Activity Selection Example

An Activity Selection Problem (Conference Scheduling Problem)

- **Input: A set of activities S = {a_1,…, a_n}**
- Each activity has start time and a finish time
  - a_i=(s_i, f_i)
- 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

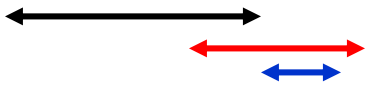
- Here are a set of start and finish times

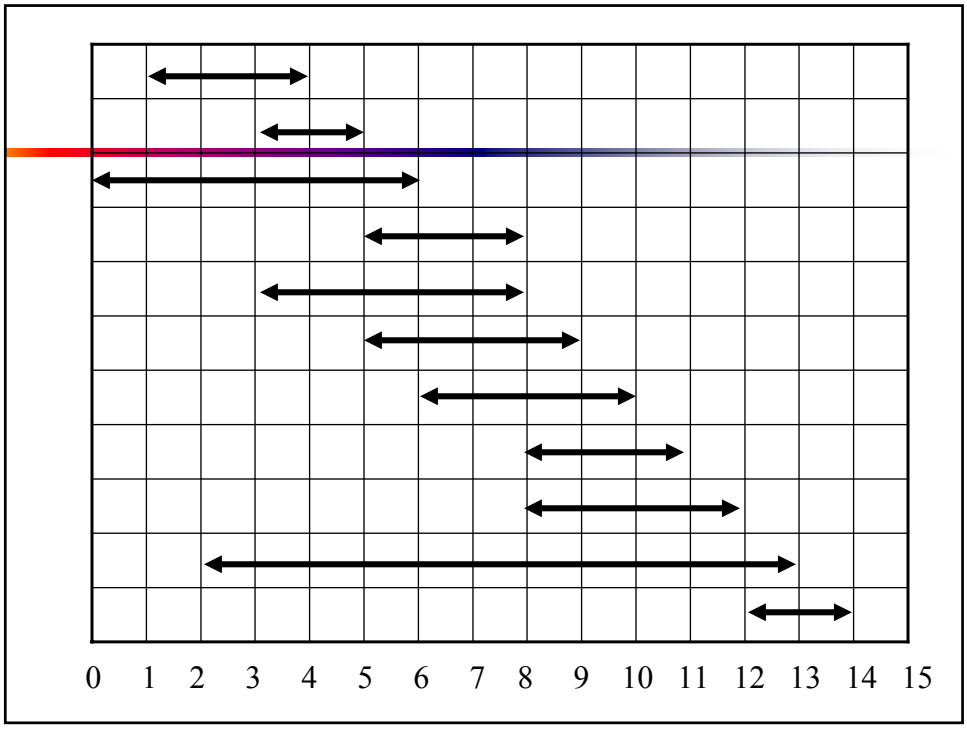
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

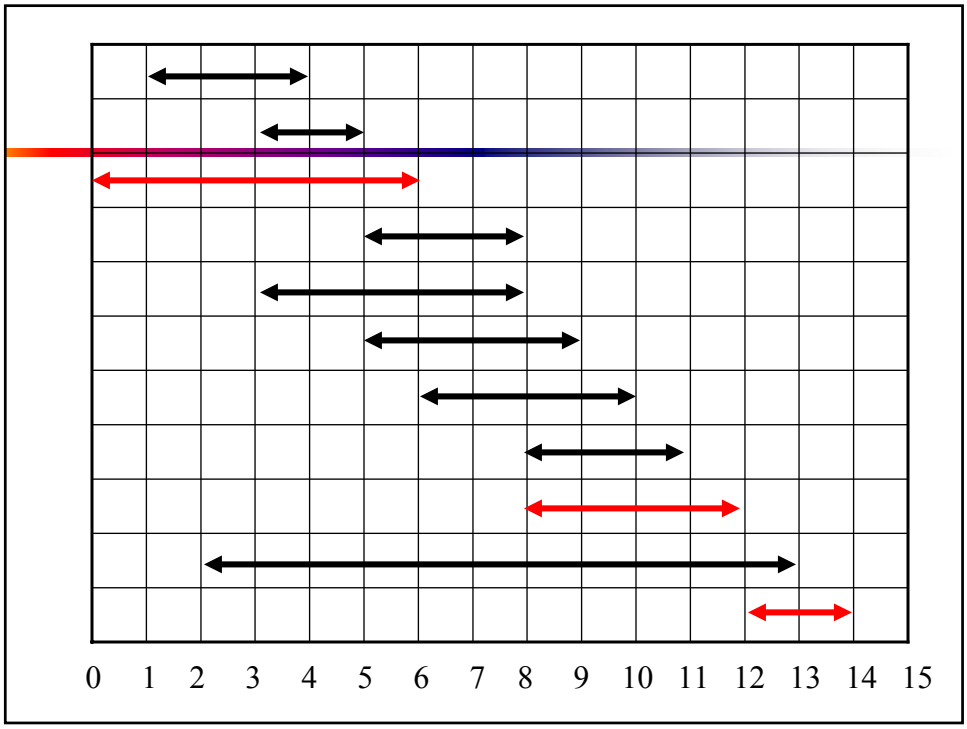
- What is the maximum number of activities that can be completed?
  - $\{a_3, a_9, a_{11}\}$ can be completed
  - But so can $\{a_1, a_4, a_8, a_{11}\}$ which is a larger set
  - But it is not unique, consider $\{a_2, a_4, a_9, a_{11}\}$

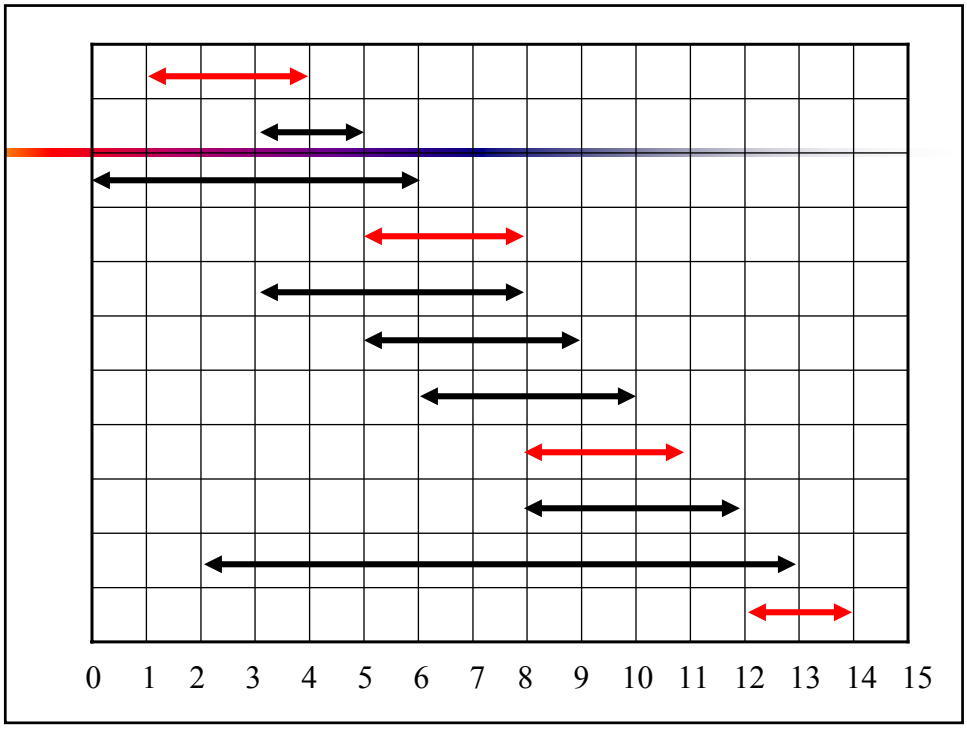
Interval Representation

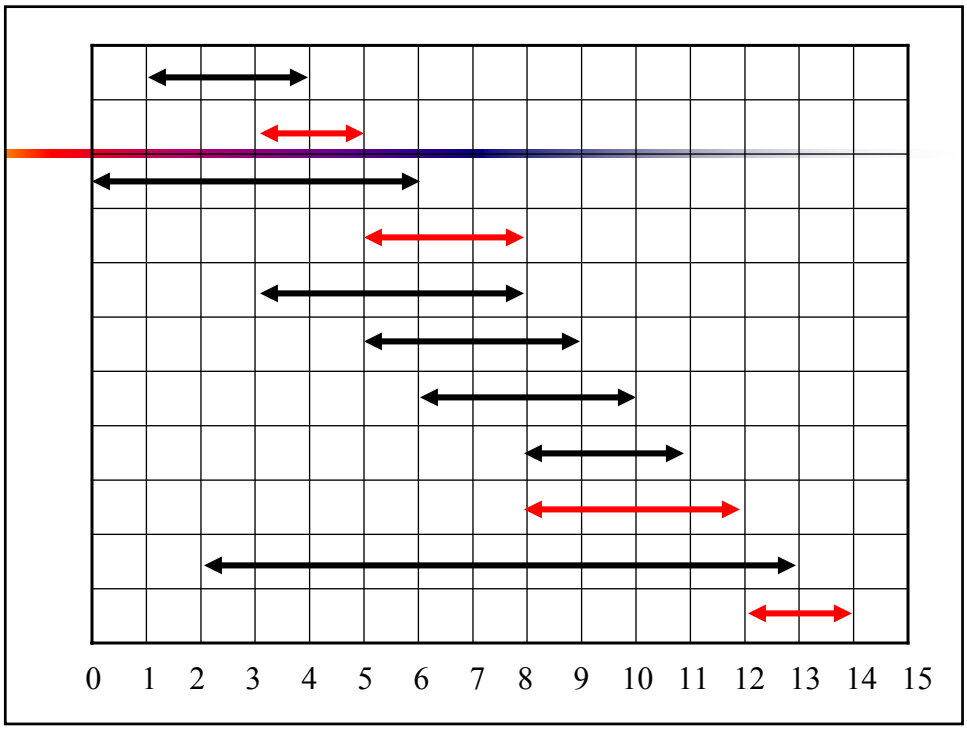
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14




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

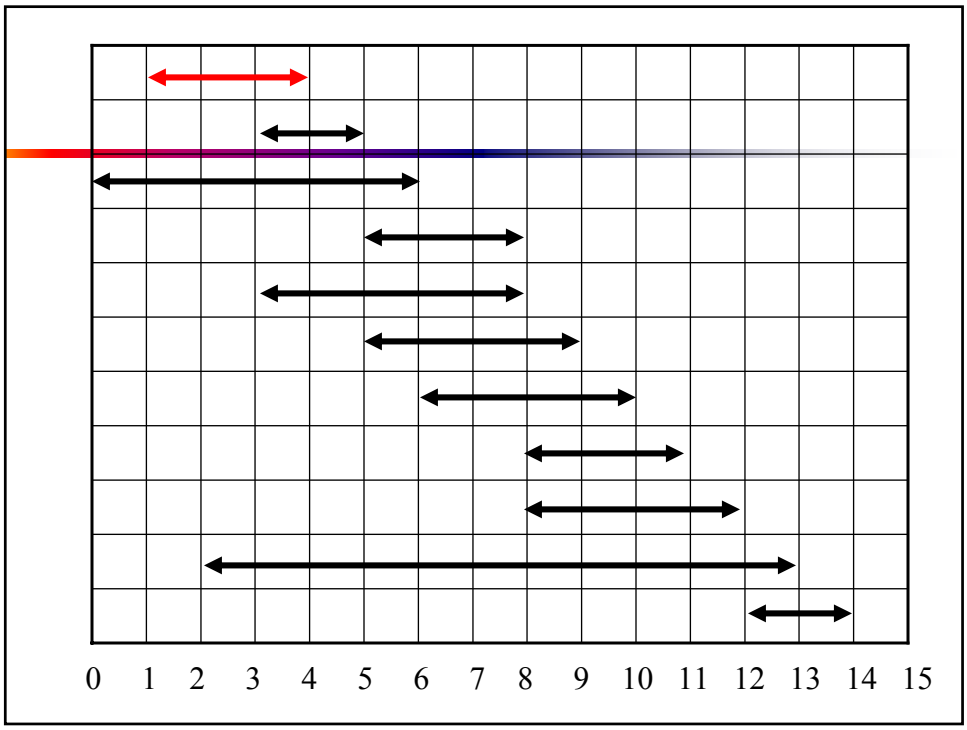

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

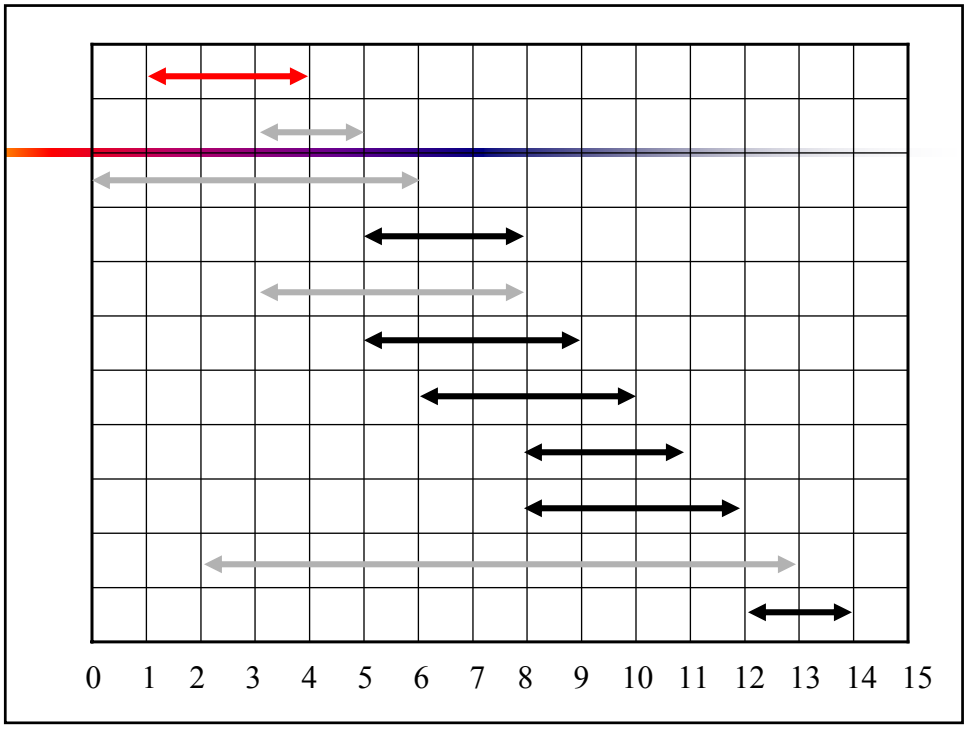

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

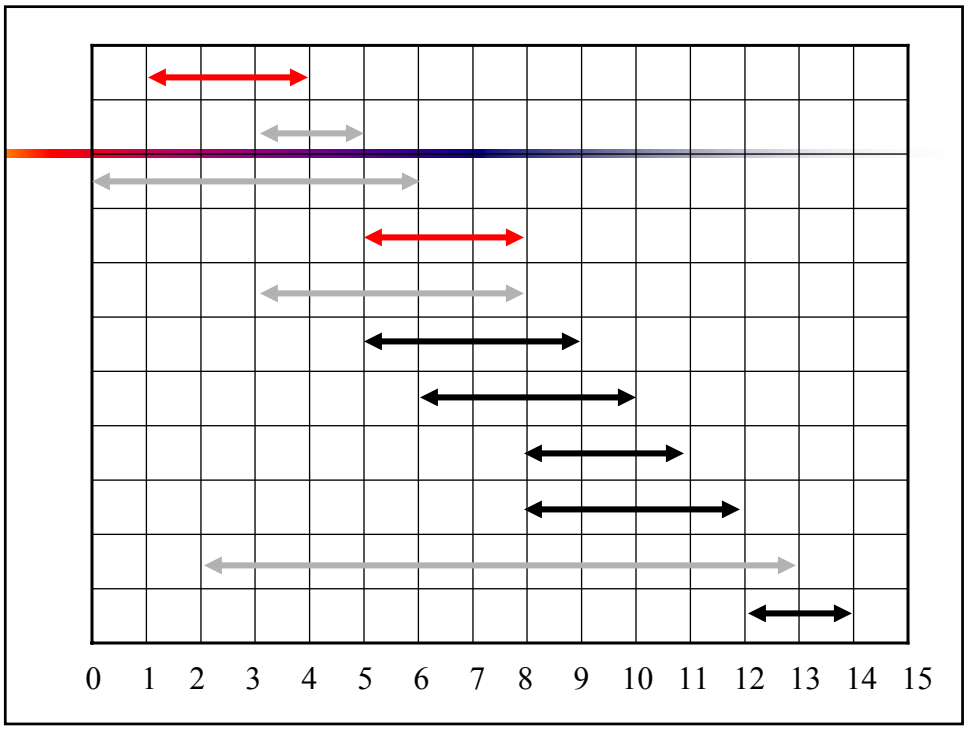

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

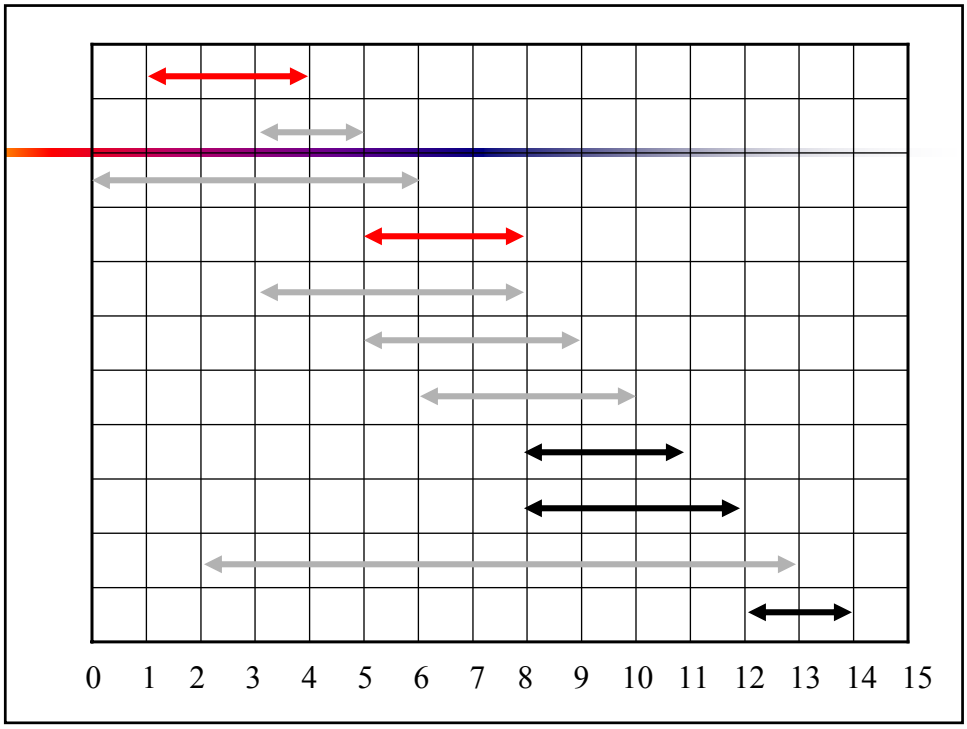
Early Finish Greedy

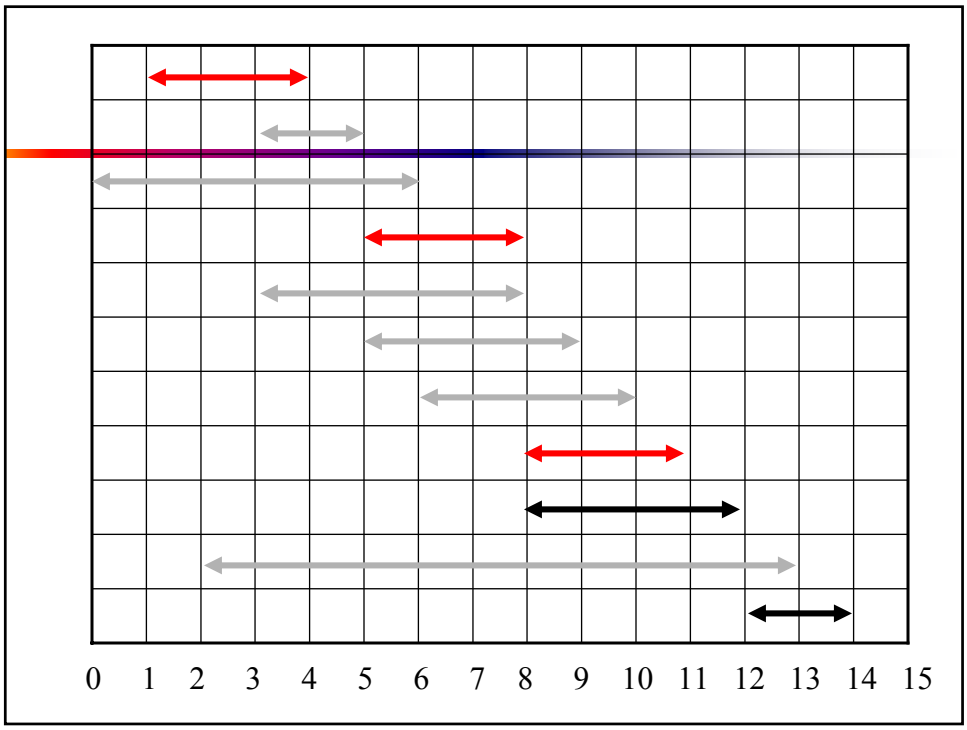
- Select the activity with the earliest finish
- Eliminate the activities that could not be scheduled
- Repeat!

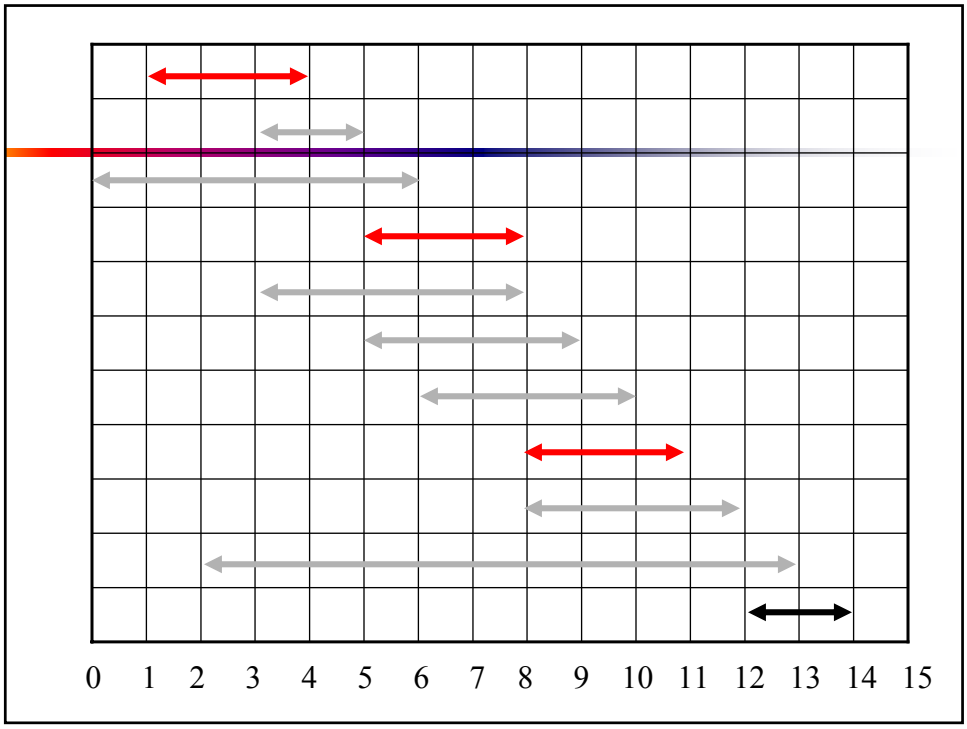


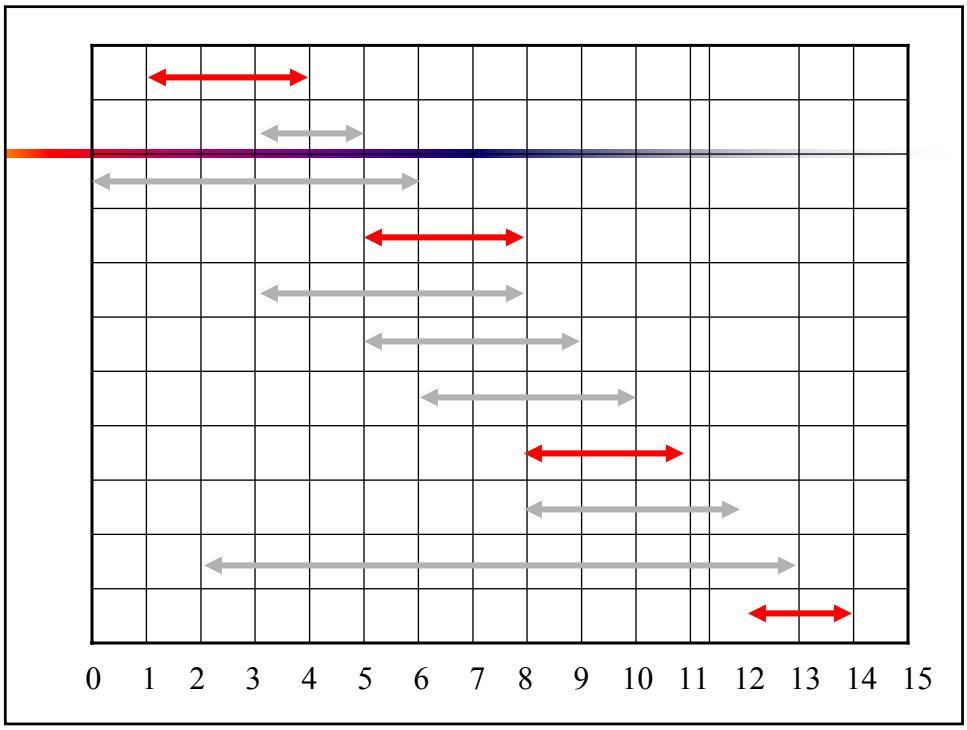

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15


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

Assuming activities are sorted by finish time

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n ← length[s]
2  A ← {a_1}
3  i ← 1
4  for m ← 2 to n
5        do if s_m ≥ f_i
6              then A ← A ∪ {a_m}
7                   i ← m
8  return A
```

Why it is Greedy?

- Greedy in the sense that it leaves as much opportunity as possible for the remaining activities to be scheduled
- The greedy choice is the one that maximizes the amount of unscheduled time remaining

Why this Algorithm is Optimal?

- We will show that this algorithm uses the following properties
  - The problem has the optimal substructure property
  - The algorithm satisfies the greedy-choice property
- Thus, it is Optimal

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}
    - $f_1 \leq f_k$ → 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

Greedy-Choice Property

- A globally optimal solution can be arrived at by making a locally optimal (greedy) choice
  - Make whatever choice seems best at the moment and then solve the sub-problem arising after the choice is made
  - The choice made by a greedy algorithm may depend on choices so far, but it cannot depend on any future choices or on the solutions to sub-problems
- Of course, we must prove that a greedy choice at each step yields a globally optimal solution

Optimal Substructures

- A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to sub-problems
  - If an optimal solution A to S begins with activity 1, then A' = A – {1} is optimal to S'={i $\in$S: $s_i \geq f_1$}

Optimal substructures

- Define the following subset of activities which are activities that can start after a_i finishes and finish before a_j starts

$$S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$$

- Sort the activities according to finish time

$$f_0 \leq f_1 \leq f_2 \leq \cdots \leq f_n < f_{n+1}$$

- We now define the the maximal set of activities from i to j as

$$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$$

- Let c[i,j] be the maximal number of activities

$$c[i, j] = c[i, k] + c[k, j] + 1$$

- Our recurrence relation for finding c[i, j] becomes

$$c[i, j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max\limits_{i<k<j} \{c[i, k] + c[k, j] + 1\} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

- We can solve this using dynamic programming, but a simpler approach exists

The End