CSE- 207
Algorithms

Lecture: 09
Greedy Algorithms

Fahad Ahmed

Lecturer, Dept. of CSE

E-mail: fahadahmed@uap-bd.edu

Greedy Algorithms

Optimization Problems

- **Optimization Problem:**
  - An optimization problem is **one in which we are given a set of input values, which are required either to be maximized or minimized** (known as objective), i.e. some **constraints or conditions**.

How do you decide which choice is optimal?

- For any optimization there are 2 key things.
  - **An objective function**
    - Normally maximize or minimize something
  - **A set of constraints**
    - What resources and limitation we have.

Solving Optimization Problems

- **Three techniques for solving optimization problems:**
  - Greedy Algorithms ("Greedy Strategy")
  - Dynamic Programming
  - Branch and Bound

We still care about Greedy Algorithms because for some problems:

- **Dynamic programming is overkill (slow)**
- **Greedy algorithm is simpler and more efficient**

Optimization Problems

- **Optimization Problem:**
  - the problem of finding the **best solution** from all **feasible solutions**

- **Optimization problems appear in so many applications**
  - *Maximize* the number of jobs using a resource **[Activity-Selection Problem]**
  - Encode the data in a file to *minimize* its size [**Huffman Encoding Problem**]
  - Collect the *maximum* value of goods that fit in a given bucket [**knapsack Problem**]
  - Select the *smallest-weight* of edges to connect all nodes in a graph **[Minimum Spanning Tree]**

GREEDY ALGORITHM: MAIN CONCEPT

- A greedy algorithm is a mathematical process that
  - looks for simple, easy-to-implement solutions to complex, **multi-step** problems
  - by deciding **which next step** will provide the **most** obvious **benefit**.

- Such algorithms are called greedy because
  - *it always makes the choice that looks best at* the moment. **(Local optimal) (Greedy choice)**
  - the algorithm doesn't consider the larger problem as a whole.
  - Once a decision has been made, it is never reconsidered.

Greedy algorithm

- A ***greedy*** *algorithm* makes a **locally optimal** choice **in the hope that** this choice will **lead to a globally optimal** solution.
- A *greedy algorithm* always makes the choice that looks best at the moment

Greedy algorithm

- A *greedy algorithm* always makes the choice that looks best at the moment
  - My everyday examples:
    - Select employee
    - Invest on stocks
    - Choose a university
    - Playing a bridge hand game

- Greedy algorithms do not always yield optimal solutions, but for **many problems they do**.

How do you decide which choice is optimal?

- Assume that you have an **objective function** that needs to be optimized (either maximized or minimized) at a given point.
- A Greedy algorithm makes greedy choices at each step to ensure that the objective function is optimized.
- The Greedy algorithm has only one shot to compute the optimal solution so that **it never goes back and reverses the decision**.

Steps of Greedy algorithm

- Make greedy choice at the beginning of each iteration
- Create sub problem
- Solve the sub problem
  - But, How?
  - Continue first two steps until the all the sub-problems are solved.

History of Greedy Algorithms

Here is an important landmark of greedy algorithms:

- Greedy algorithms were conceptualized for many graph walk algorithms in the 1950s.
- Esdger Djikstra conceptualized the algorithm to generate minimal spanning trees. He aimed to shorten the span of routes within the Dutch capital, Amsterdam.
- In the same decade, Prim and Kruskal achieved optimization strategies that were based on minimizing path costs along weighed routes.
- The Greedy search paradigm was registered as a different type of optimization strategy in the NIST records in 2005.
- Till date, protocols that run the web, such as the open-shortest-path-first (OSPF) and many other network packet switching protocols use the greedy strategy to minimize time spent on a network.

Activity selection problem

Activity Selection Problem

- The **activity selection problem** is a classic optimization problem concerning the selection of **non-conflicting** activities to perform within a given time frame.

- The problem is to select the **maximum number of activities** that can be **performed by a single** person or machine, assuming that a person can only **work on a single activity at a time**.

- A classic application of this problem is in **scheduling** a room for multiple competing events, each having its own time requirements (start and end time).

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**

Activity Selection Problem- Example

- Suppose we have a set of activities $\{a_1; a_2; \ldots ; a_n\}$ ***that wish to use a resource, such as a lecture hall, which*** can serve only one activity at a time.
- Each activity a_i has a ***start time s_i and a finish time f_i, where $0 \leq s_i < f_i < a$.***
- We have to select the **maximum-size subset** of activities that are mutually **compatible**.
- Two activities are **compatible if** their intervals **do not overlap**.

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?*

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

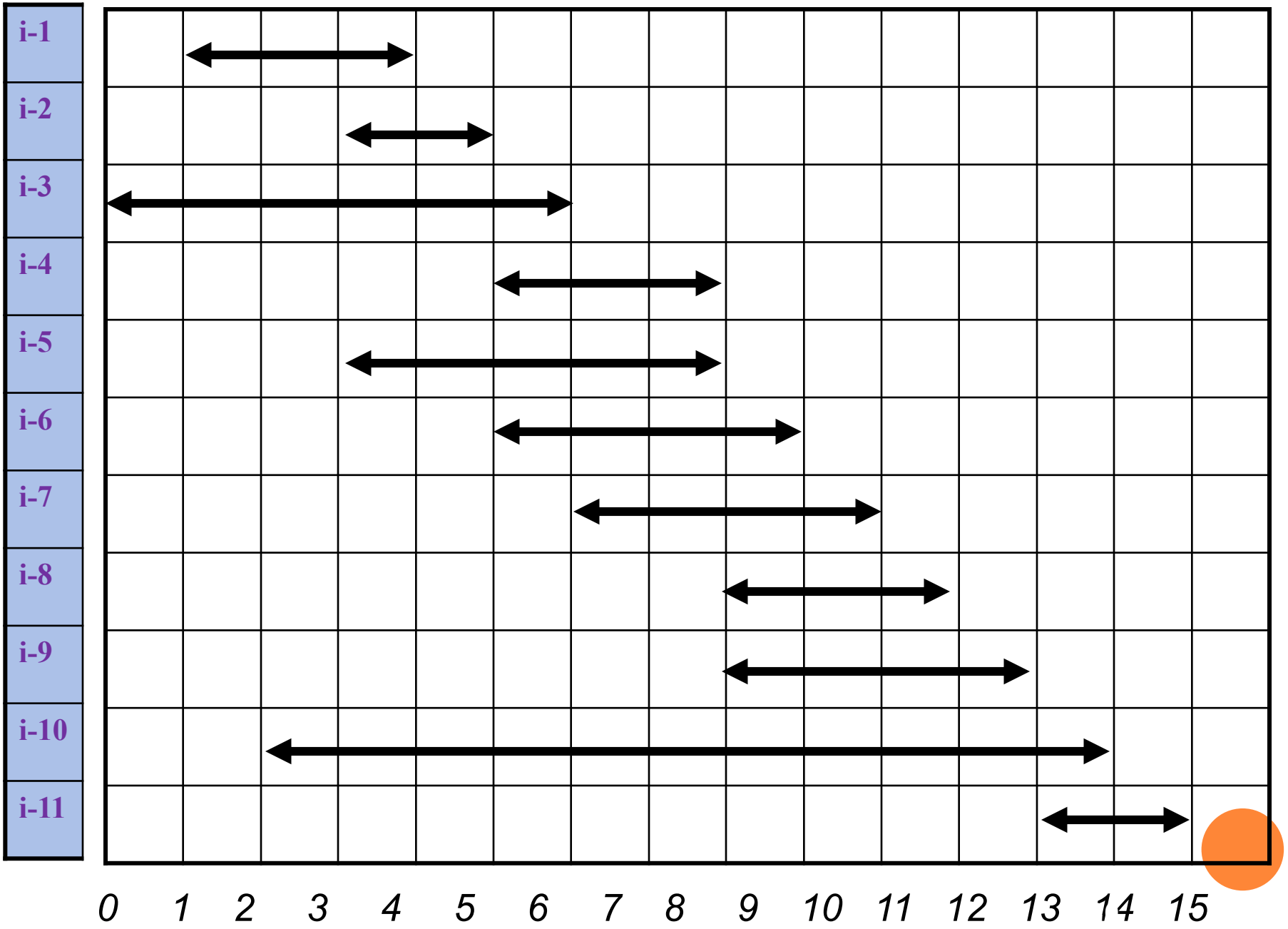
Not observed yet

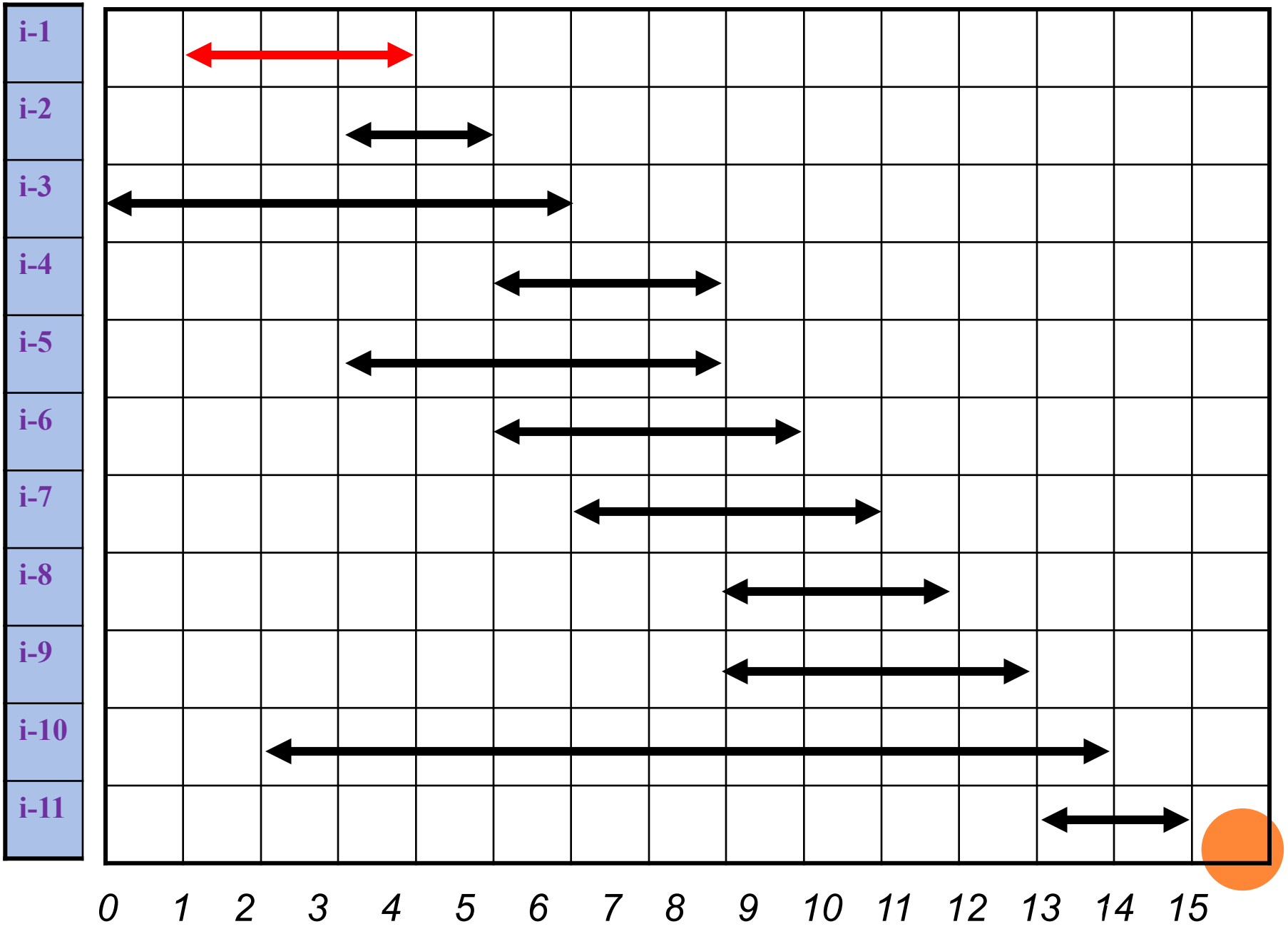
Added in optimal solution

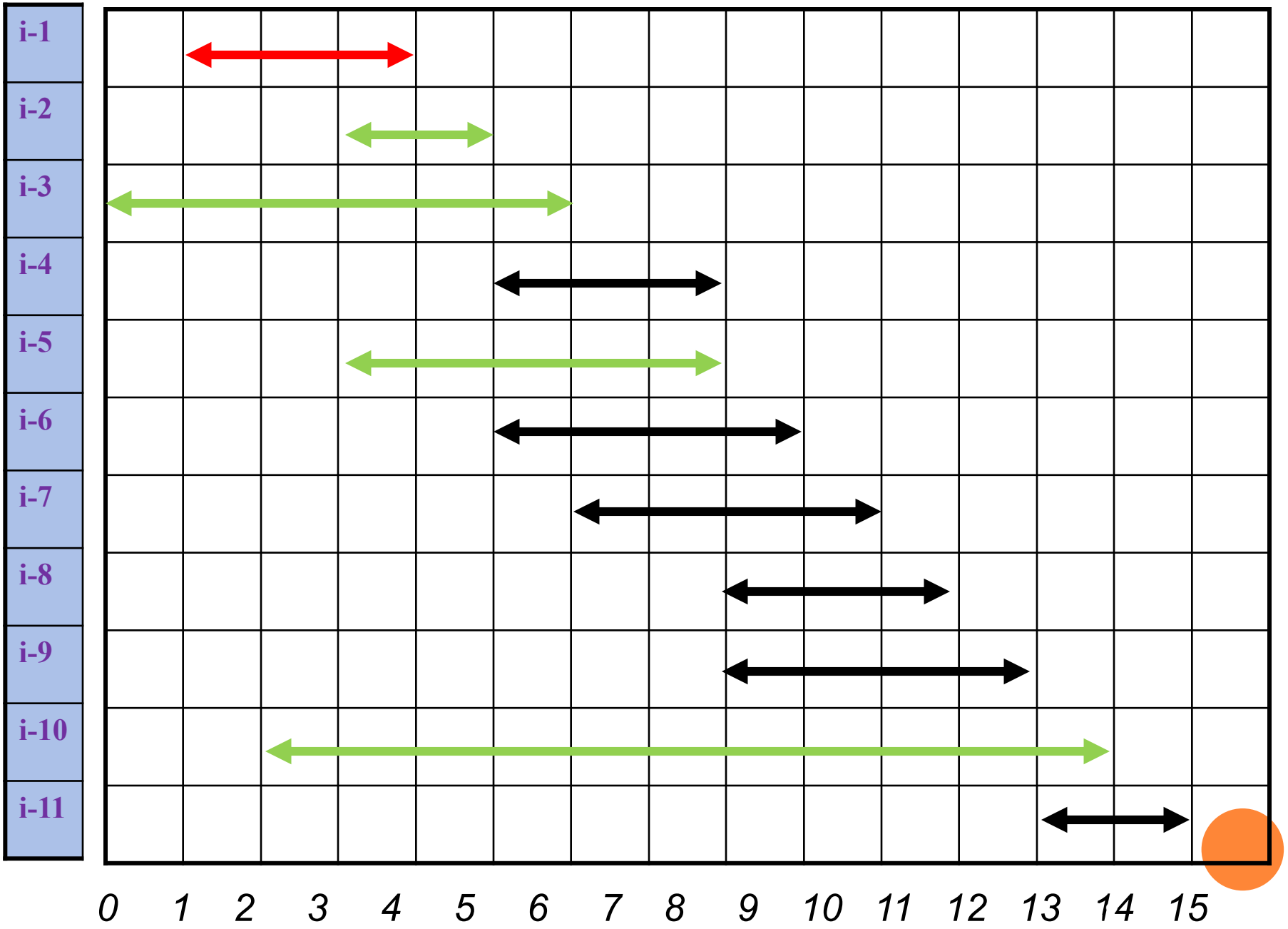
Removed from the list

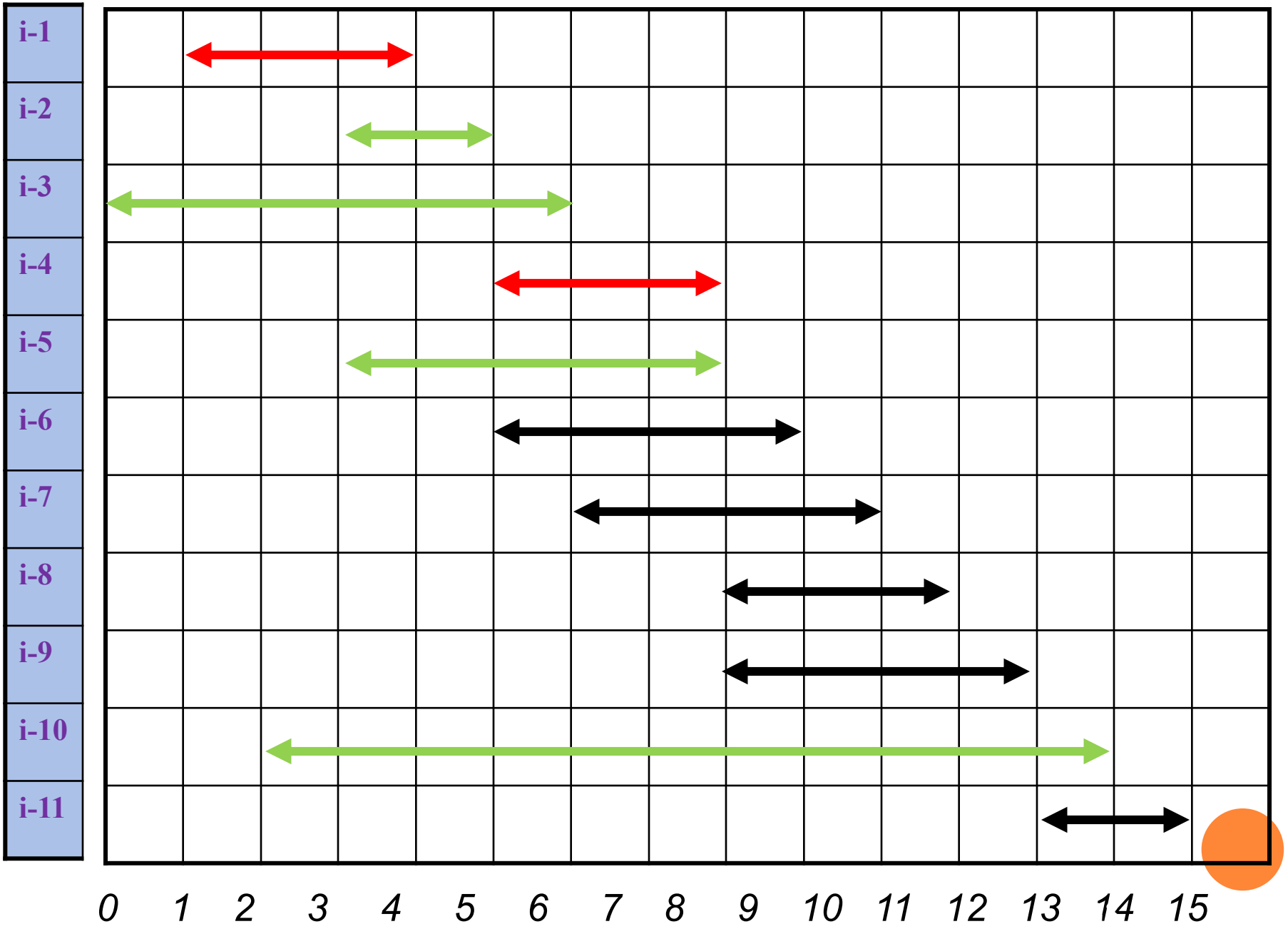
Early Finish Greedy

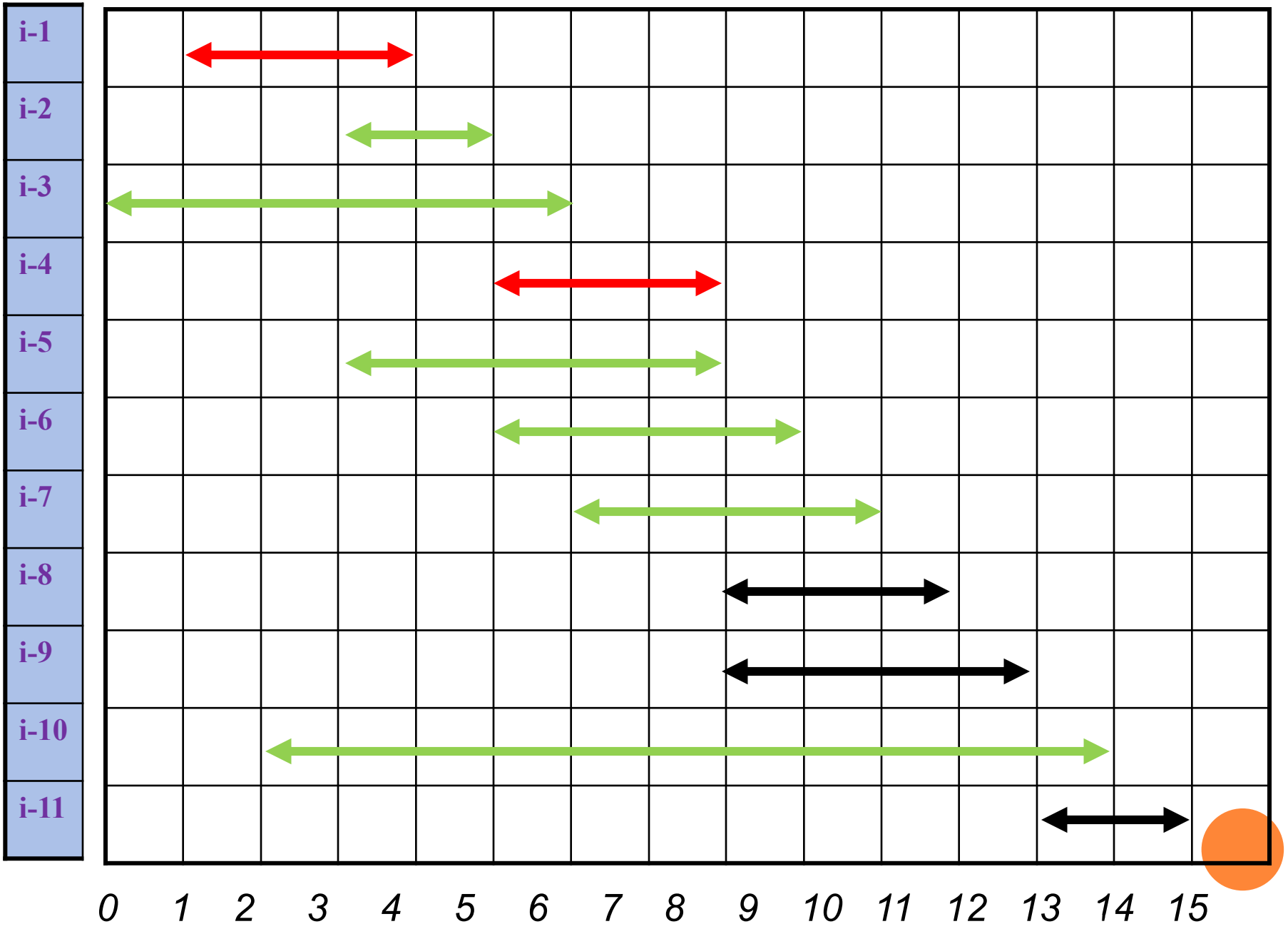
1. Select the activity with the earliest finish
2. Eliminate the activities that could not be scheduled
3. Repeat!

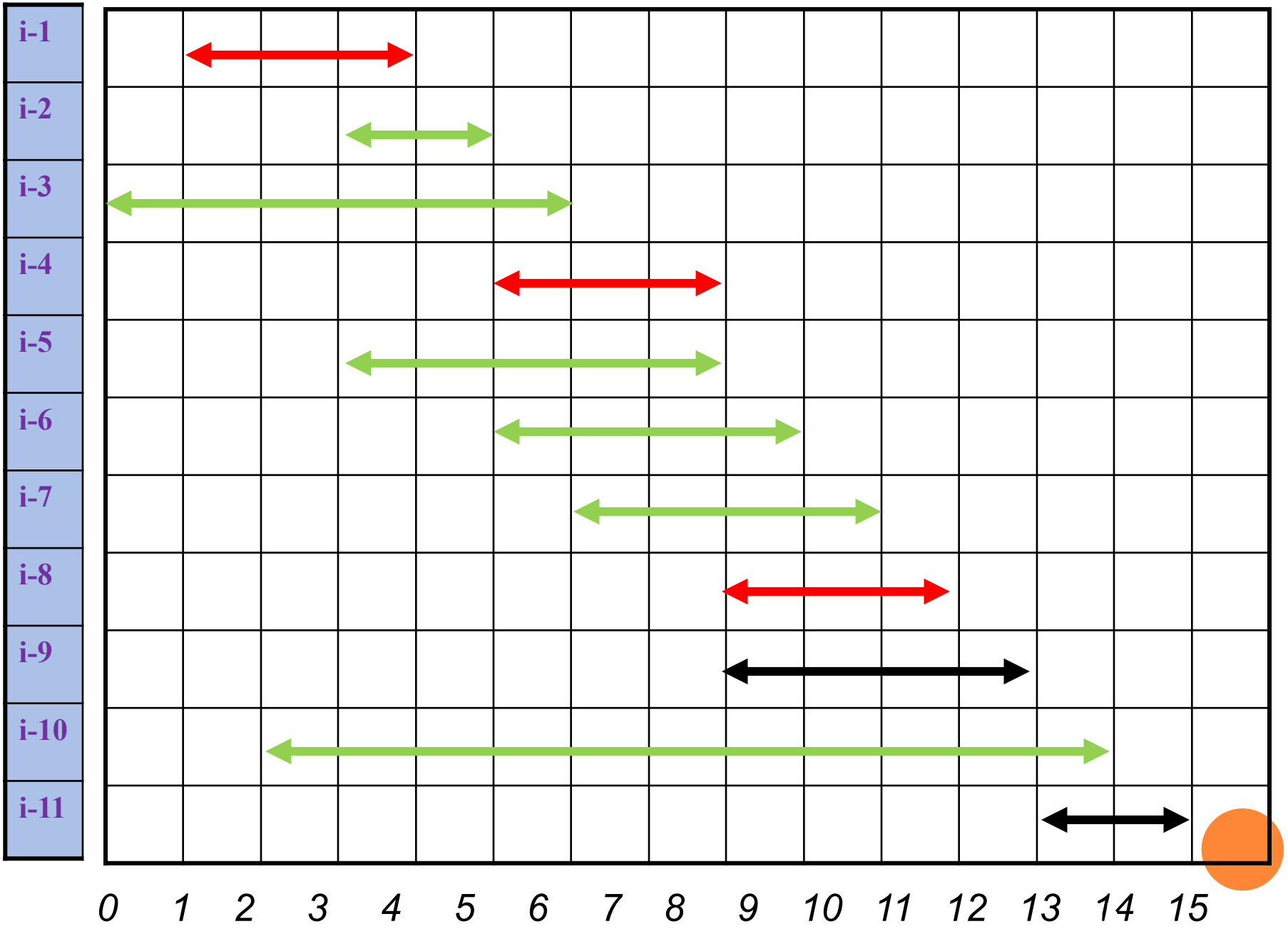

i-1
i-2
i-3
i-4
i-5
i-6
i-7
i-8
i-9
i-10
i-11
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

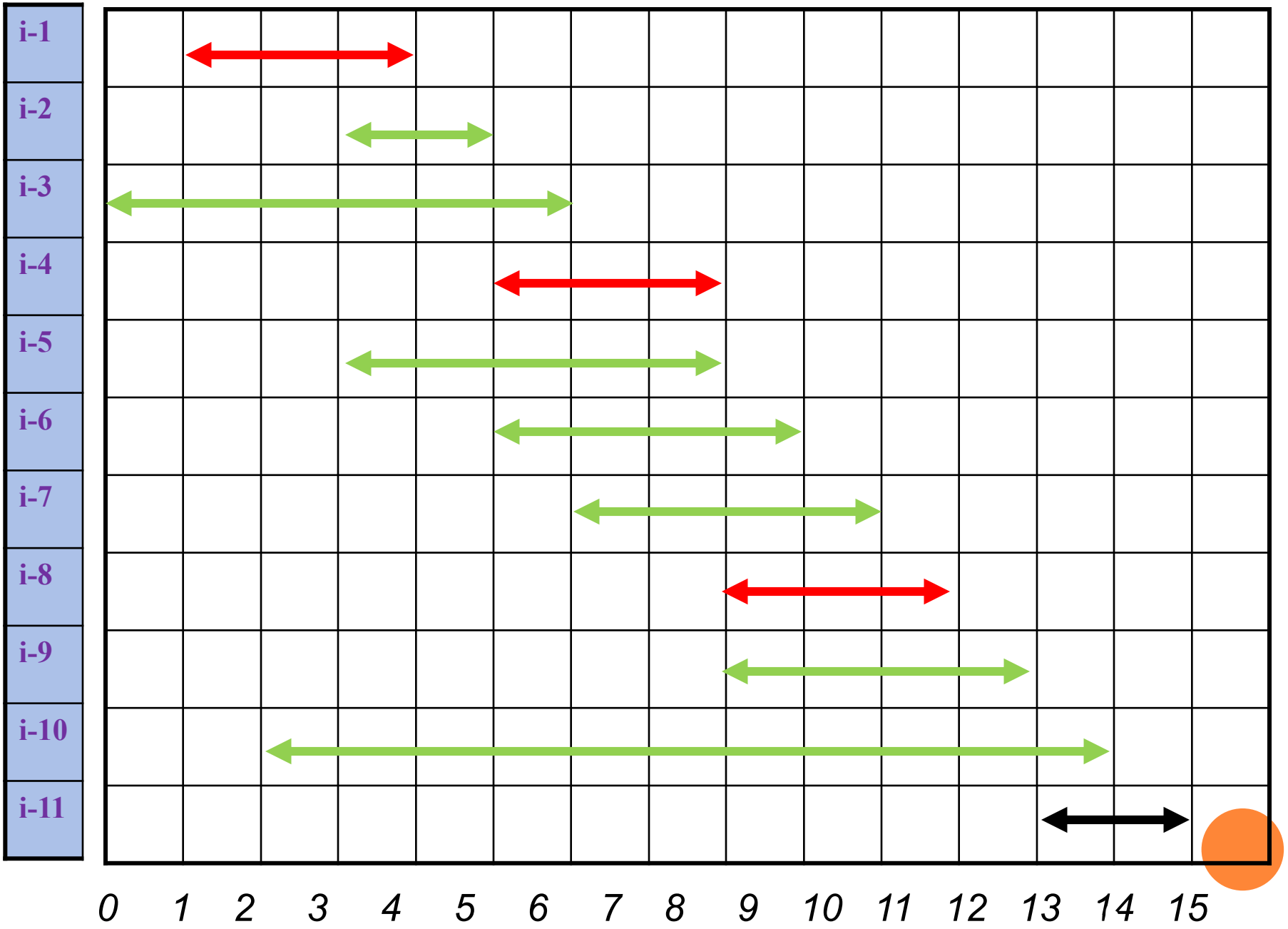

i-1
i-2
i-3
i-4
i-5
i-6
i-7
i-8
i-9
i-10
i-11
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15


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

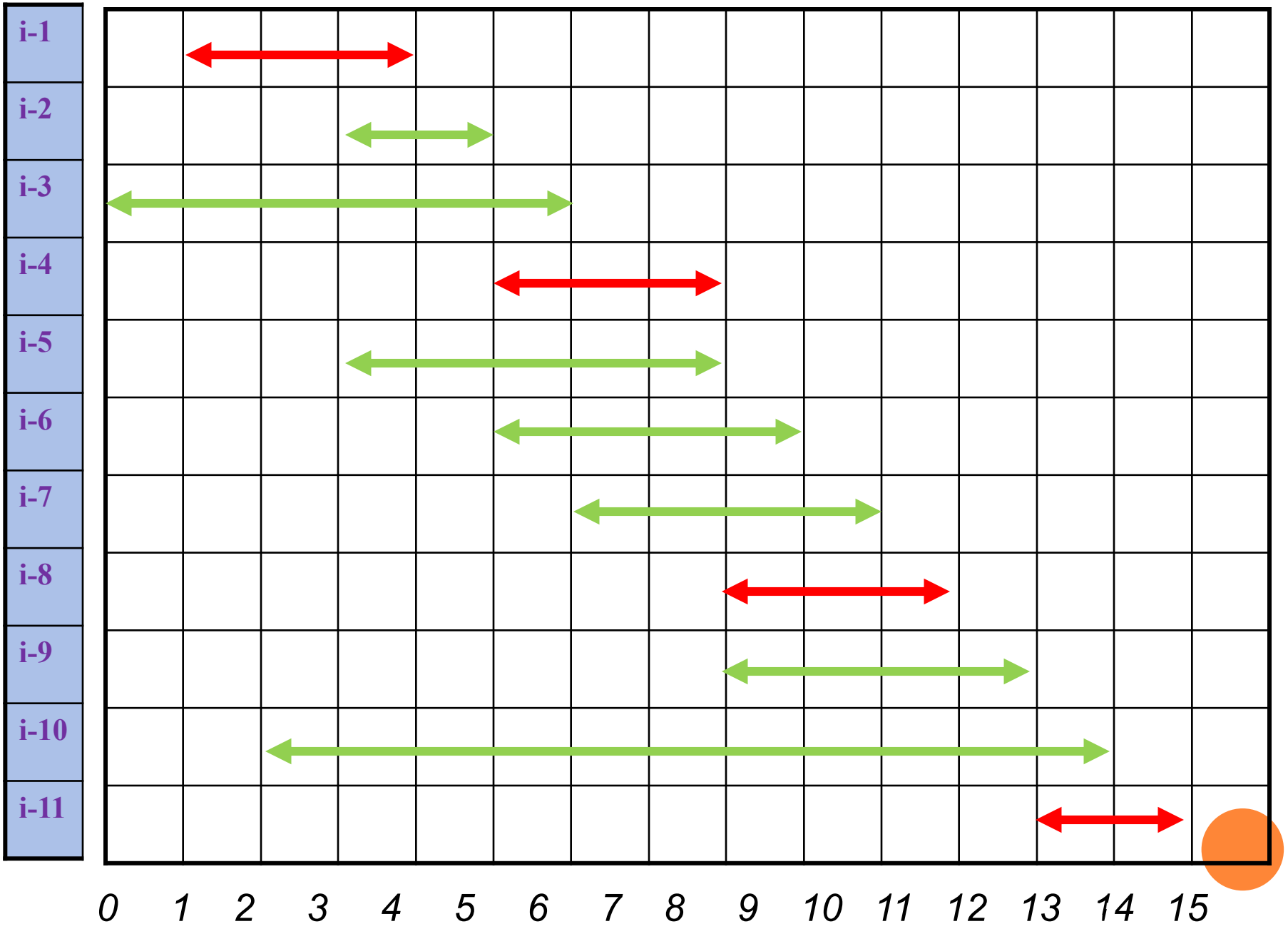

i-1
i-2
i-3
i-4
i-5
i-6
i-7
i-8
i-9
i-10
i-11
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15


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


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


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

Select the activity with the earliest finish



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 $\{a_2, a_4, a_9, a_{11}\}$*

Activity Selection Problem- Another Example

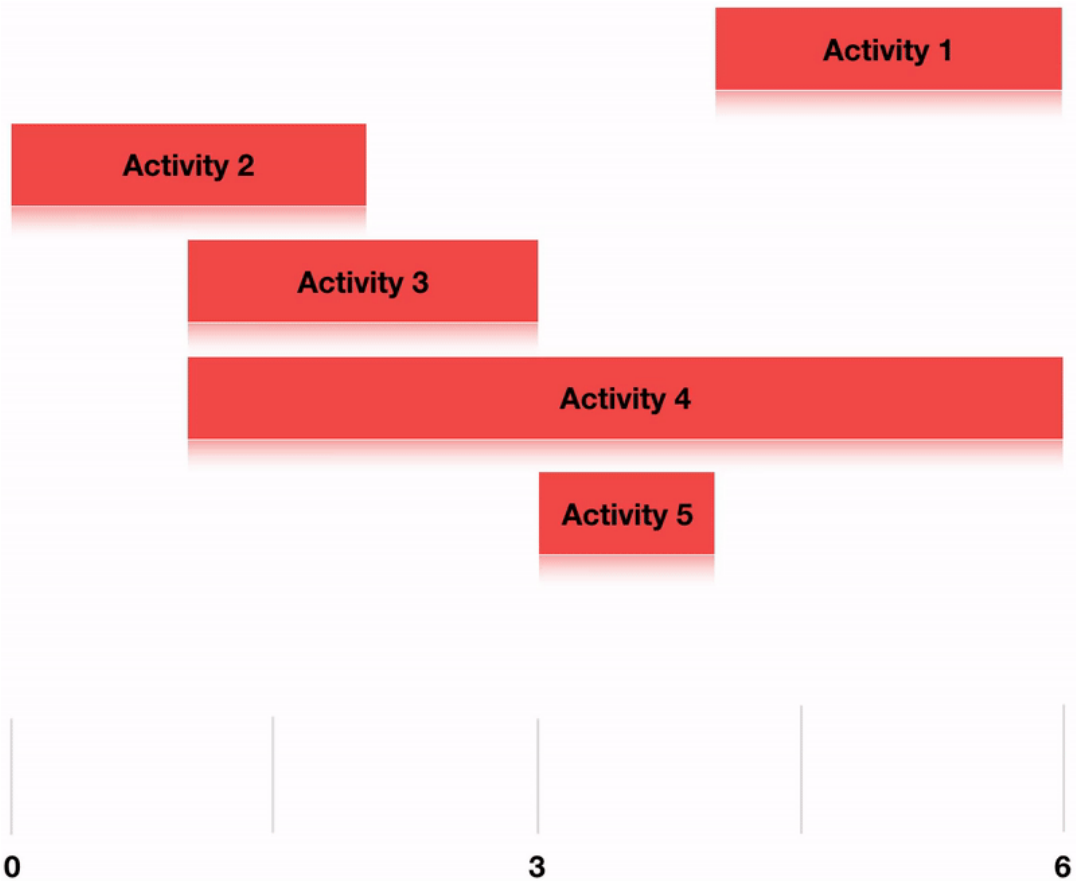
Activity (a_i)	2	3	5	1	4
s_i	0	1	3	4	1
f_i	2	3	4	6	6

Sorted according to finish time

Activity (a_i)	2	3	5	1	4
s_i	0	1	3	4	1
f_i	2	3	4	6	6

Sorted according to finish time

To take the element with the least finish time, we will iterate over the list of the activities and will select the first activity and then we will select the activity which is starting next after the currently selected activity finishes.



How to solve?

- Brute force
  - Generate all possible subsets of non-conflicting activities
  - Choose the largest subset
- Greedy algorithm
  - Steps:
    - Make greedy choice at the beginning of each iteration
    - Create sub problem
    - Solve the sub problem

Activity Selection Problem- Greedy Algorithm

- **Make greedy choices:** select a job to start with. Which one?
  - Select the job that finishes first
  - **Assumption:** Jobs are sorted according to finishing time.
- **Create sub problems:** leaving this job, leaves you with a smaller number of jobs to be selected.
- **Solve Sub problems:** Continue first two steps until the all the jobs are finished. (recursion!)

Activity Selection Problem –

```
//  n   -->  Total number of activities
//  s[] -->  An array that contains start time of all activities
//  f[] -->  An array that contains finish time of all activities
void Activities(int s[], int f[], int n)
{
    int i, j;

    printf ("Following activities are selected n");
    // The first activity always gets selected
    i = 0;

    printf("%d ", i);
    // Consider rest of the activities
    for (j = 1; j < n; j++)
    {
      // If this activity has start time greater than or equal to the
     //finish time of previously selected activity, then select it
      if (s[j] >= f[i])
      {
          printf ("%d ", j);
          i = j; // update the last scheduled activity
      }
    }
}
```

Running time complexity = O(n)

ACTIVITY SELECTION PROBLEM – RECURSIVE SOLUTION

- *s -> the set of start time*
- *f -> the set of finish time.*
- *k -> index of last selected activity*
- *n -> number of activities*

```
RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]      // find the first activity in S_k to finish
3        m = m + 1
4  if m ≤ n
5        return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)
6  else return ∅
```

Running time complexity = O(n)

- *Note:In order to start, we add the fictitious activity a_0 with $f_0 = 0$, so that subproblem S_0 is the entire set of activities S. The initial call, which solves the entire problem, is RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, n)*

Why Activity Selection Problem 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

Activity Selection Problem –

Analysis of the Algorithm

We can clearly see that the algorithm is taking a **O(n)** running time, where n is the number of activities.

if the arrays passed to the function are not sorted, we can sort them in **O(nlgn)** time.

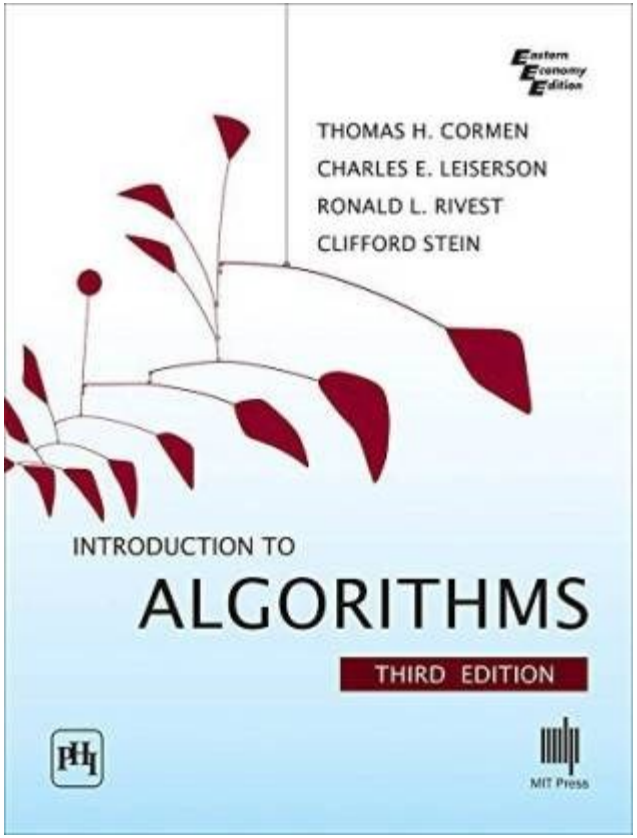
Activity Selection Problem –

Applications

- Scheduling events in a room having multiple competing events
- Scheduling and manufacturing multiple products having their time of production on the same machine
- Scheduling meetings in one room
- Several use cases in Operations Research

Reference

- Chapter 16
- **Introduction to Algorithms, 3rd Edition** Thomas H. Cormen



Thanks to All