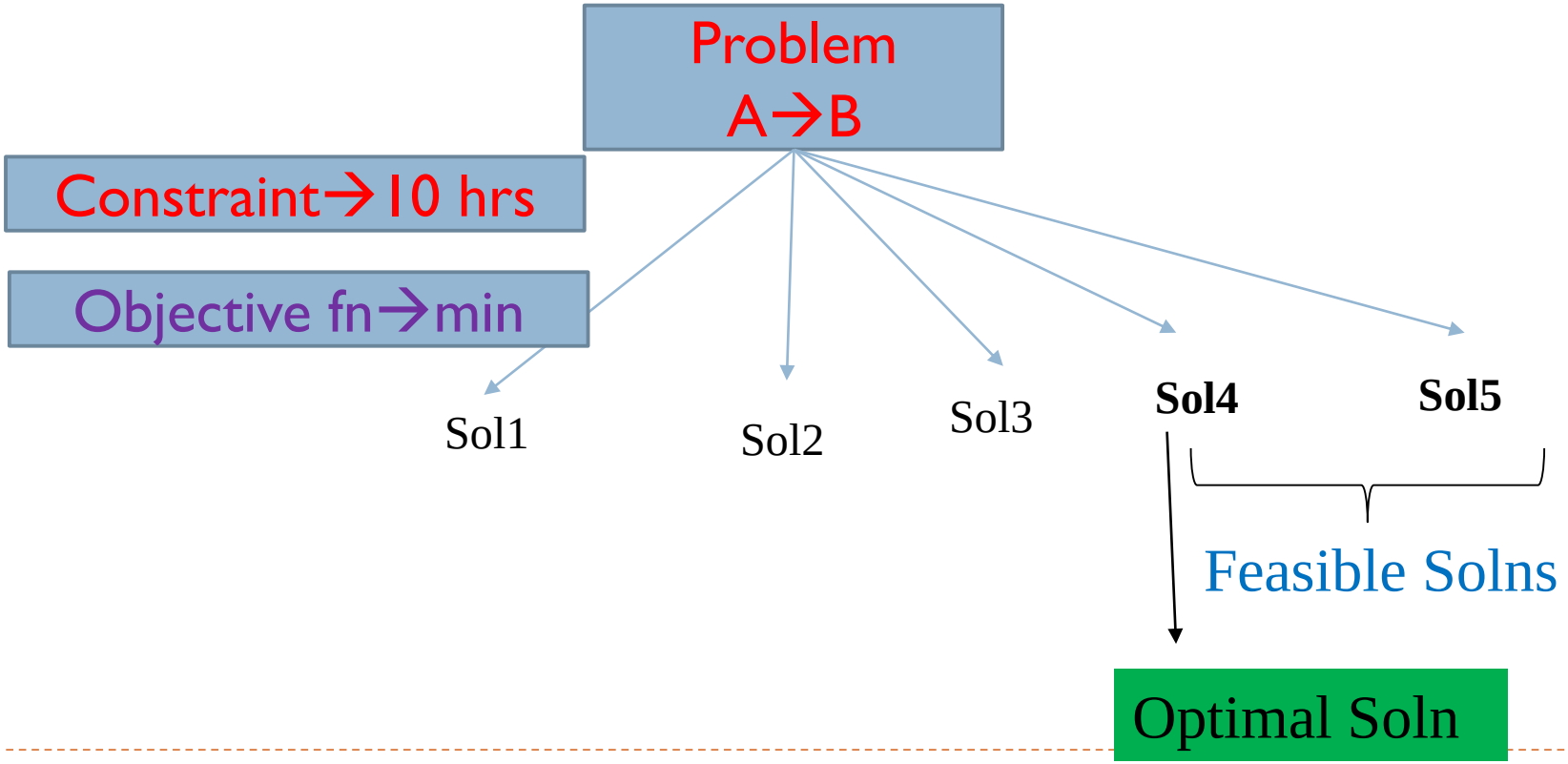
# Greedy Method

Activity Selection Problem

- Greedy Method – General Method
- Elements of Greedy Method
- Greedy Algorithms
  - Activity Selection Problem
  - Huffman Coding
  - Fractional knapsack problem

# Greedy Method

- A design strategy for solving Problems
  - Solves Optimization Problems
    - Requires min/max results



# 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

# Greedy Choice must be

- **Feasible**
  - Satisfy the problem's constraints
- **Locally optimal**
  - Be the best local choice among all feasible choices
- **Irrevocable**
  - Once made, the choice can't be changed on subsequent steps.

# The General Method

- An algorithm that works in stages considering one input at a time
- At each stage a decision is made whether the input gives an optimal solution
- This is done by considering the inputs in an order determined by some selection procedure
- If the partial solution is not optimal then it is not added to the solution
- The selection procedure is based on some optimization measure, the measure may be the objective function

# Greedy Algorithm

```
Algorithm Greedy(a,n)
{Solution:=∅
 for i:=1 to n do
 {
    x:=Select(a);
    if Feasible(Solution,x)
     Solution:=Union(solution,x);
 }
 return Solution;
}
```

# Greedy Algorithm

- Select → The function `Select` selects an input from `a[]` and removes it. The selected value is assigned to `x`.
- Feasible → Is a Boolean-valued function that determines whether `x` can be included into the `solution` vector.
- Union → The function `Union` combines `x` with the `Solution`.

# An Activity Selection Problem (Conference Scheduling Problem)

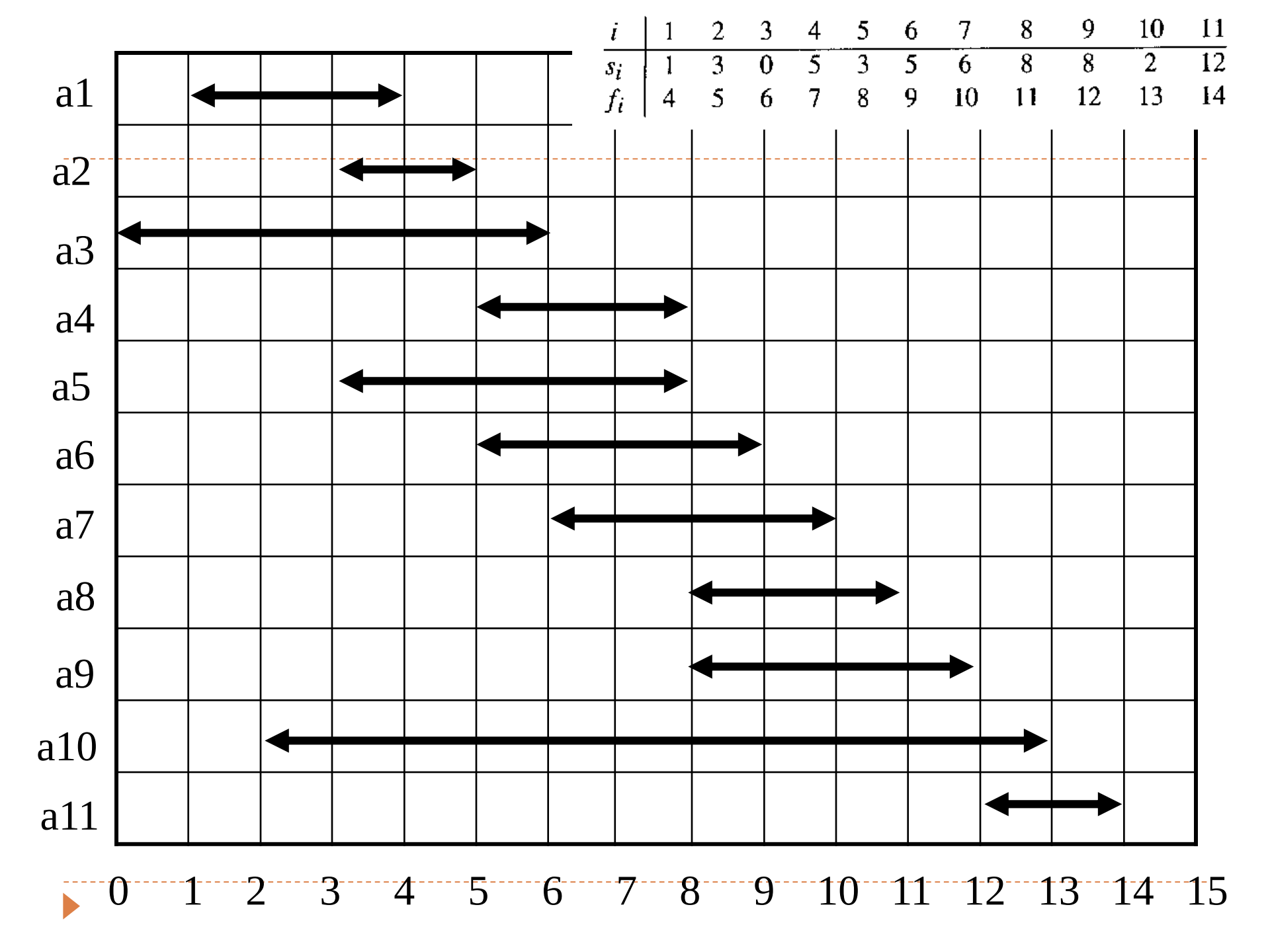
- **Input: A set of activities S = {$a_1$,…, $a_n$}**
- Each activity has start time and a finish time
  - $a_i$=($s_i$, $f_i$)
- Two activities are compatible if and only if their interval does not overlap
- **Output: a maximum-size subset of mutually compatible activities**

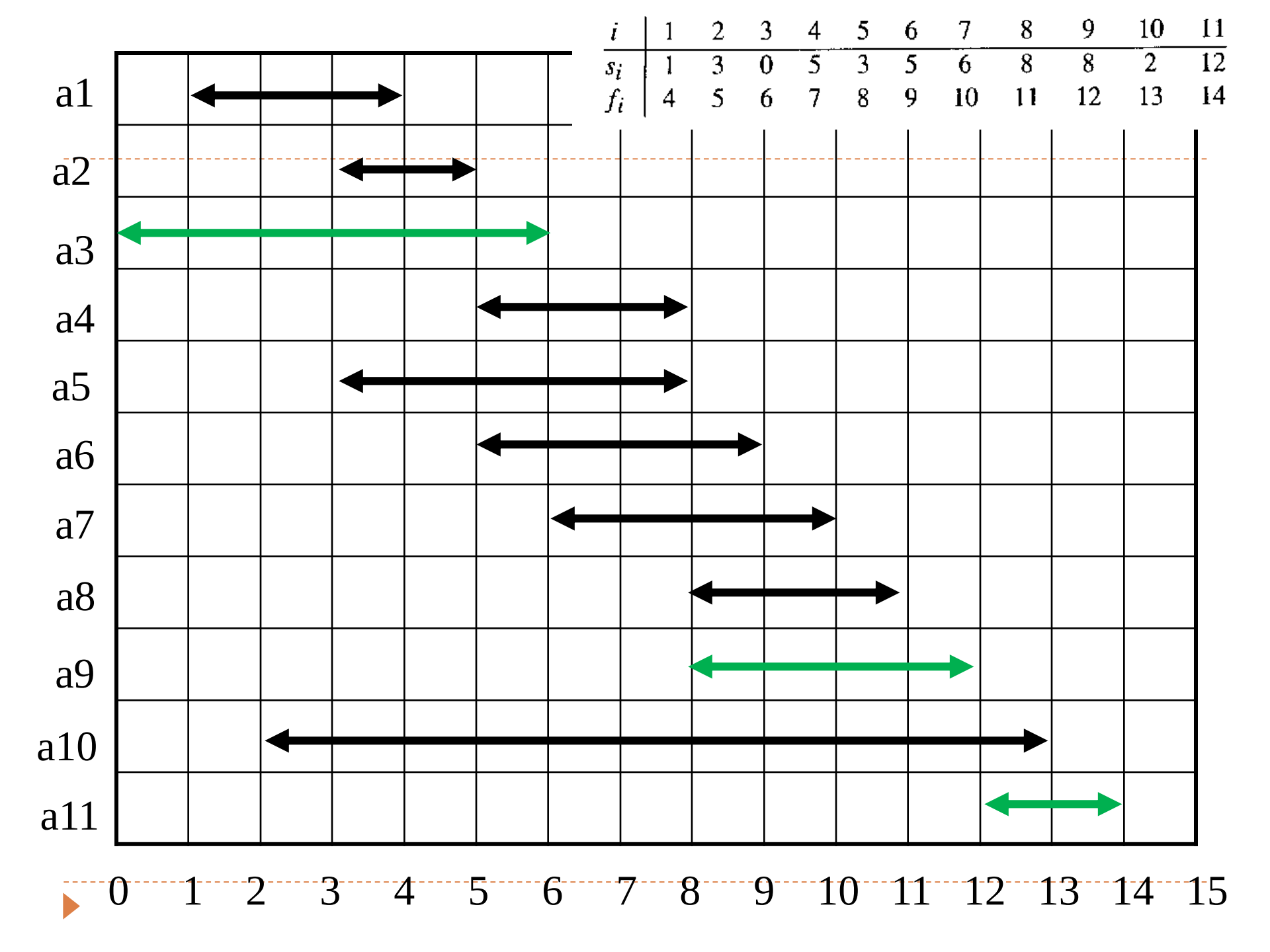
# The Activity Selection Problem

- Here are a set of start and finish times

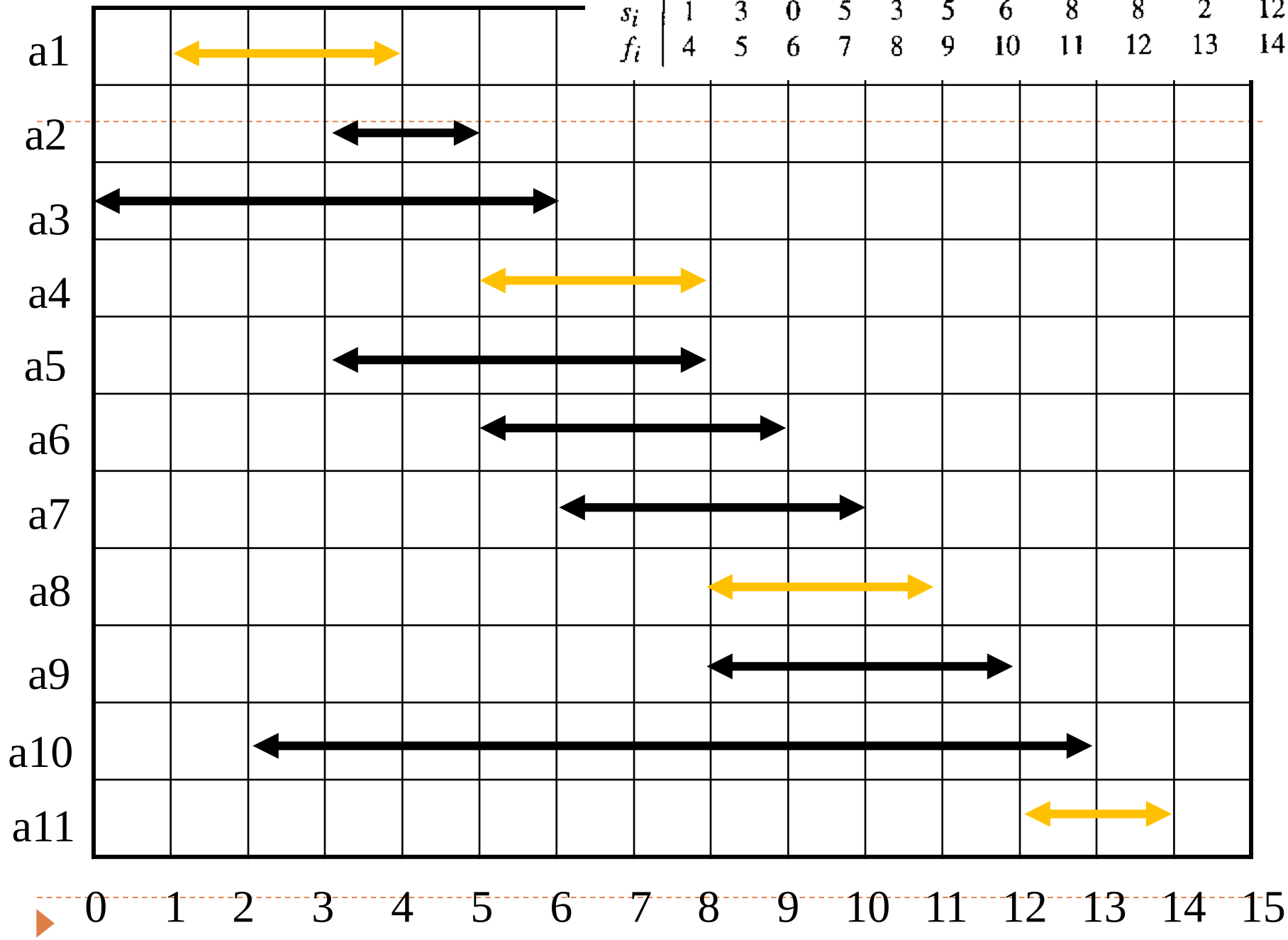
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 1 | 3 | 0 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| $f_i$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

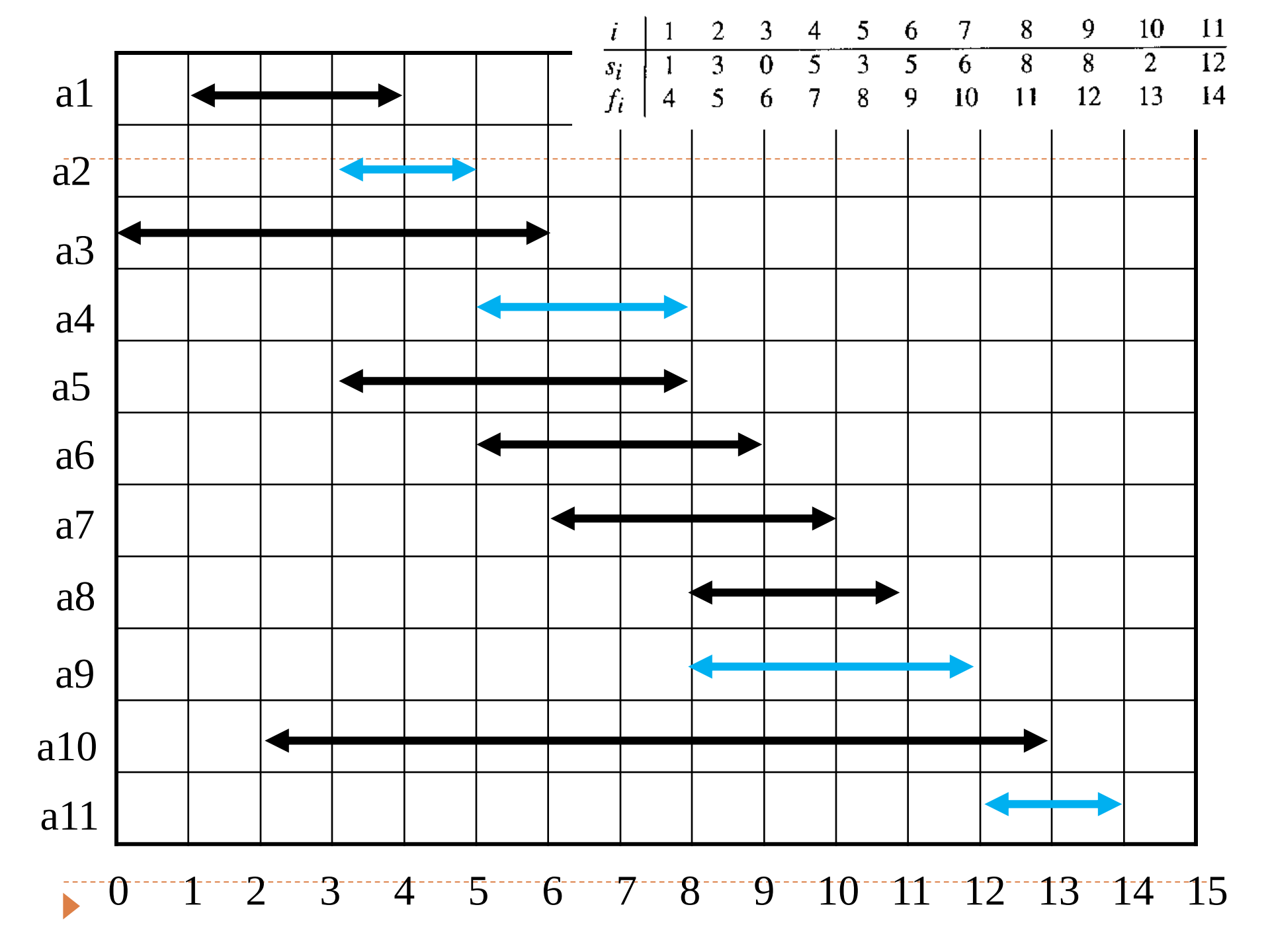
- What is the maximum number of activities that can be completed?
  - $\{a_3, a_9, a_{11}\}$ can be completed
  - But so can $\{a_1, a_4, a_8, a_{11}\}$ which is a larger set
  - But it is not unique, consider $\{a_2, a_4, a_9, a_{11}\}$


i 1 2 3 4 5 6 7 8 9 10 11
si 1 3 0 5 3 5 6 8 8 2 12
fi 4 5 6 7 8 9 10 11 12 13 14
a1
a2
a3
a4
a5
a6
a7
a8
a9
a10
a11
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15


i 1 2 3 4 5 6 7 8 9 10 11
si 1 3 0 5 3 5 6 8 8 2 12
fi 4 5 6 7 8 9 10 11 12 13 14
a1
a2
a3
a4
a5
a6
a7
a8
a9
a10
a11
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

| $i$ | 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 |

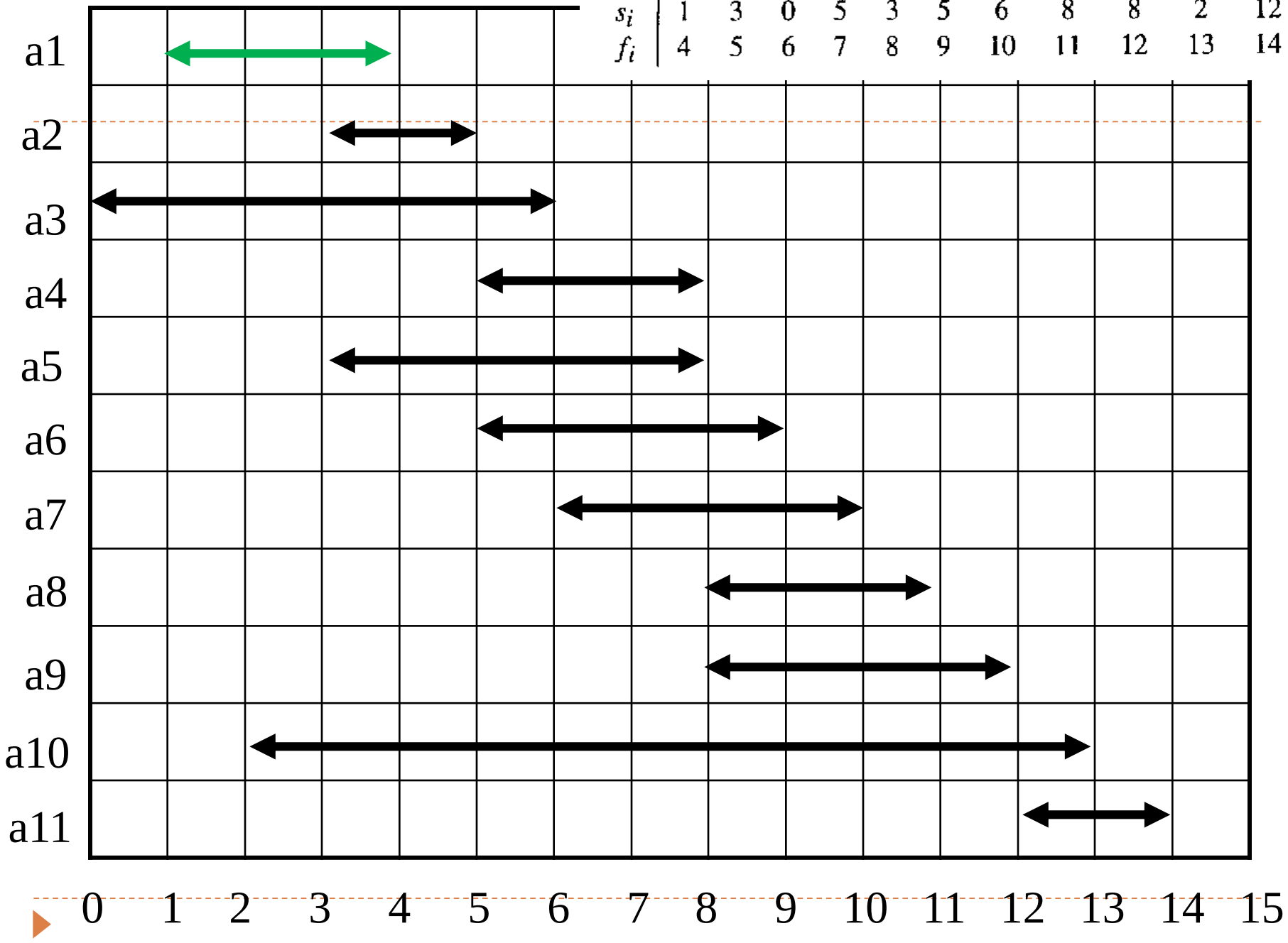



i 1 2 3 4 5 6 7 8 9 10 11
si 1 3 0 5 3 5 6 8 8 2 12
fi 4 5 6 7 8 9 10 11 12 13 14
a1
a2
a3
a4
a5
a6
a7
a8
a9
a10
a11
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

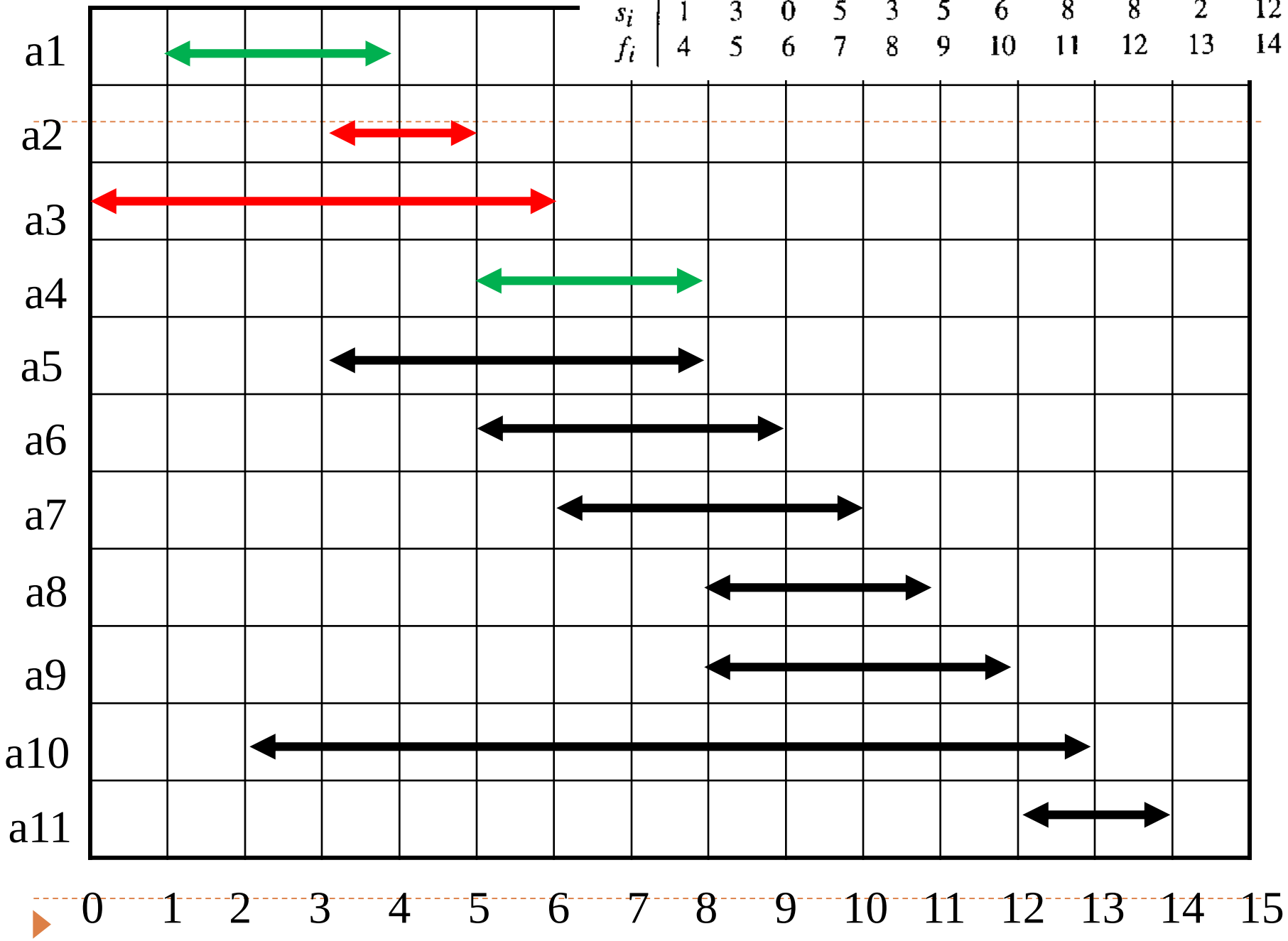
# Early Finish Greedy

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

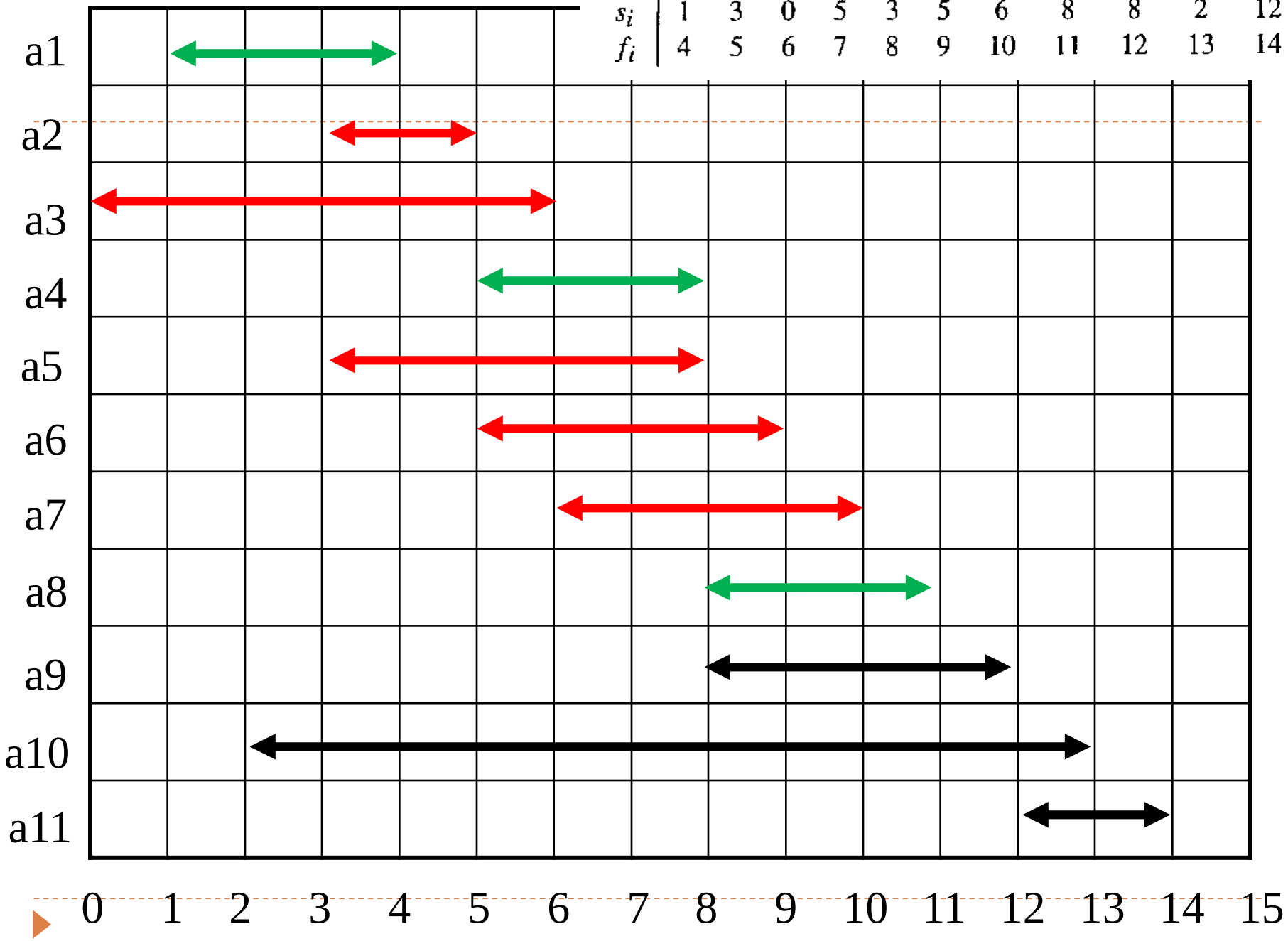
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 1 | 3 | 0 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| $f_i$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |



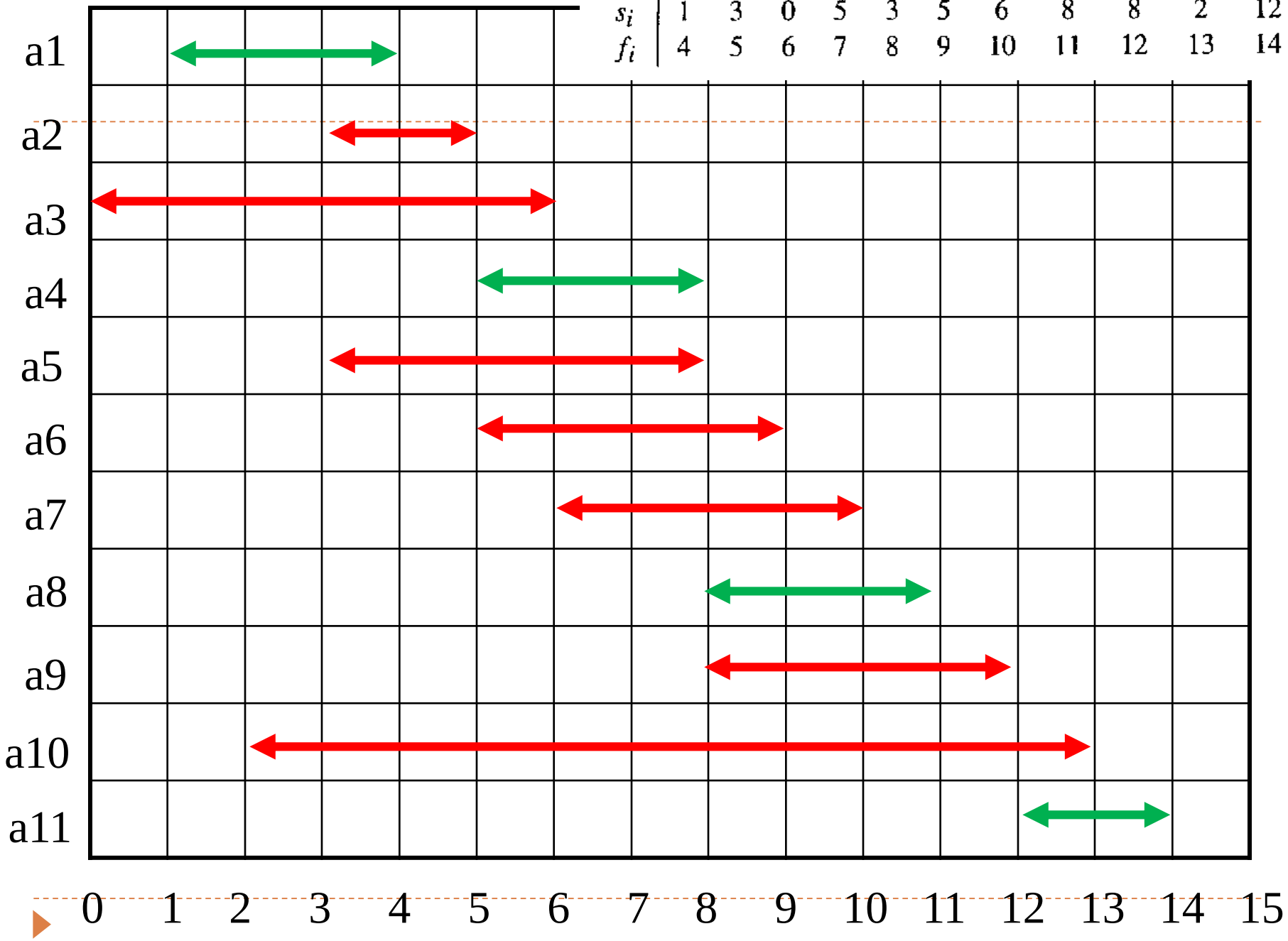
| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 1 | 3 | 0 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| $f_i$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |



| $i$ | 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 |



| $i$ | 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 |



# Problem

| 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 |

# 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

# Elements of Greedy Strategy

- A 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
- 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 the sub-problems.
  - If an optimal solution A to S begins with activity 1,

A'=A – {1} is optimal to S'={ i∈ $S$: $si >= f_1$}

# Greedy Algorithm Design

Comparison:

Dynamic Programming

- At each step, the choice is determined based on solutions of subproblems.
- Sub-problems are solved first.
- Bottom-up approach
- Can be slower, more complex

Greedy Algorithms

- At each step, we quickly make a choice that currently looks best.
  --A local optimal (greedy) choice.
- Greedy choice can be made first before solving further sub-problems.
- Top-down approach
- Usually faster, simpler