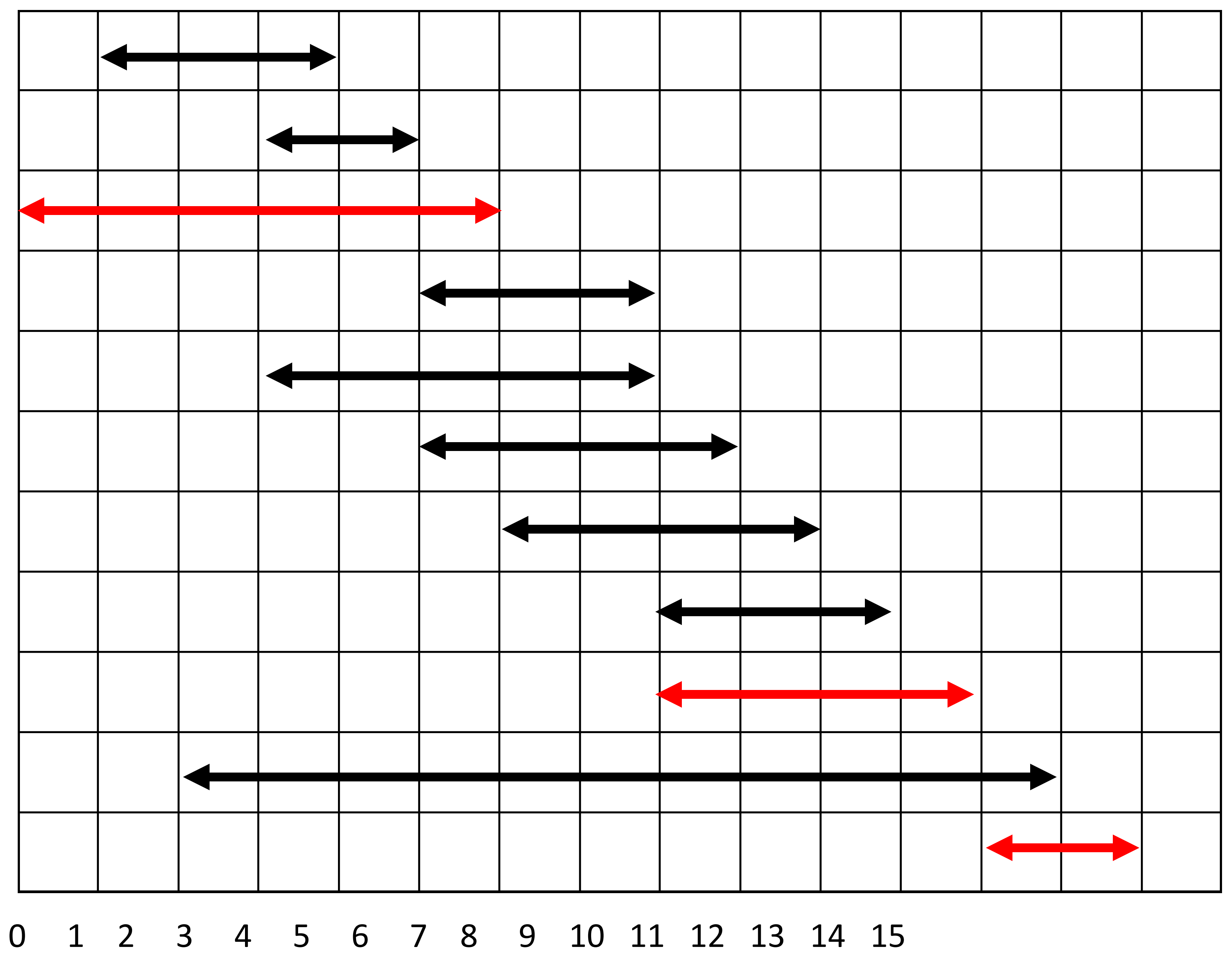
The Activity Selection Problem

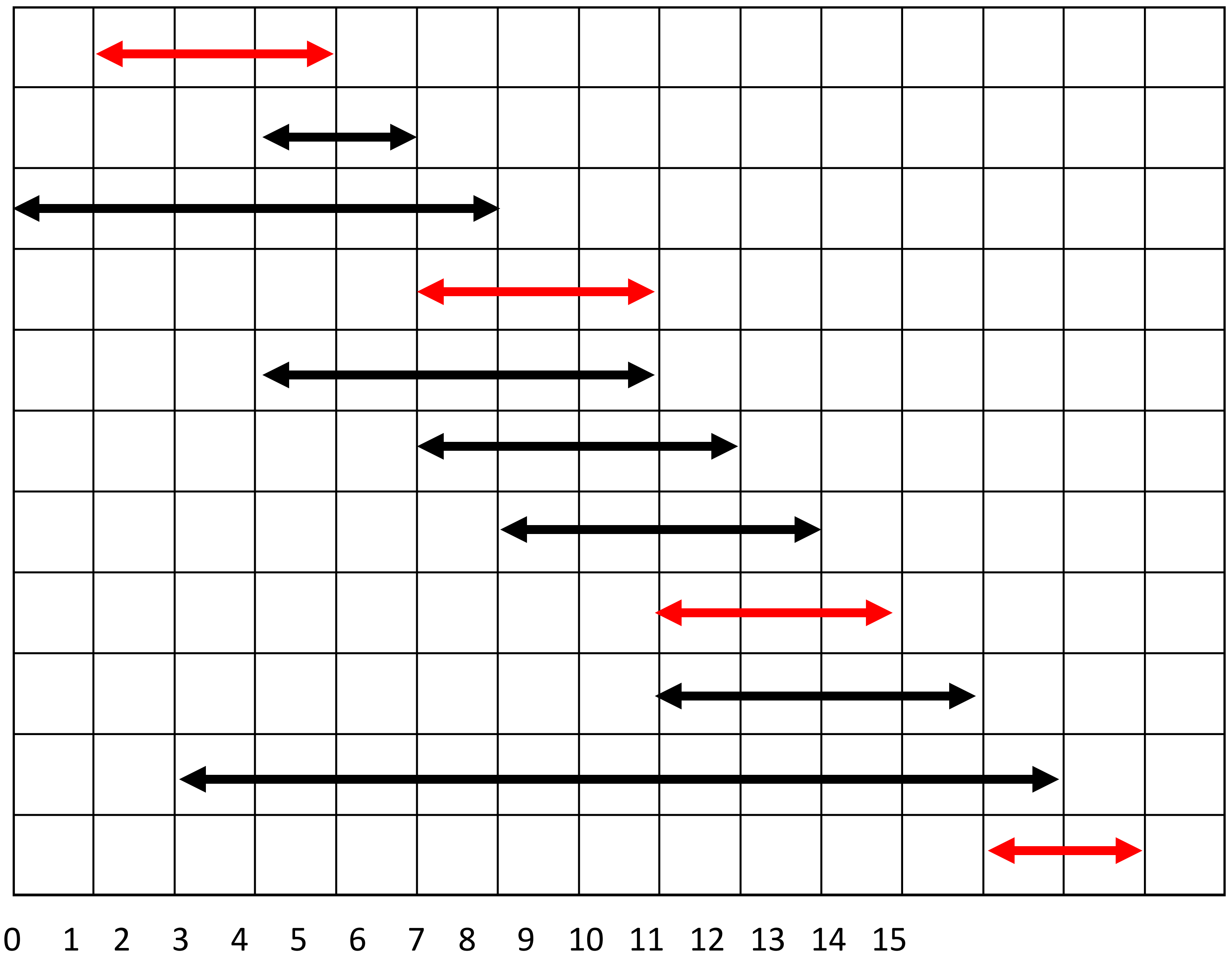
- Here are a set of start and finish times

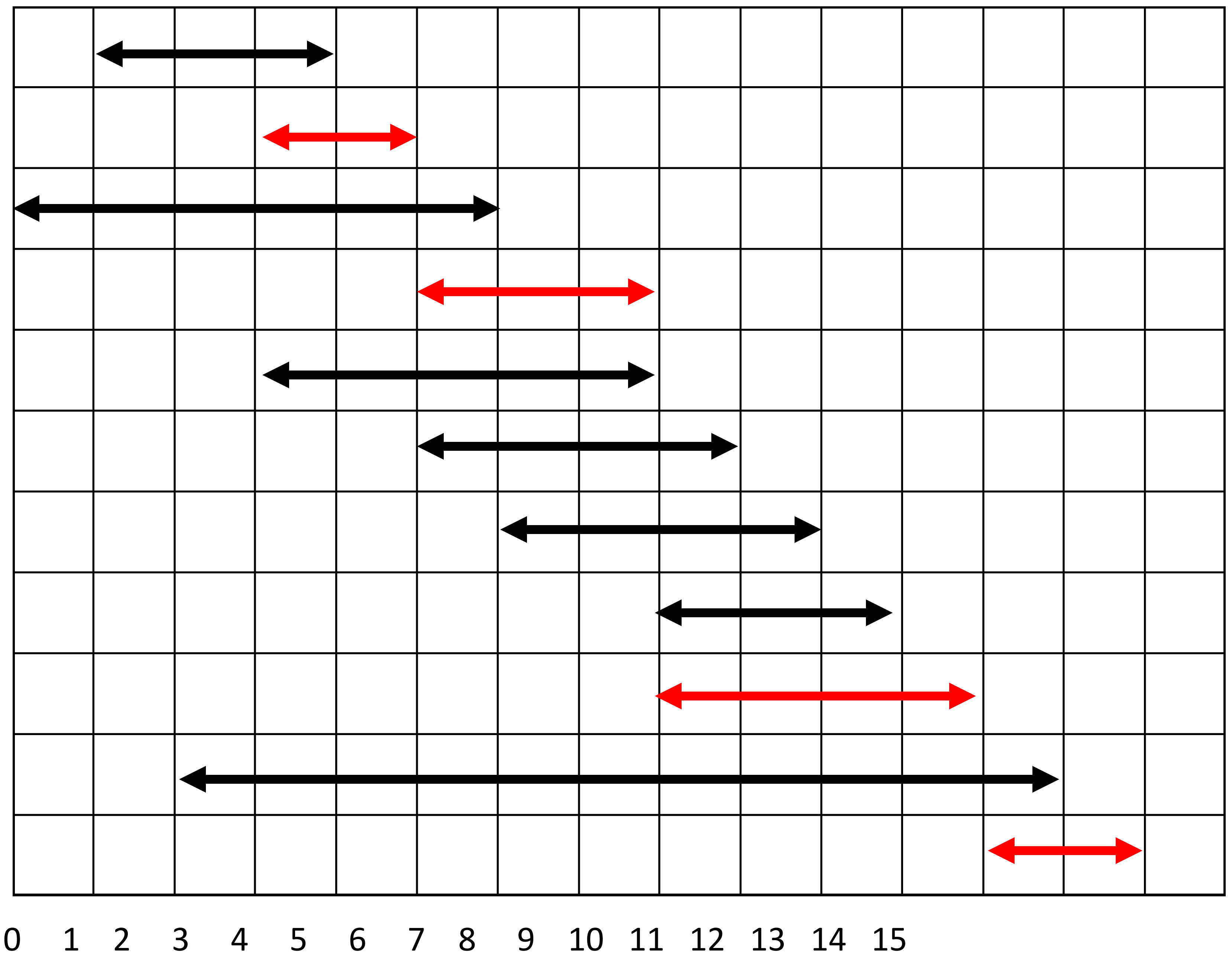
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

- What is the maximum number of activities that can be completed?
  - $\{a_3, a_9, a_{11}\}$ can be completed
  - But so can $\{a_1, a_4, a_8, a_{11}\}$ which is a larger set
  - But it is not unique, consider $\{a_2, a_4, a_9, a_{11}\}$


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Activity Selection: Dynamic Programming

$$S_{i,j} = \{a_k | f_i \leq s_k, f_k \leq s_j\}.$$

set of activities $S_{i,j}$ that start after activity a_i finishes and end before activity a_j starts.

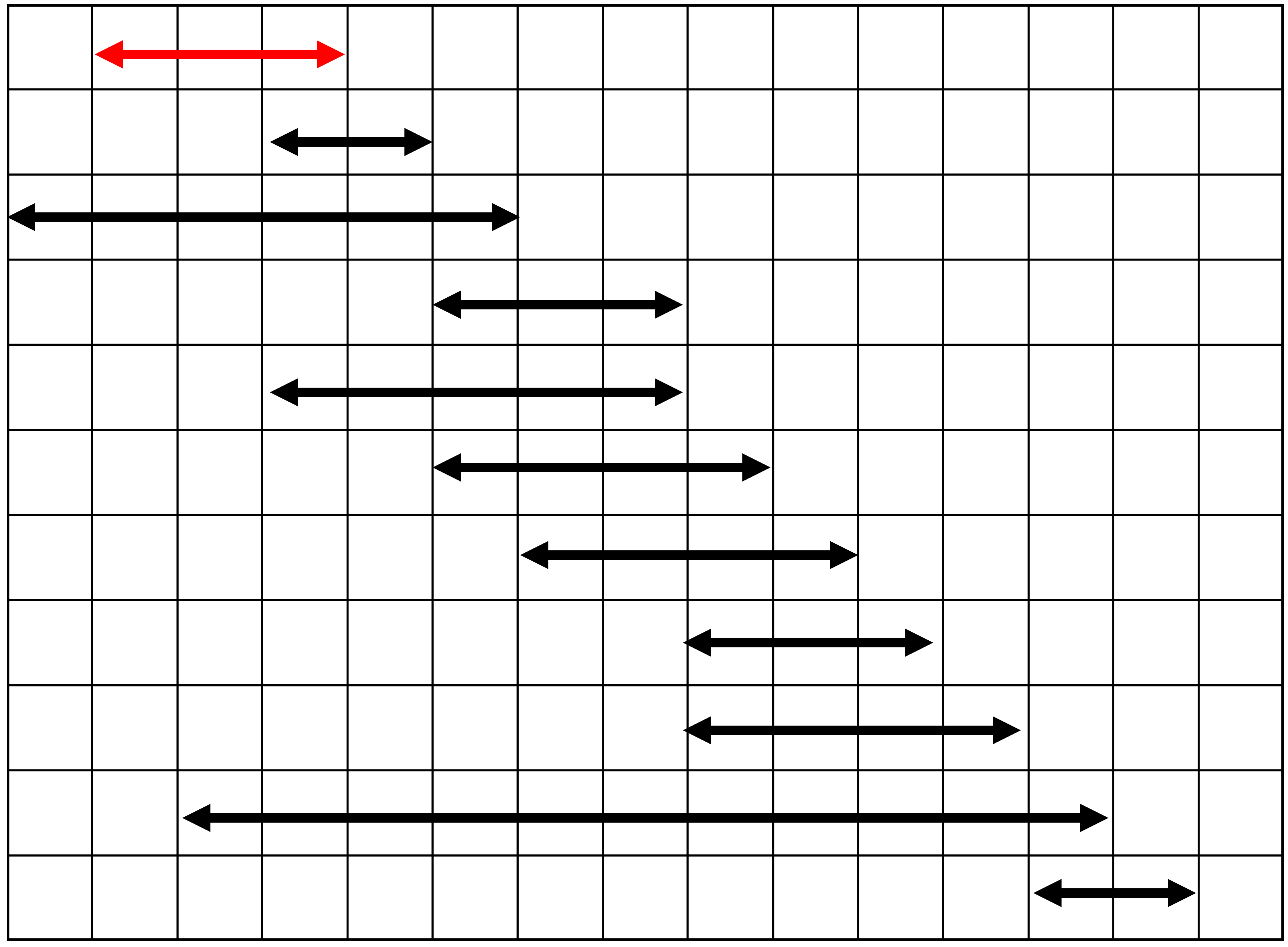
$$|A_{i,j}| = 1 + |A_{i,k}| + |A_{k,j}|$$

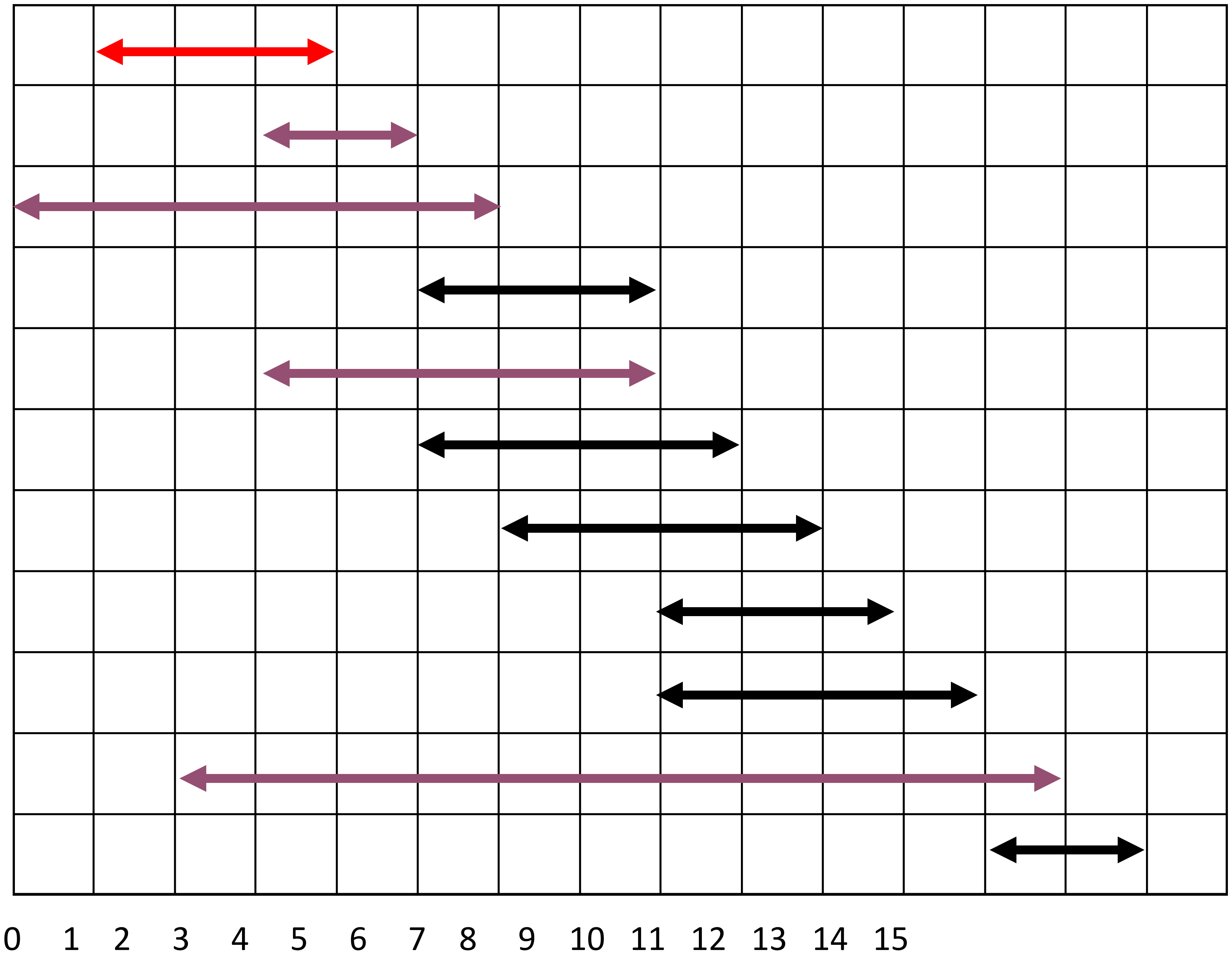
Let $A_{i,j}$ be a maximum subset of non-conflicting activities from the subset $S_{i,j}$.

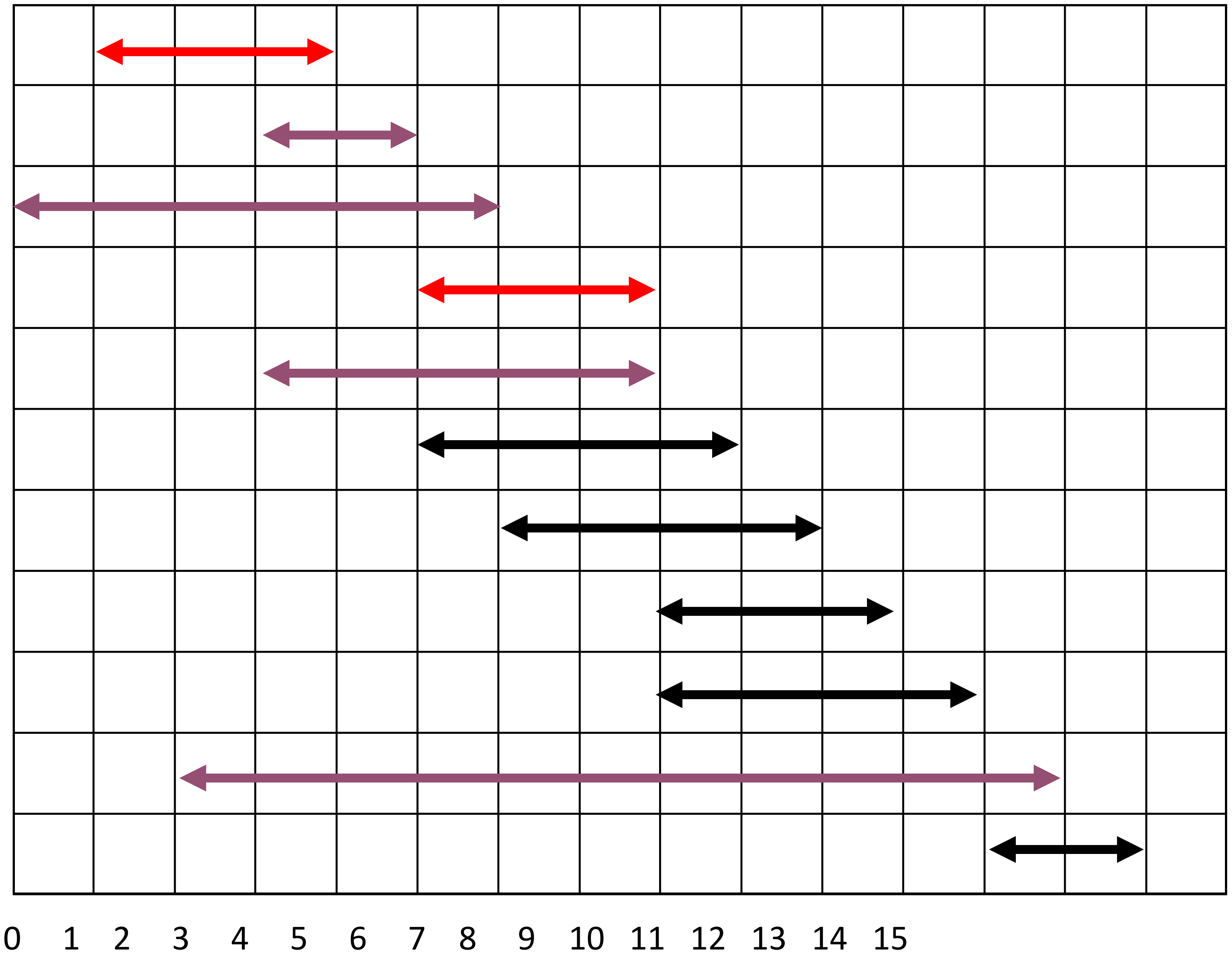
$$|A_{i,j}| = \max_{a_k \in S_{i,j}} 1 + |A_{i,k}| + |A_{k,j}|$$

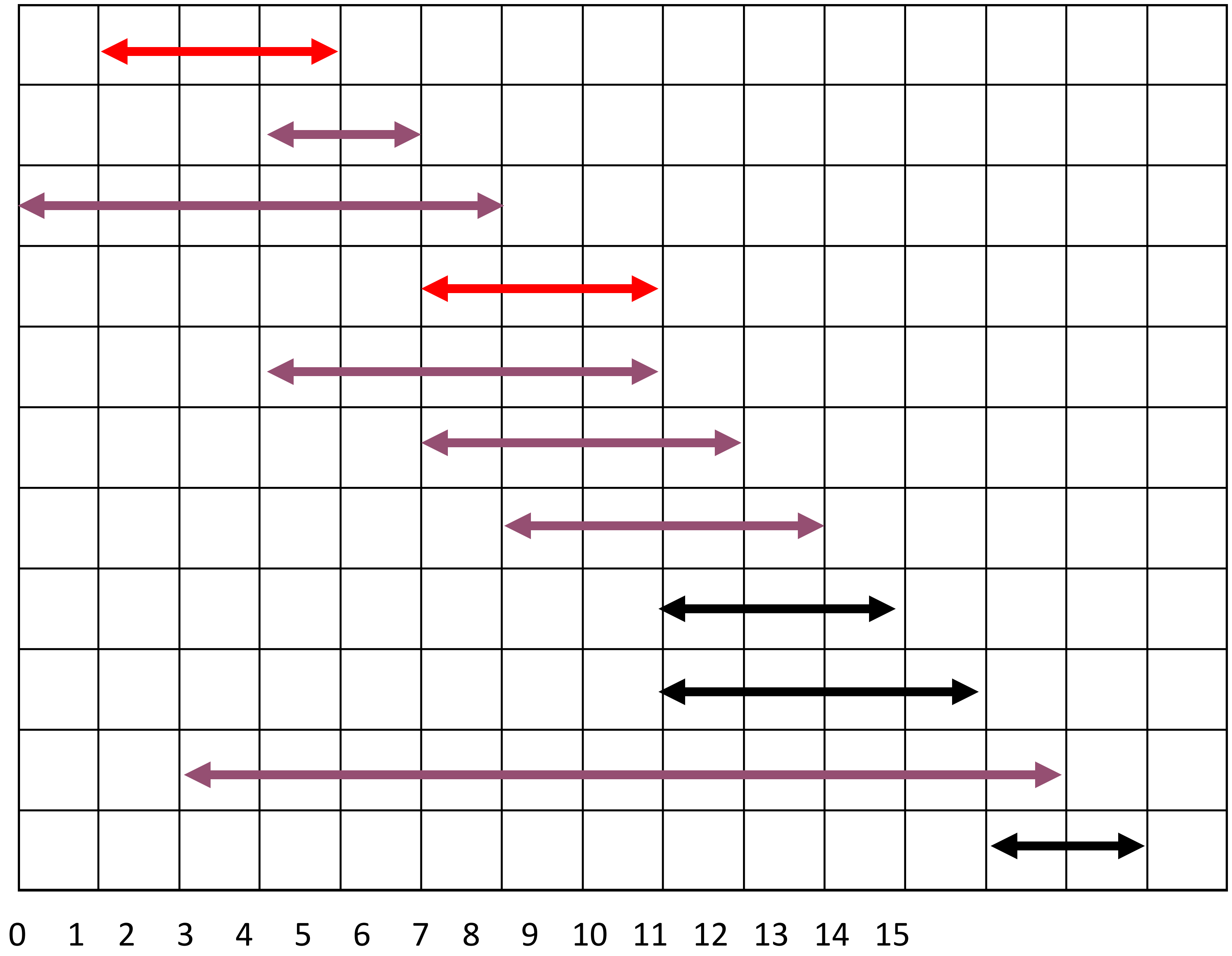
Early Finish Greedy

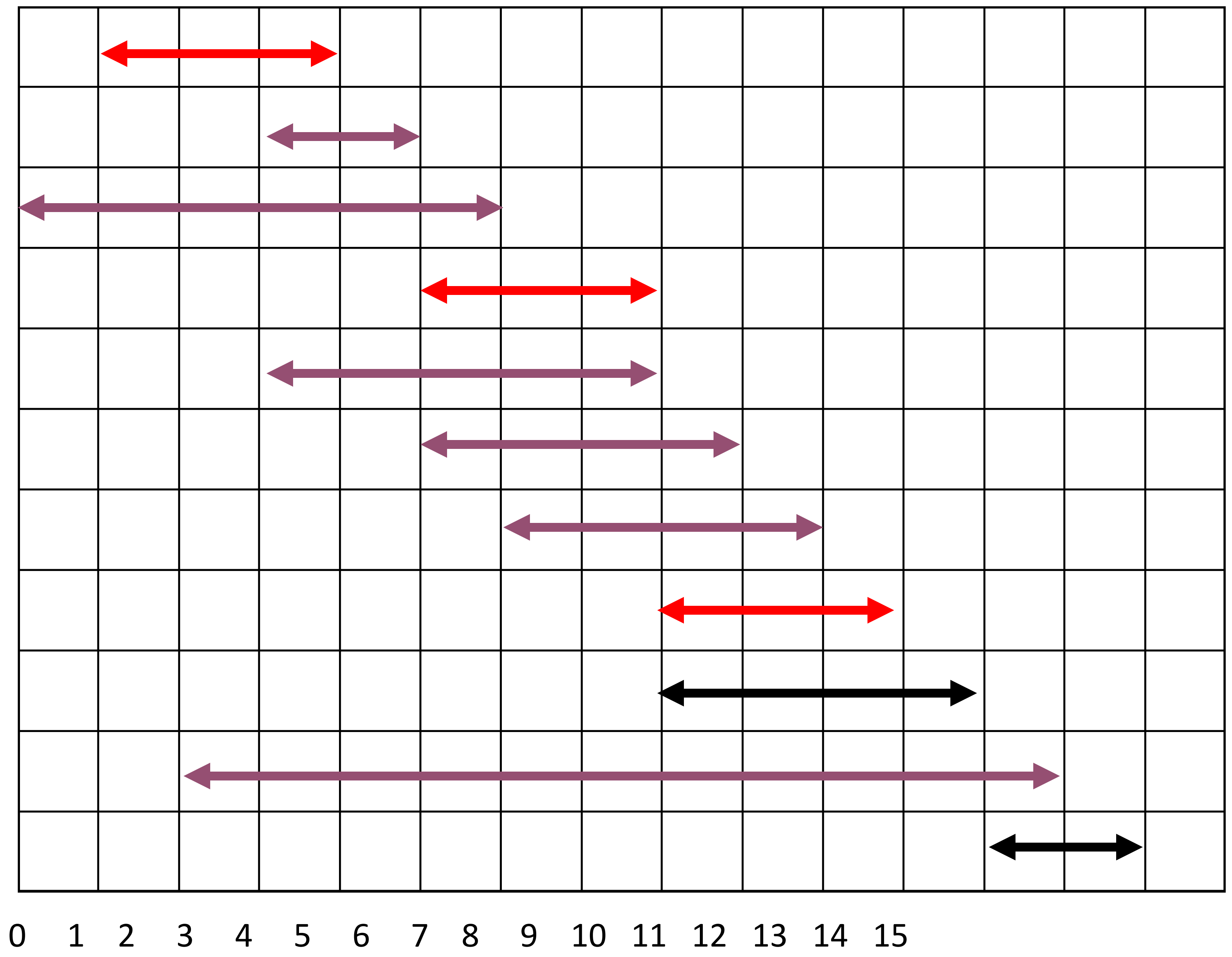
- Select the activity with the earliest finish
- Eliminate the activities that could not be scheduled
- Repeat!

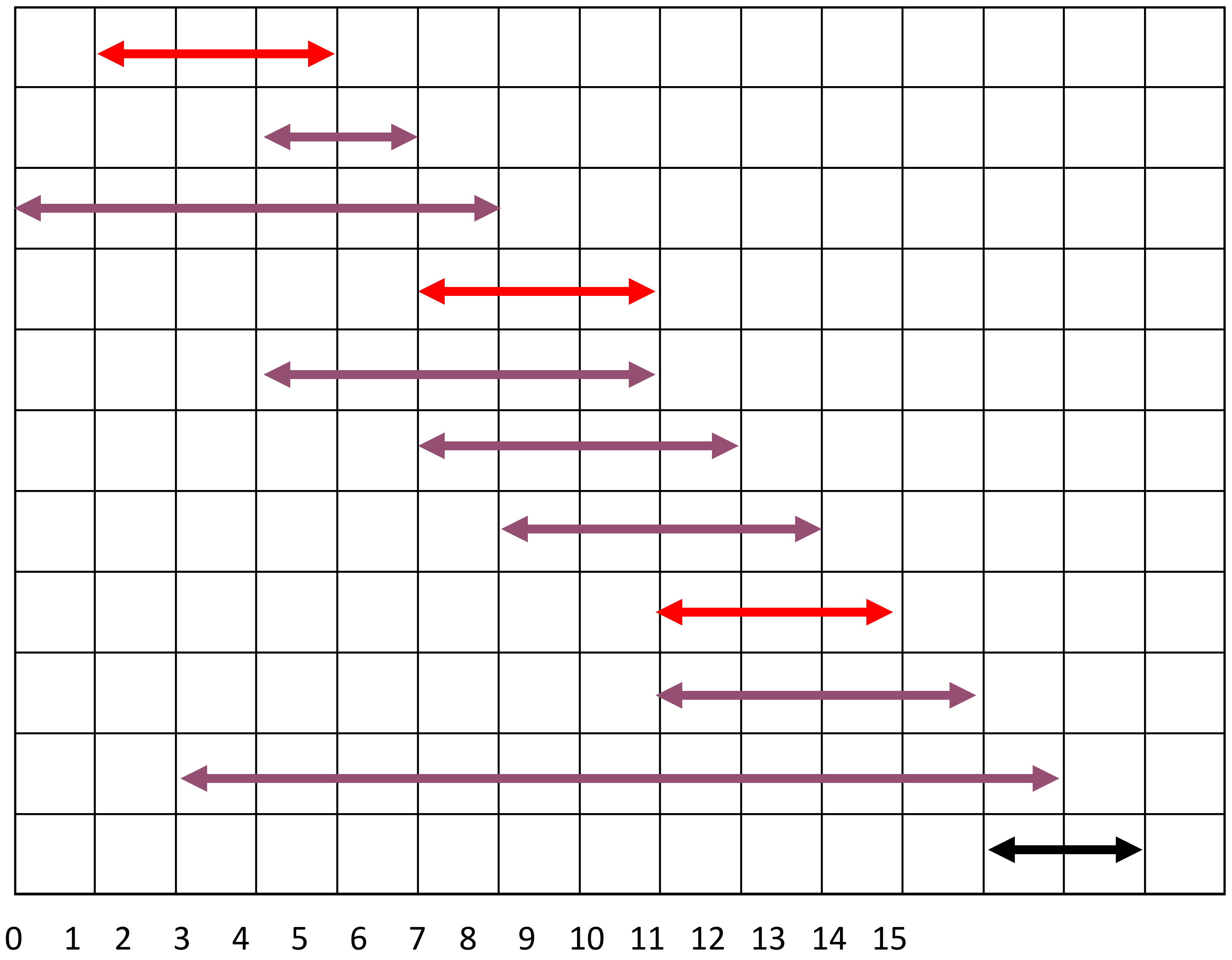

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

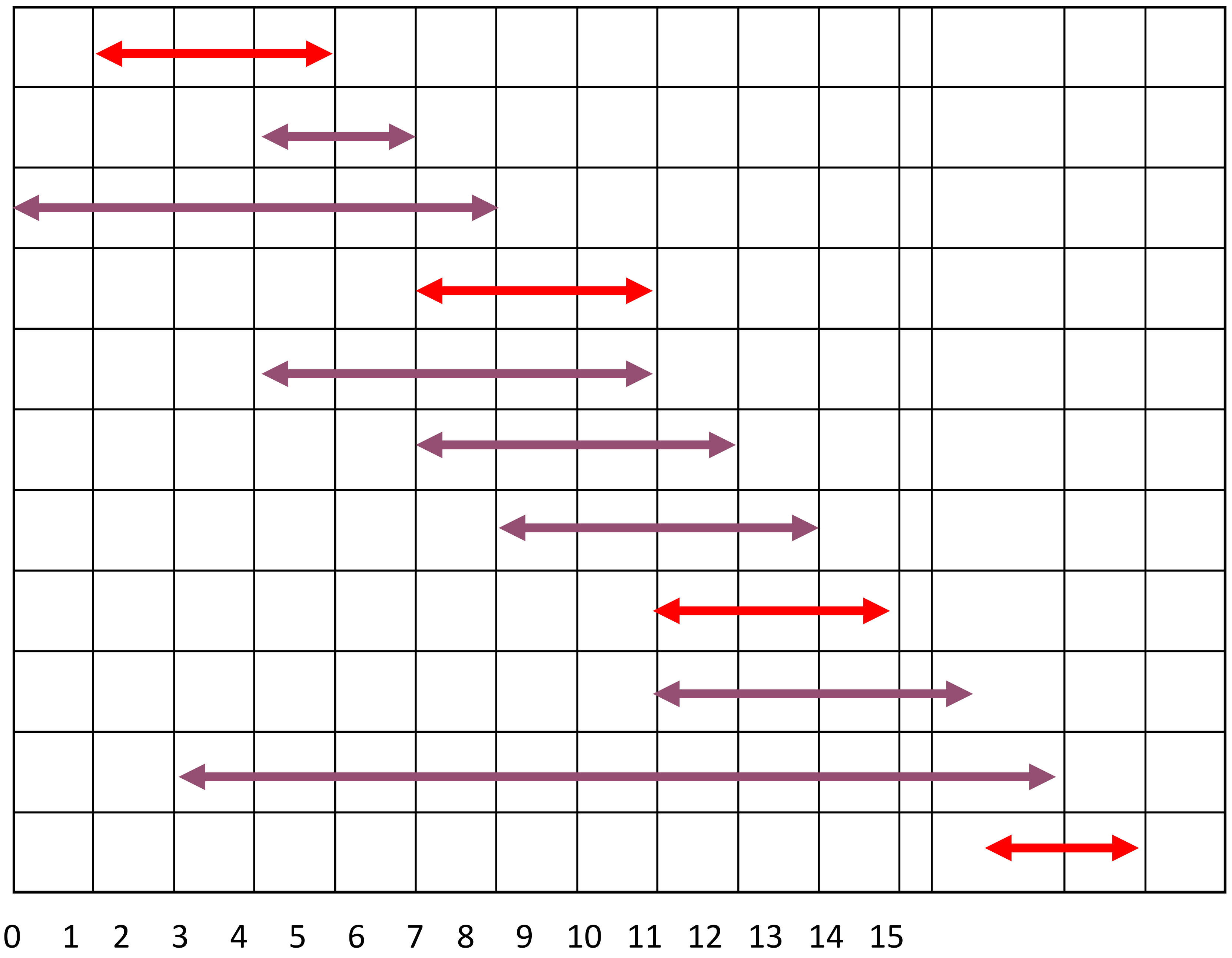

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

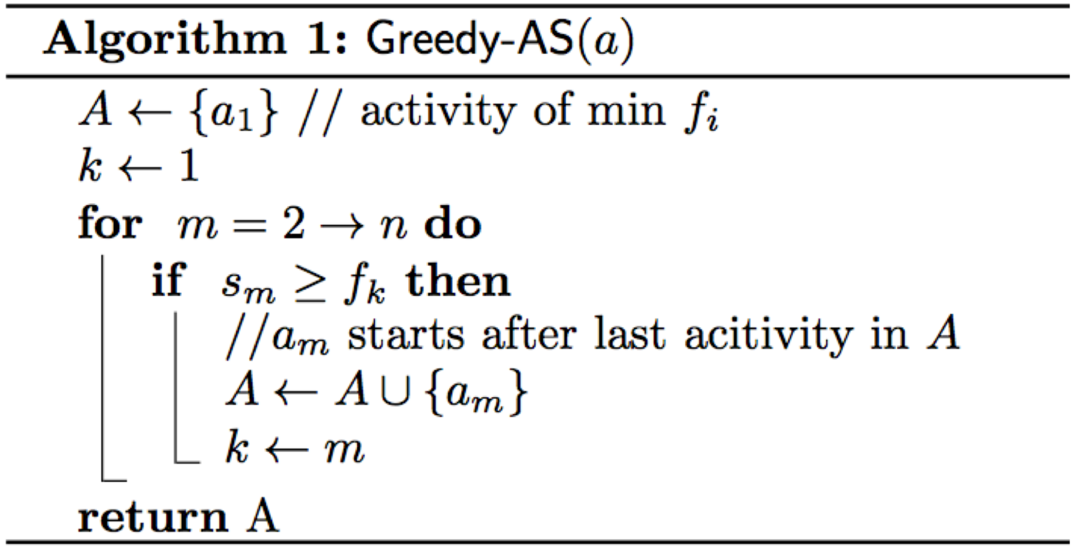

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Assuming activities are sorted by finish time



```
Algorithm 1: Greedy-AS($a$)
  $A \leftarrow \{a_1\}$ // activity of min $f_i$
  $k \leftarrow 1$
  for $m = 2 \rightarrow n$ do
      if $s_m \geq f_k$ then
          //$a_m$ starts after last acitivity in $A$
          $A \leftarrow A \cup \{a_m\}$
          $k \leftarrow m$
  return A
```

O(n log n)

Greedy-Choice Property

- Show there is an optimal solution that begins with a greedy choice (with activity 1, which as the earliest finish time)
- Suppose $A \subseteq S$ in an optimal solution
  - Order the activities in A by finish time. The first activity in A is k
    - If k = 1, the schedule A begins with a greedy choice
    - If $k \neq 1$, show that there is an optimal solution B to S that begins with the greedy choice, activity 1
  - Let $B = A - \{k\} \cup \{1\}$
    - $f_1 \leq f_k \rightarrow$ activities in B are disjoint (compatible)
    - B has the same number of activities as A
    - Thus, B is optimal

Optimal Substructures

- Once the greedy choice of activity 1 is made, the problem reduces to finding an optimal solution for the activity-selection problem over those activities in S that are compatible with activity 1
  - Optimal Substructure
  - If A is optimal to S, then $A' = A - \{1\}$ is optimal to $S'=\{i \in S: s_i \geq f_1\}$
  - Why?
    - If we could find a solution B' to S' with more activities than A', adding activity 1 to B' would yield a solution B to S with more activities than A ➔ contradicting the optimality of A
- After each greedy choice is made, we are left with an optimization problem of the same form as the original problem
  - By induction on the number of choices made, making the greedy choice at every step produces an optimal solution

Elements of Greedy Strategy

- An greedy algorithm makes a sequence of choices, each of the choices that seems best at the moment is chosen
  - NOT always produce an optimal solution

- Two ingredients that are exhibited by most problems that lend themselves to a greedy strategy
  - Greedy-choice property
  - Optimal substructure

0-1 Knapsack Problem

A thief robbing a store finds n items.

ith item: worth v_i dollars

w_i pounds

W, w_i, v_i are integers.

He can carry at most W pounds.

Fractional Knapsack Problem

A thief robbing a store finds n items.

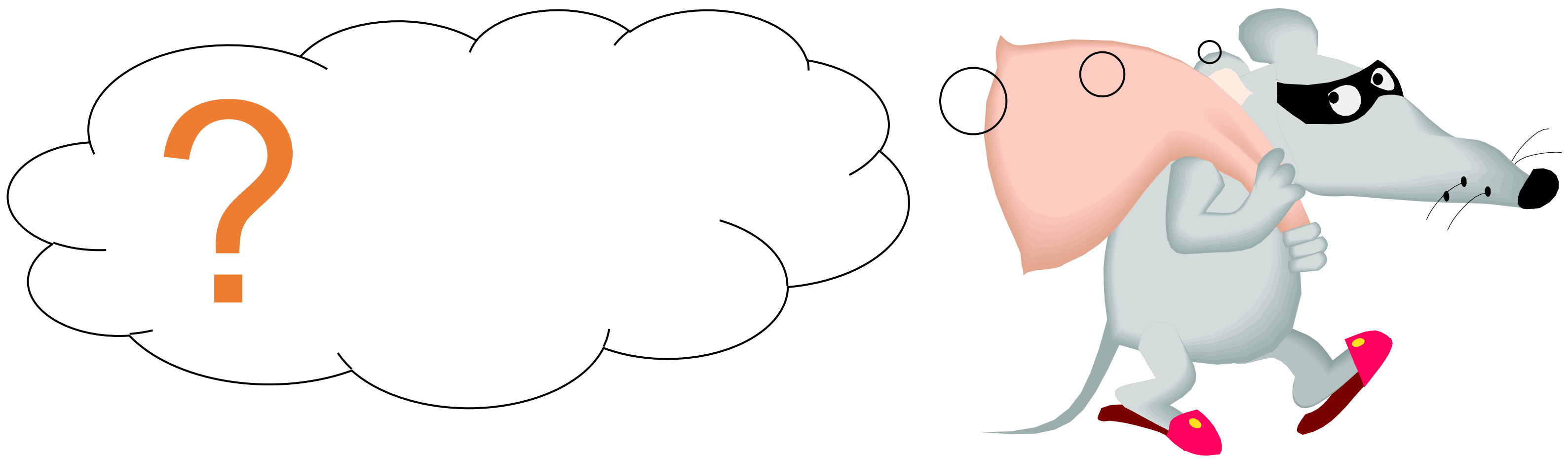
ith item: worth v_i dollars

w_i pounds

W, w_i, v_i are integers.

He can carry at most W pounds.

He can take fractions of items.



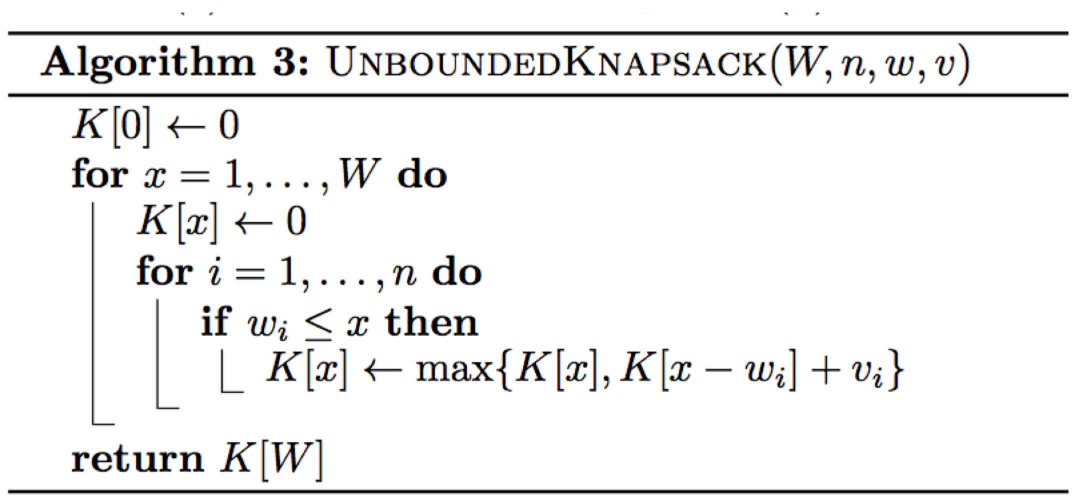
Knapsack Problems

- Both exhibit the optimal-substructure property
  - 0-1: If item *j* is removed from an optimal packing, the remaining packing is an optimal packing with weight at most W-w_j
  - Fractional: If *w pounds* of item *j* is removed from an optimal packing, the remaining packing is an optimal packing with weight at most W-w that can be taken from other *n-1* items plus w_j – w of item *j*

Greedy Algorithm for Fractional Knapsack

- Fractional knapsack can be solvable by the greedy strategy
  - Compute the value per pound v_i/w_i for each item
  - Obeying a greedy strategy, take as much as possible of the item with the greatest value per pound.
  - If the supply of that item is exhausted and there is still more room, take as much as possible of the item with the next value per pound, and so forth until there is no more room
  - O(n lg n) (we need to sort the items by value per pound)
  - Greedy Algorithm?
  - Correctness?

Greedy Algorithm for Fractional Knapsack



Algorithm 3: UNBOUNDEDKNAPSACK(W, n, w, v)

```
$K[0] \leftarrow 0$
for $x = 1, \ldots, W$ do
    $K[x] \leftarrow 0$
    for $i = 1, \ldots, n$ do
        if $w_i \leq x$ then
            $K[x] \leftarrow \max\{K[x], K[x - w_i] + v_i\}$
return $K[W]$
```

O-1 knapsack is harder!

- 0-1 knapsack cannot be solved by the greedy strategy
  - Unable to fill the knapsack to capacity, and the empty space lowers the effective value per pound of the packing
  - We must compare the solution to the sub-problem in which the item is included with the solution to the sub-problem in which the item is excluded before we can make the choice
  - Dynamic Programming

O-1 knapsack Algorithm

Algorithm 4: ZeroOneKnapsack(W, n, w, v)

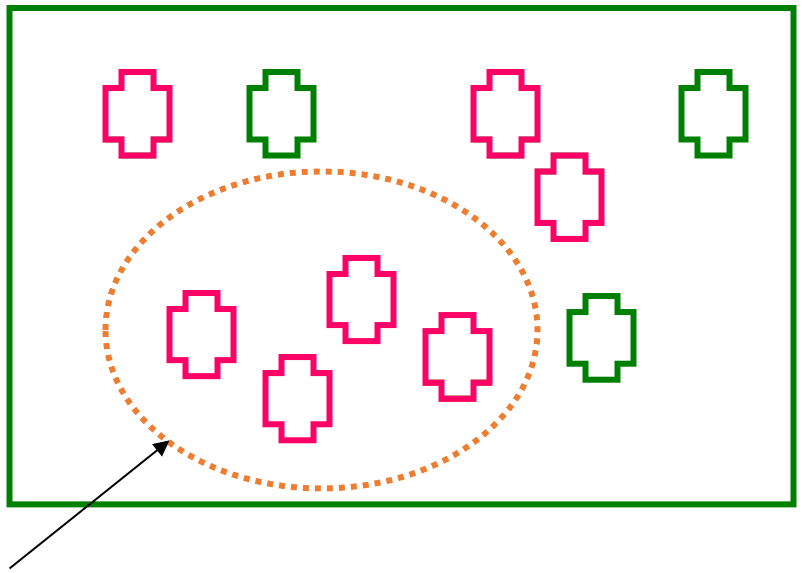
```
for $x = 1, \ldots, W$ do
    $K[x, 0] \leftarrow 0$
for $j = 1, \ldots, n$ do
    $K[0, j] \leftarrow 0$
for $j = 1, \ldots, n$ do
    for $x = 1, \ldots, W$ do
        $K[x, j] \leftarrow K[x, j-1]$
        if $w_j \leq x$ then
            $K[x, j] \leftarrow \max\{K[x, j], K[x - w_j, j-1] + v_j\}$
return $K[W, n]$
```

Greedy Algorithms

For the optimization problems:

Greedy Approach can solve these problems:

Dynamic Programming can solve these problems:



- Dynamic Programming may be "overkill" for some problems
- Greedy Strategy is simpler and more efficient.
- Greedy algorithms do not always give the optimal solution, but they frequently give good (approximate) solutions.

Remark

- Divide & Conquer Algorithms
  - Karatsuba’s multiplication, Insertion-sort, merge-sort
- Randomized Algorithms
  - Quick sort, Bloom’s filter, Chord algorithm
- Greedy Algorithms
  - Prim’s algorithm, Dijkstra’s algorithm
  - Fractional Knapsack
- Dynamic Programming
  - Bellman Ford algorithms
  - 0-1 Knapsack