Design and Analysis of Algorithm

Lecture-9:
Greedy Algorithm

Activities Selection

Assume activities are sorted by finish time

GREEDY-ACTIVITY-SELECTOR(s, f)

$n \leftarrow length[s]$
$A \leftarrow \{a_1\}$
$i \leftarrow 1$
for $m \leftarrow 2$ **to** n
    do if $s_m \geq f_i$
        then $A \leftarrow A \cup \{a_m\}$
            $i \leftarrow m$
return A

Example

There are 6 activities with corresponding start and end time, the objective is to compute an execution schedule having maximum number of non-conflicting activities

Start Time (s)	Finish Time (f)	Activity Name
5	9	a1
1	2	a2
3	4	a3
0	6	a4
5	7	a5
8	9	a6

Example

Step 1: Sort the given activities in ascending order according to their finishing time.

Start Time (s)	Finish Time (f)	Activity Name
1	2	a2
3	4	a3
0	6	a4
5	7	a5
5	9	a1
8	9	a6

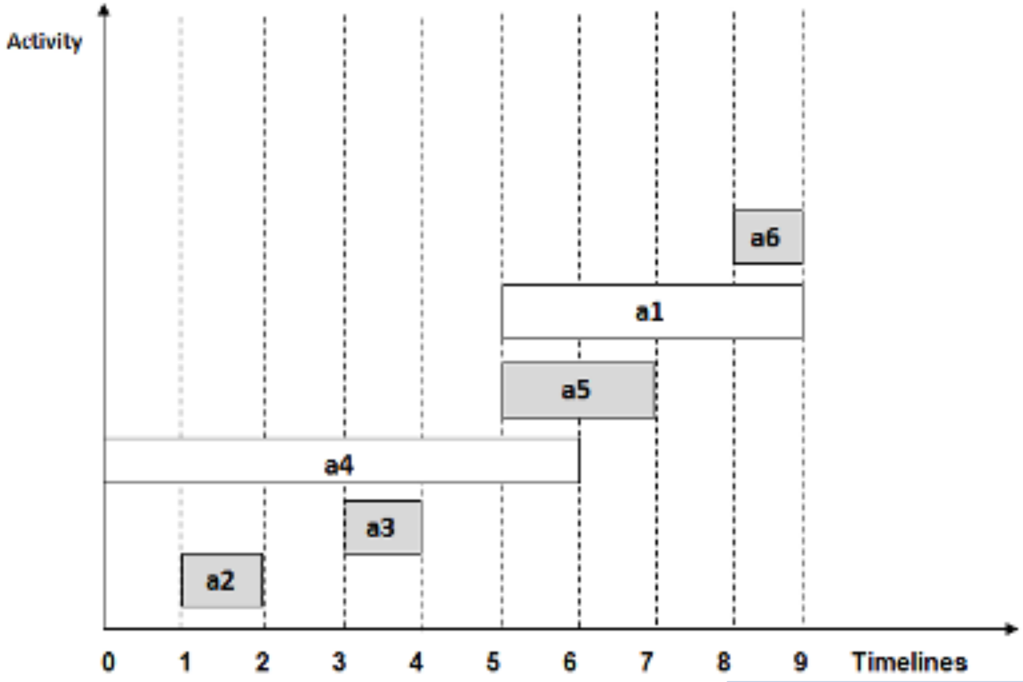
Example

Step 2: Select the first activity from sorted array act[] and add it to the sol[] array, thus **sol = {a2}**.

Step 3: Repeat the steps 4 and 5 for the remaining activities in act[].

Step4: If the start time of the currently selected activity is greater than or equal to the finish time of the previously selected activity, then add it to sol[].

Step 5: Select the next activity in act[]



Example

For the data given in the above table,

1. Select activity **a3**. Since the start time of **a3** is greater than the finish time of **a2** (i.e. s(a3) > f(a2)), we add **a3** to the solution set.

   Thus ***sol = {a2, a3}.***

2. Select **a4**. Since s(a4) < f(a3), it is not added to the solution set.
3. Select **a5**. Since s(a5) > f(a3), **a5** gets added to solution set.

   Thus **sol = {a2, a3, a5}**

4. Select **a1**. Since s(a1) < f(a5), **a1** is not added to the solution set.
5. Select **a6**. **a6** is added to the solution set since s(a6) > f(a5).

   Thus **sol = {a2, a3, a5, a6}.**

Step 6: At last, print the array `sol[]`

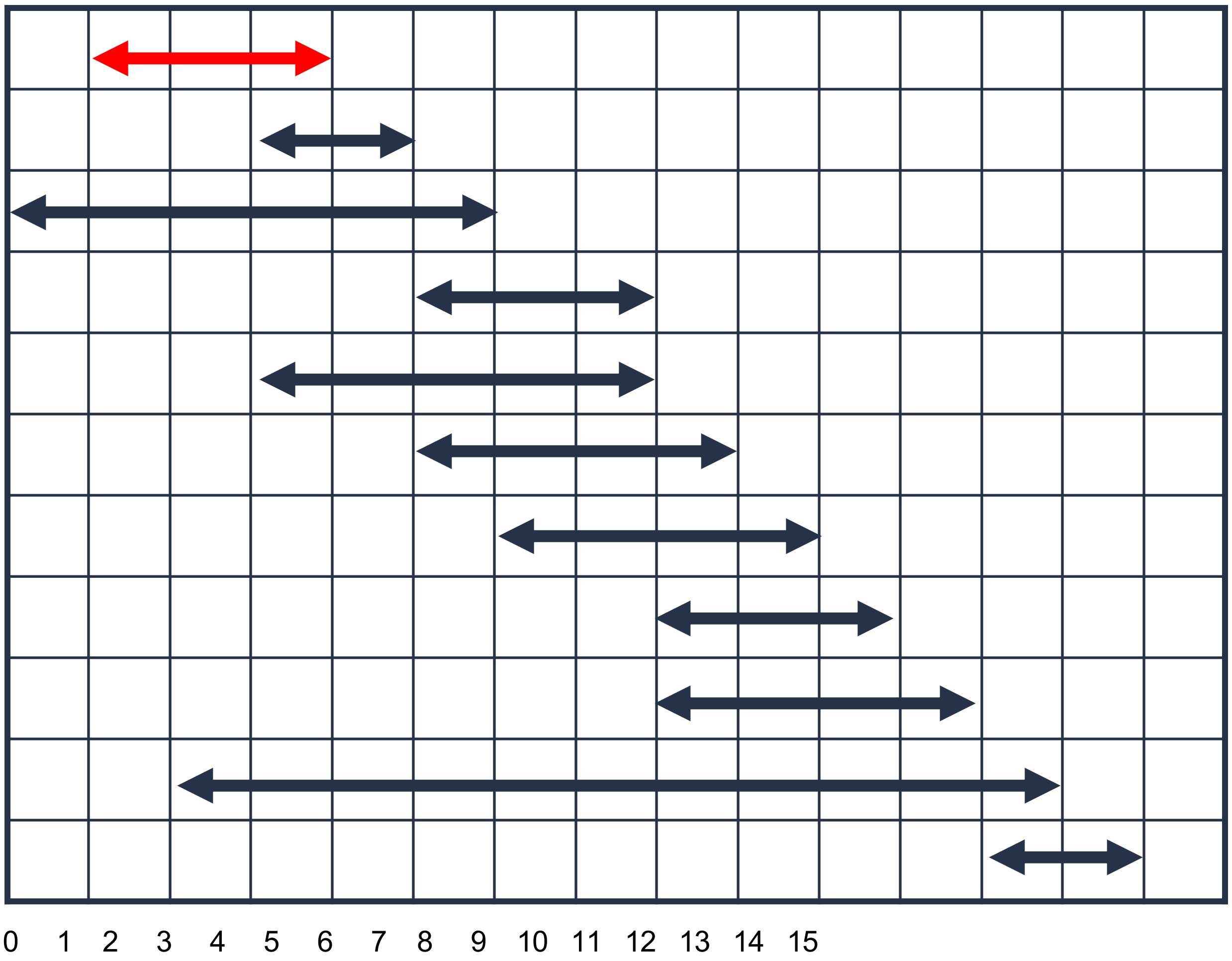
Hence, the execution schedule of maximum number of non-conflicting activities will be:

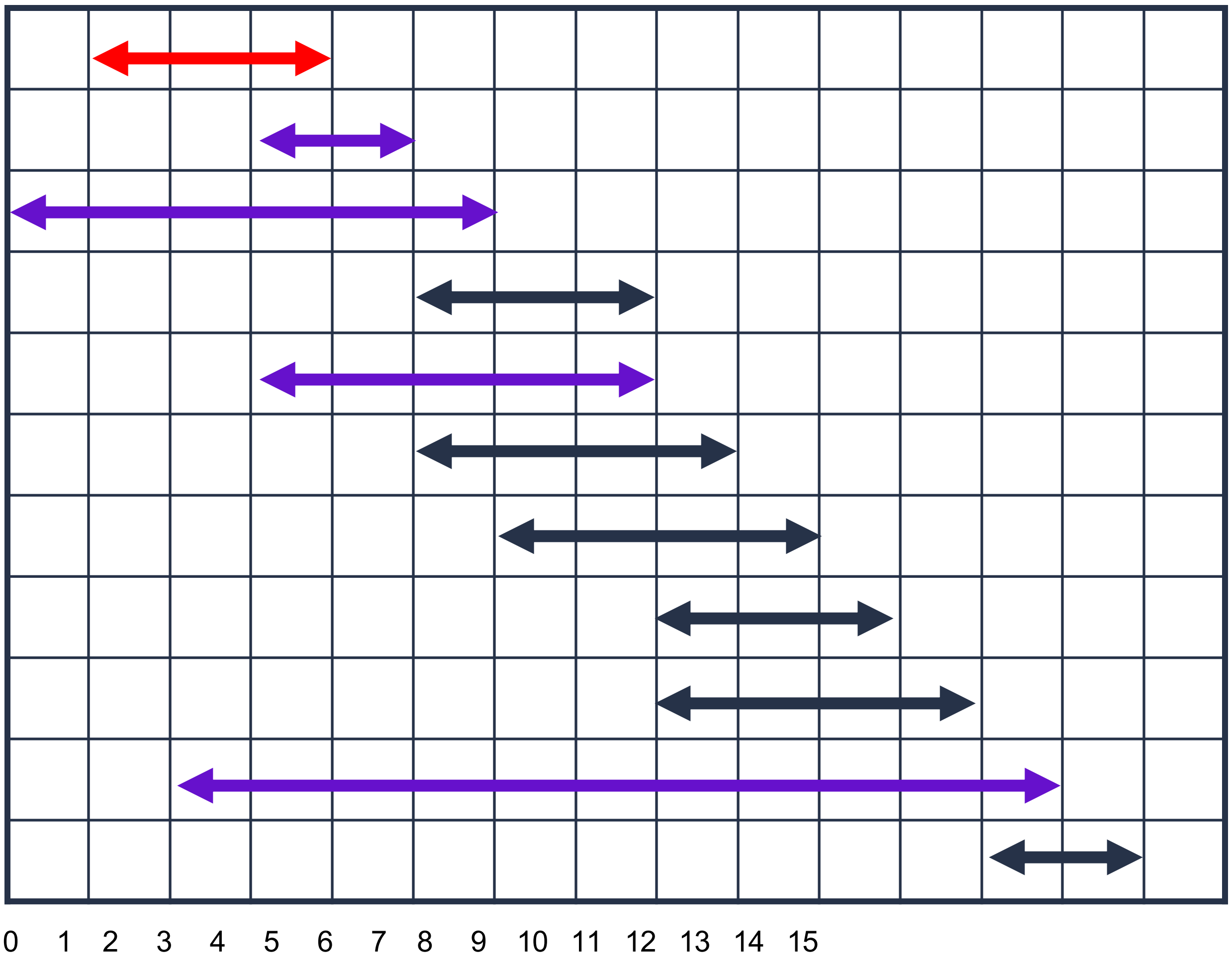
`(1,2)`

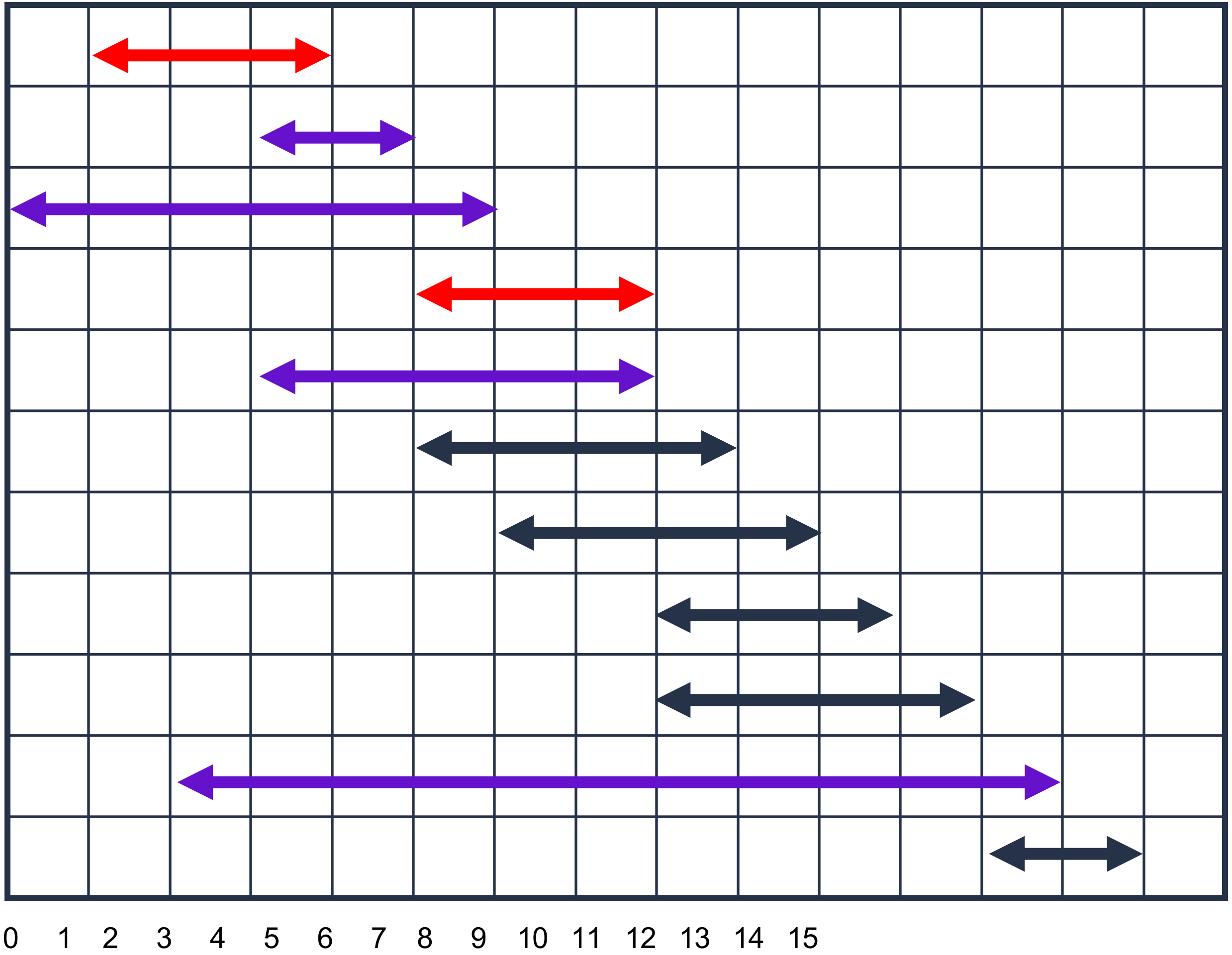
`(3,4)`

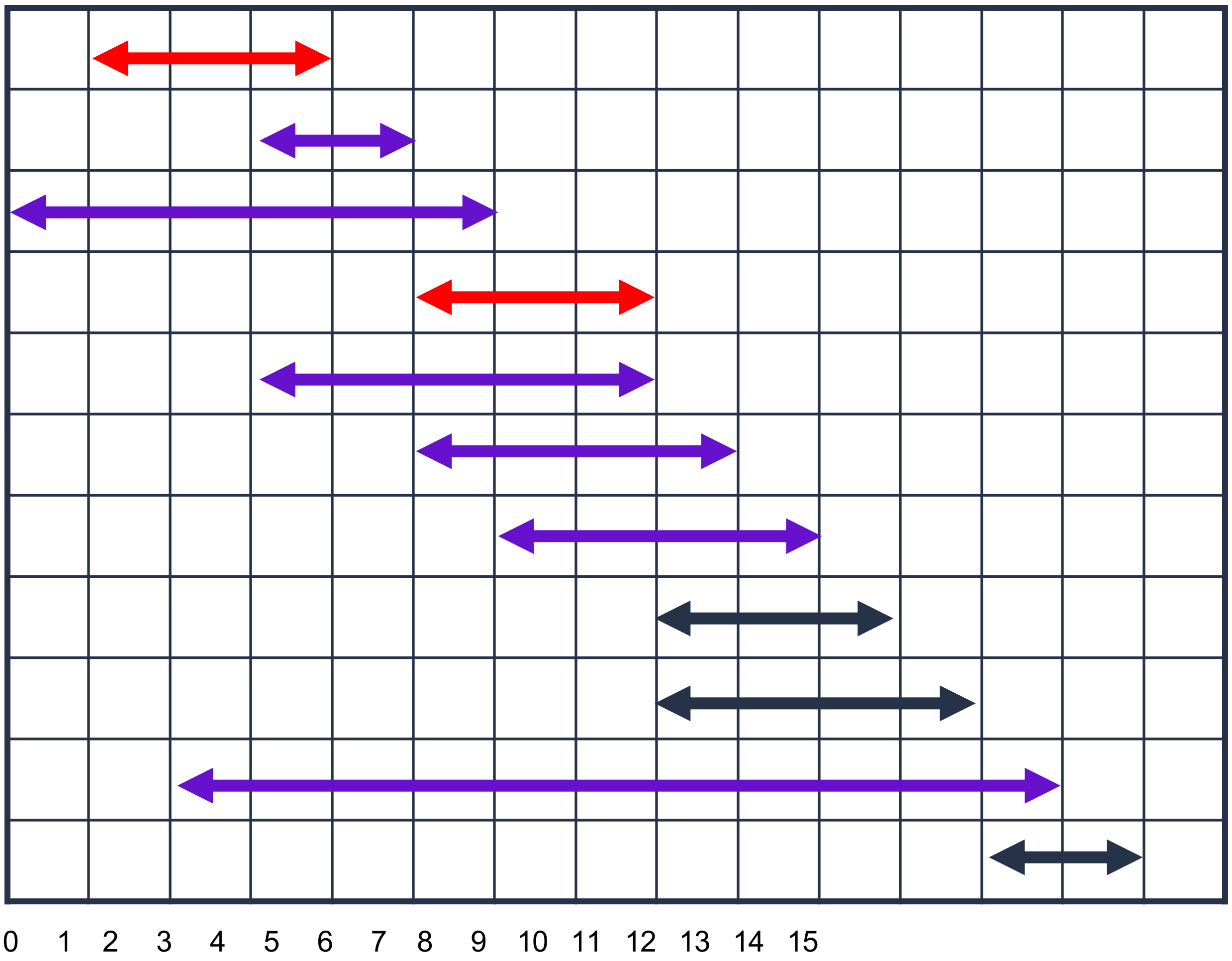
`(5,7)`

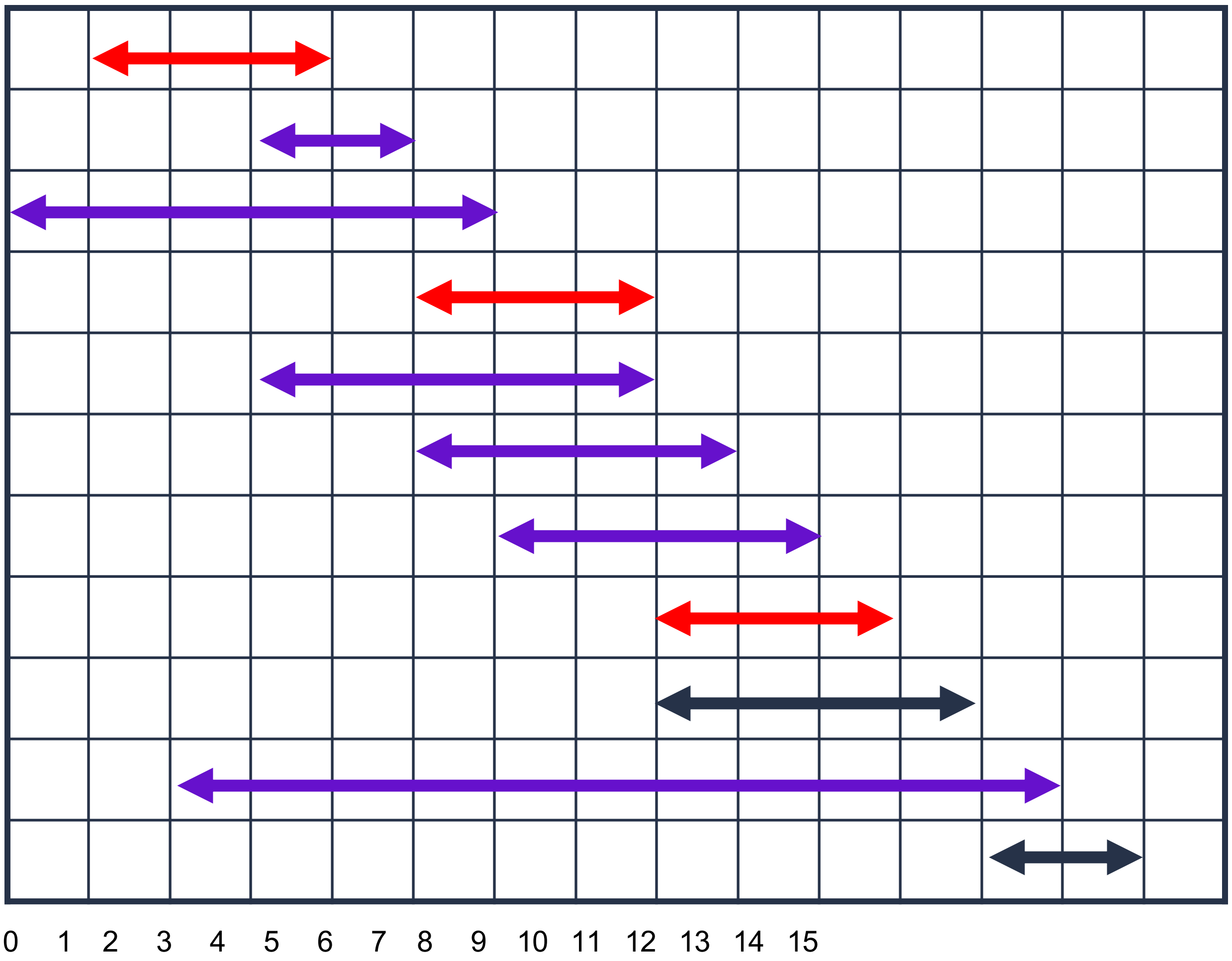
`(8,9)`

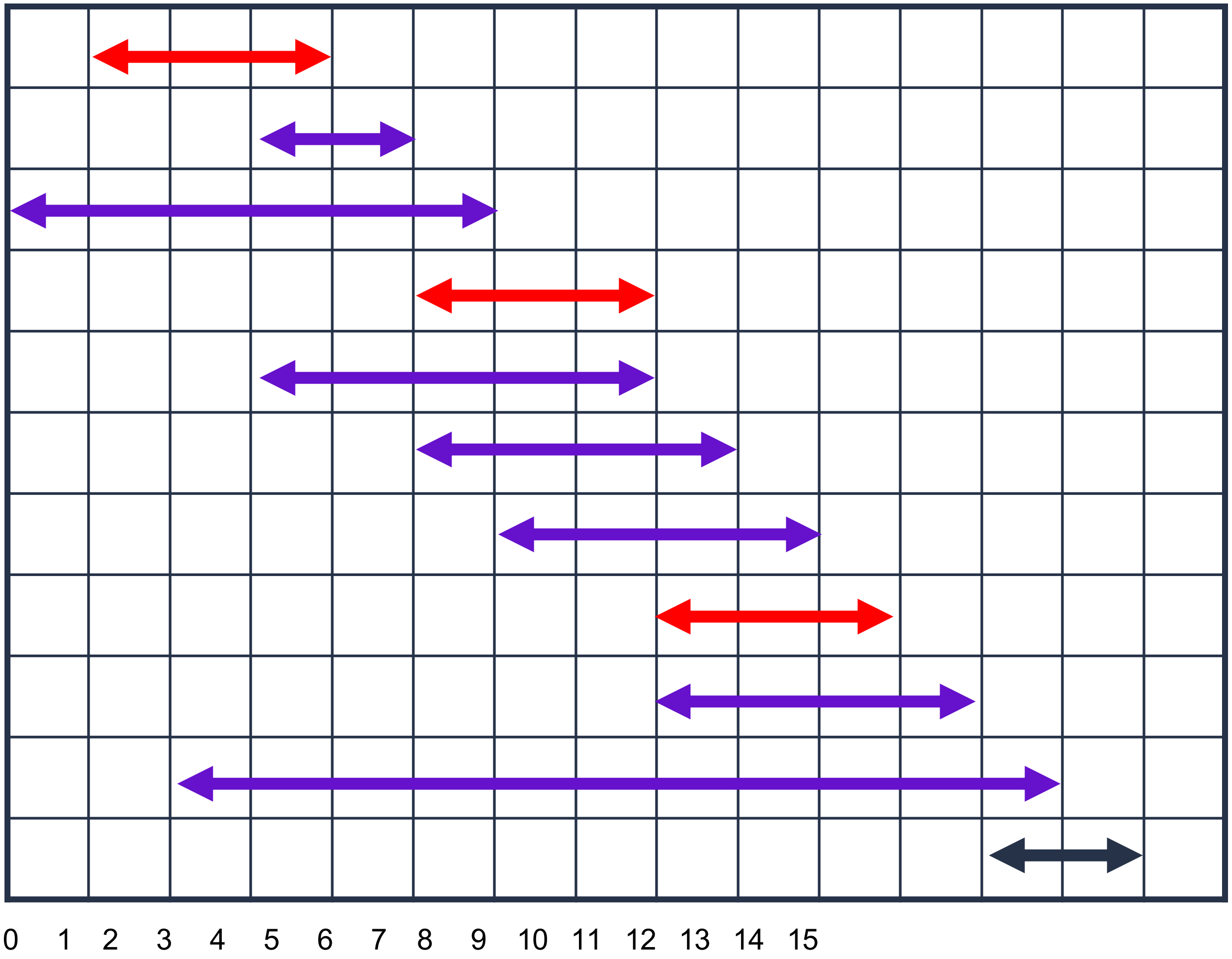

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

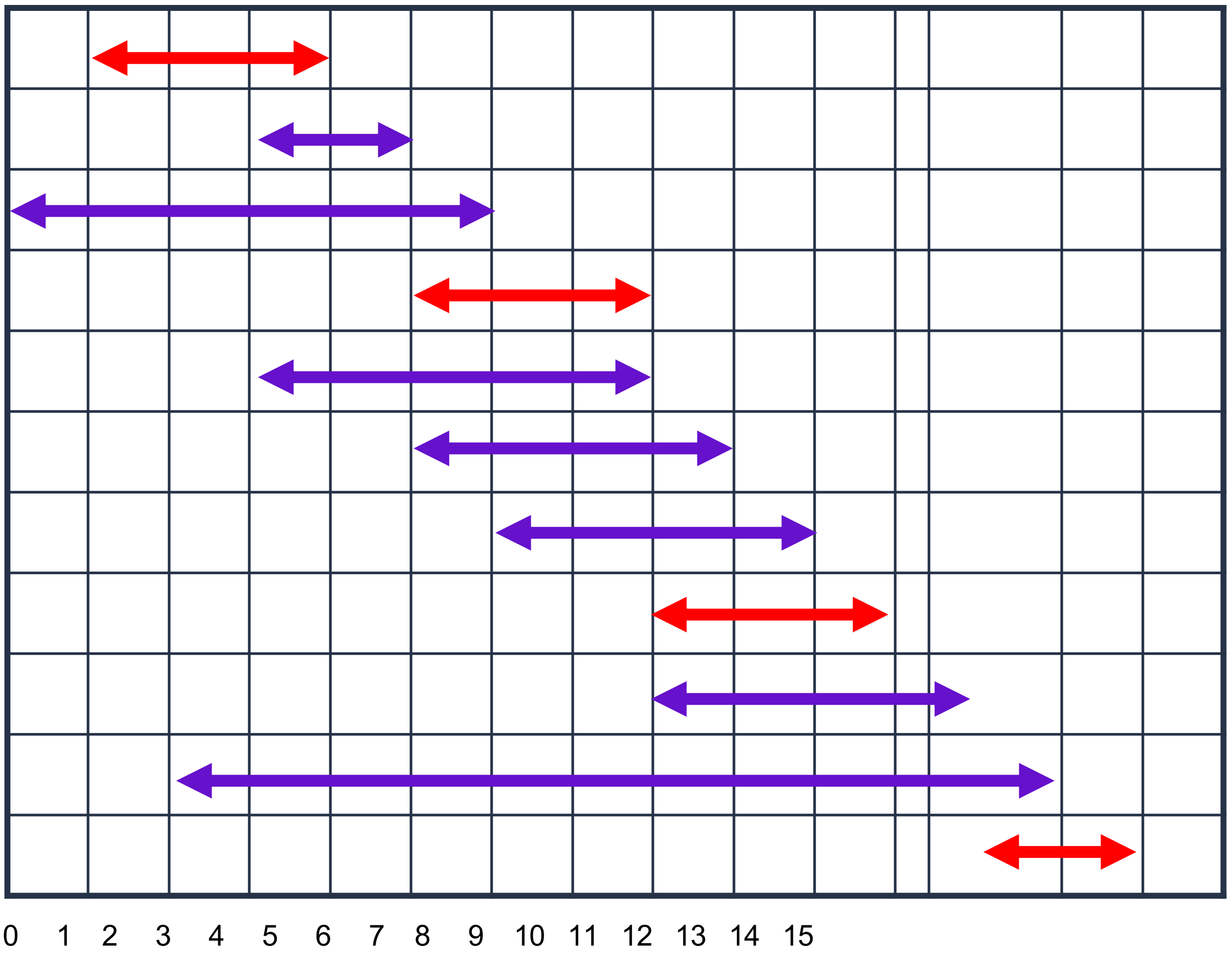

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15


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

Algorithm

```
Algorithm JS(d, j, n)
// d[i] ≥ 1, 1 ≤ i ≤ n are the deadlines, n ≥ 1. The jobs
// are ordered such that p[1] ≥ p[2] ≥ ··· ≥ p[n]. J[i]
// is the ith job in the optimal solution, 1 ≤ i ≤ k.
// Also, at termination d[J[i]] ≤ d[J[i + 1]], 1 ≤ i < k.
{
    d[0] := J[0] := 0; // Initialize.
    J[1] := 1; // Include job 1.
    k := 1;
    for i := 2 to n do
    {
        // Consider jobs in nonincreasing order of p[i]. Find
        // position for i and check feasibility of insertion.
        r := k;
        while ((d[J[r]] > d[i]) and (d[J[r]] ≠ r)) do r := r - 1;
        if ((d[J[r]] ≤ d[i]) and (d[i] > r)) then
        {
            // Insert i into J[ ].
            for q := k to (r + 1) step -1 do J[q + 1] := J[q];
            J[r + 1] := i; k := k + 1;
        }
    }
    return k;
}
```

Given that

$(p_1, p_2, p_3, p_4) = (100,10,15,27)$

$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

Sort jobs based on profit

$(p_1, p_4, p_3, p_2) = (100,27,15,10)$

$(d_1, d_4, d_3, d_2) = (2, 1, 2, 1)$

Initialize

J_0	J_1	J_2	J_3	J_4
0	1			

Complexity of the algorithm

For the given algorithm there are two parameters in terms of which its complexity can be measured.

1. *Total number of jobs (say n)*
2. *Number of jobs selected (say s)*

There are two loops (1) For loop (2) While loop

For Loop

It will run maximum of n-1 times, So complexity is o(n)

While Loop

In any iteration it will run for k times

Maximum possible value of k is s

The complexity of the given algorithm will thus be O(ns)

Theorems

Theorem Let J be a set of k jobs and $\sigma = i_1, i_2, \ldots, i_k$ a permutation of jobs in J such that $d_{i_1} \leq d_{i_2} \leq \cdots \leq d_{i_k}$. Then J is a feasible solution iff the jobs in J can be processed in the order σ without violating any deadline.

Theorem The greedy method described above always obtains an optimal solution to the job sequencing problem.