

# **Design and Analysis of Algorithm**

## **Greedy Methods** **(Activity Selection Algorithm, Task Scheduling, Huffman Coding )**

**Lecture – 45 - 50**

# Overview

- A greedy algorithm always makes the choice that looks best at the moment. (i.e. it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution).
- The objective of this section is to explore optimization problems that are solvable by greedy algorithms.

# Greedy Algorithm

- In mathematics, computer science and economics, an optimization problem is the problem of finding the best solution from all feasible solutions.
- Algorithms for optimization problems typically go through a sequence of steps, with a set of choices at each step.
- Many optimization problems can be solved using a greedy approach.
- Greedy algorithms are simple and straightforward.

# Greedy Algorithm

- A greedy algorithm always makes the choice that looks best at the moment.
- That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.
- Greedy algorithms do not always yield optimal solutions, but for many problems they do.
- These algorithms are easy to invent, easy to implement and most of the time provides best and optimized solution.

# Greedy Algorithm

- Application of Greedy Algorithm:
  - A simple but nontrivial problem, the **activity-selection problem**, for which a greedy algorithm efficiently computes a solution.
  - In combinatorics,(a branch of mathematics), a 'matroid' is a structure that abstracts and generalizes the notion of linear independence in vector spaces. Greedy algorithm always produces an optimal solution for such problems. **Scheduling unit-time tasks with deadlines and penalties** is an example of such problem.

# Greedy Algorithm

- Application of Greedy Algorithm:
  - An important application of greedy techniques is the design of **data-compression codes** (i.e. **Huffman code**) .
  - The greedy method is quite powerful and works well for a wide range of problems. They are:
    - Minimum-spanning-tree algorithms  
(Example: Prims and Kruskal algorithm)
    - Single Source Shortest Path.  
(Example: Dijkstra's and Bellman ford algorithm)

# Greedy Algorithm

- Application of Greedy Algorithm:
  - A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems.
  - This property is a key ingredient of assessing the applicability of **dynamic programming** as well as **greedy algorithms**.
  - The subtleties between the above two techniques are illustrated with the help of two variants of a classical optimization problem known as **knapsack problem**. These variants are:
    - **0-1 knapsack problem** (Dynamic Programming)
    - **Fractional knapsack problem** (Greedy Algorithm)

# Greedy Algorithm

- **Problem 1: An activity-selection problem**

- Is a problem of scheduling several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities.
- Let us take a set  $S = \{a_1, a_2, a_3, \dots, a_n\}$  on  $n$  proposed activities that wish to use a resource (i.e. lecture hall) which can serve only one activity at a time.
- Assume that the activities are sorted in monotonically increasing order of finish time:

$$f_1 \leq f_2 \leq f_3 \leq \dots \leq f_{n-1} \leq f_n$$



# Greedy Algorithm

- **Problem 1: An activity-selection problem**

- Each activity  $a_i$  has a start time  $s_i$  and a finish time  $f_i$ , where  $0 \leq s_i < f_i < \infty$ .
- Activities  $a_i$  and  $a_j$  are compatible if the intervals  $[s_i, f_i]$  and  $[s_j, f_j]$  do not overlap. (i.e.  $[s_j \geq f_i]$  ).
- In the activity-selection problem, the main goal is to select a maximum-size subset of mutually compatible activities.

# Greedy Algorithm

- **Problem 1: An activity-selection problem**

Example 1: Given 10 activities with their start and finish time compute a schedule where the largest number of activities take place.

Activity	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$	$a_8$	$a_9$	$a_{10}$
$s_i$	1	2	3	4	7	8	9	9	11	12
$f_i$	3	5	4	7	10	9	11	13	12	14

# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

First arrainging the following activities in increasing order on their finishing

Activity	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$	$a_8$	$a_9$	$a_{10}$
$s_i$	1	2	3	4	7	8	9	9	11	12
$f_i$	3	5	4	7	10	9	11	13	12	14



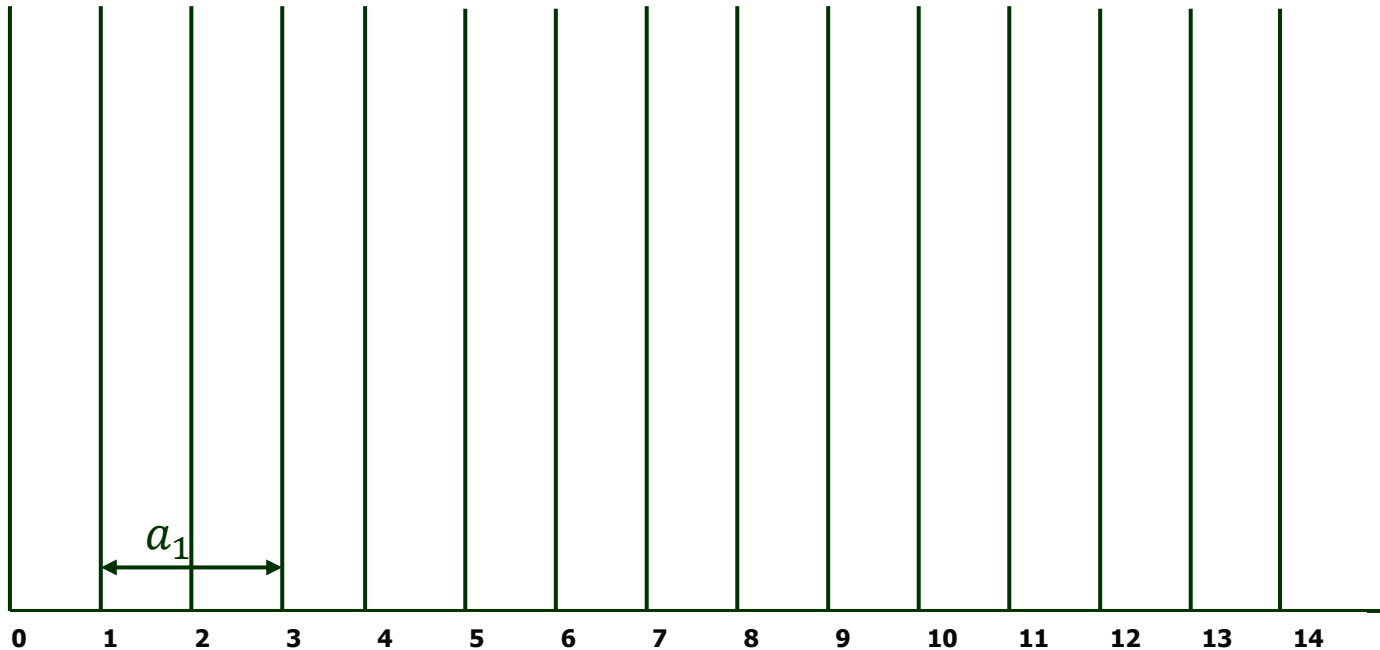
Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14

# Greedy Algorithm

- **Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14

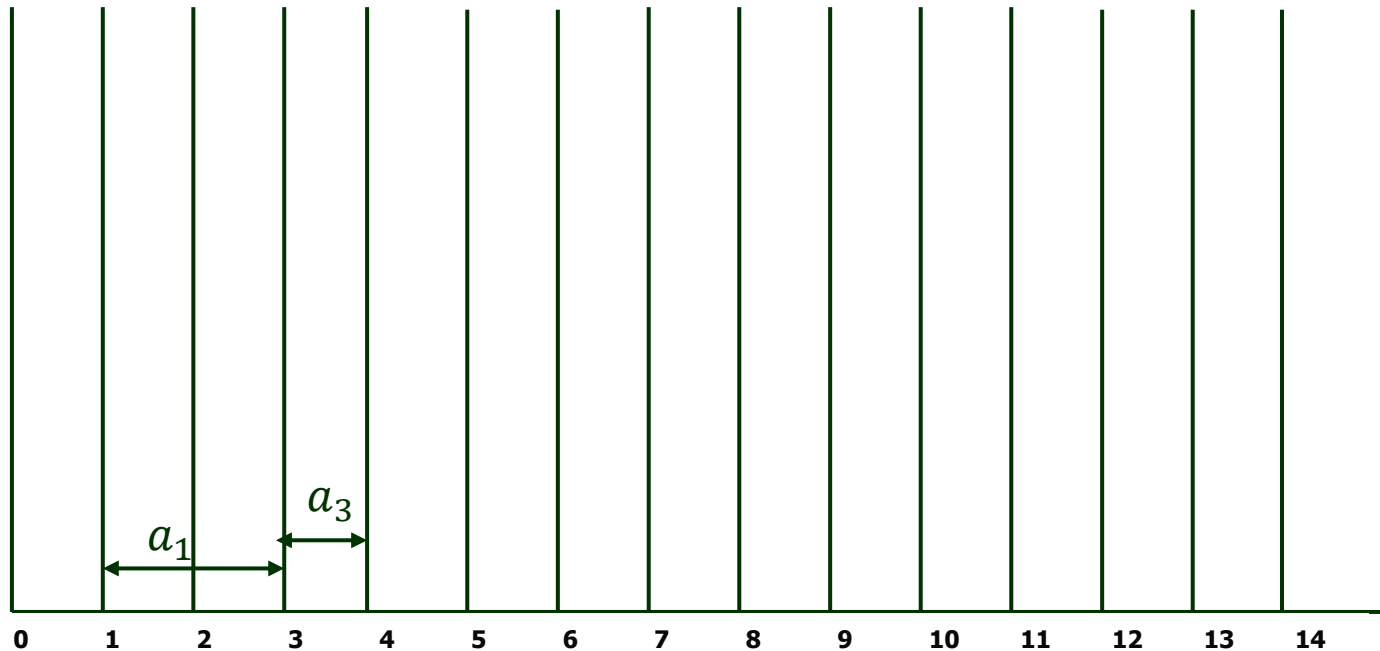


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14

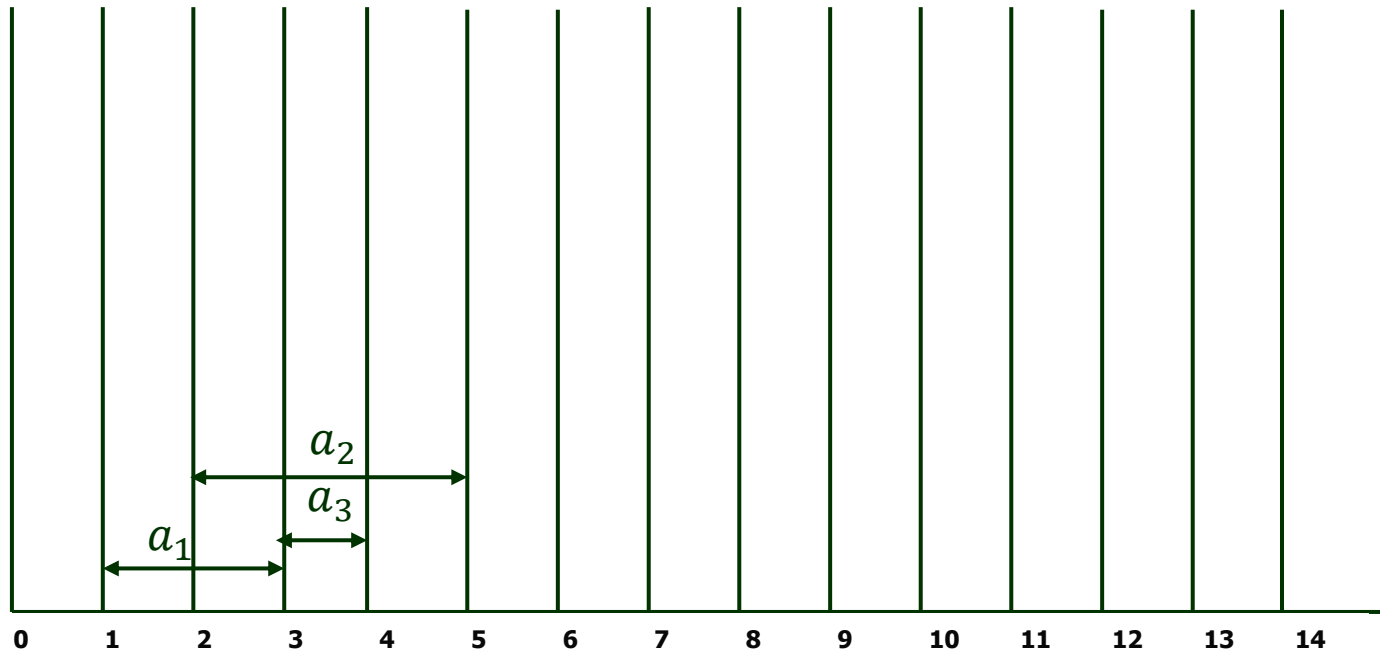


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14

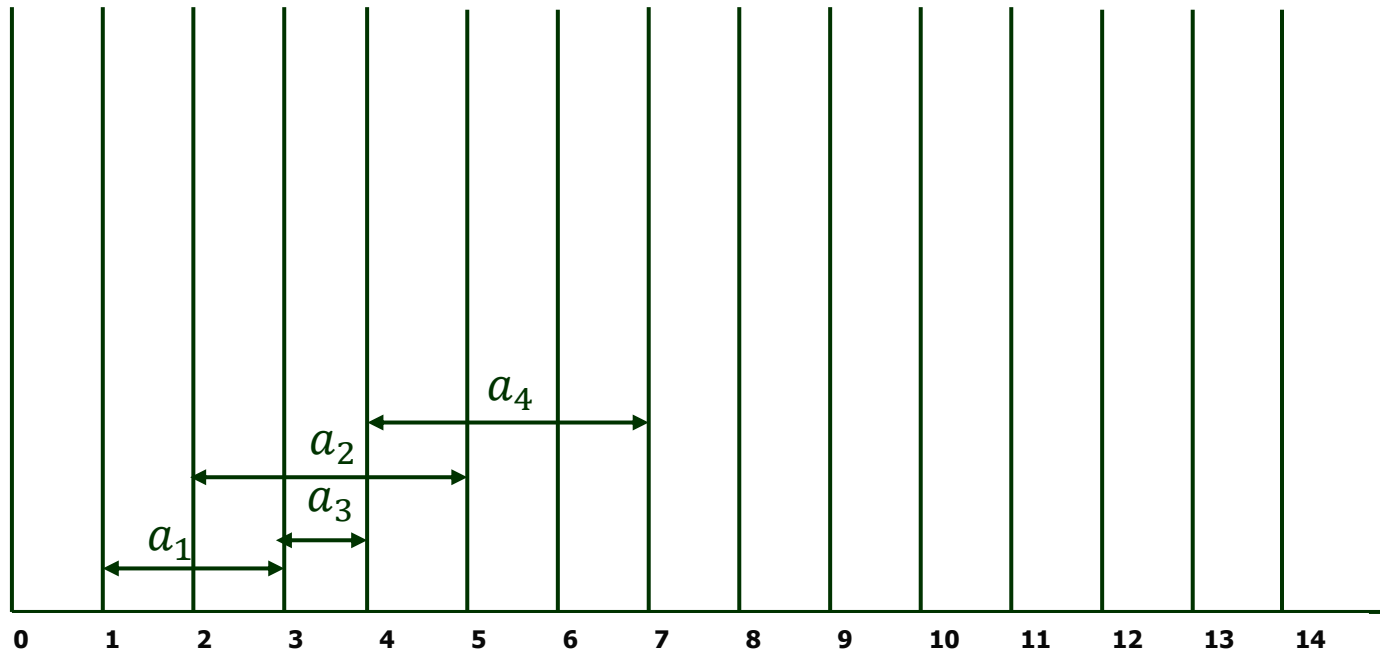


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14

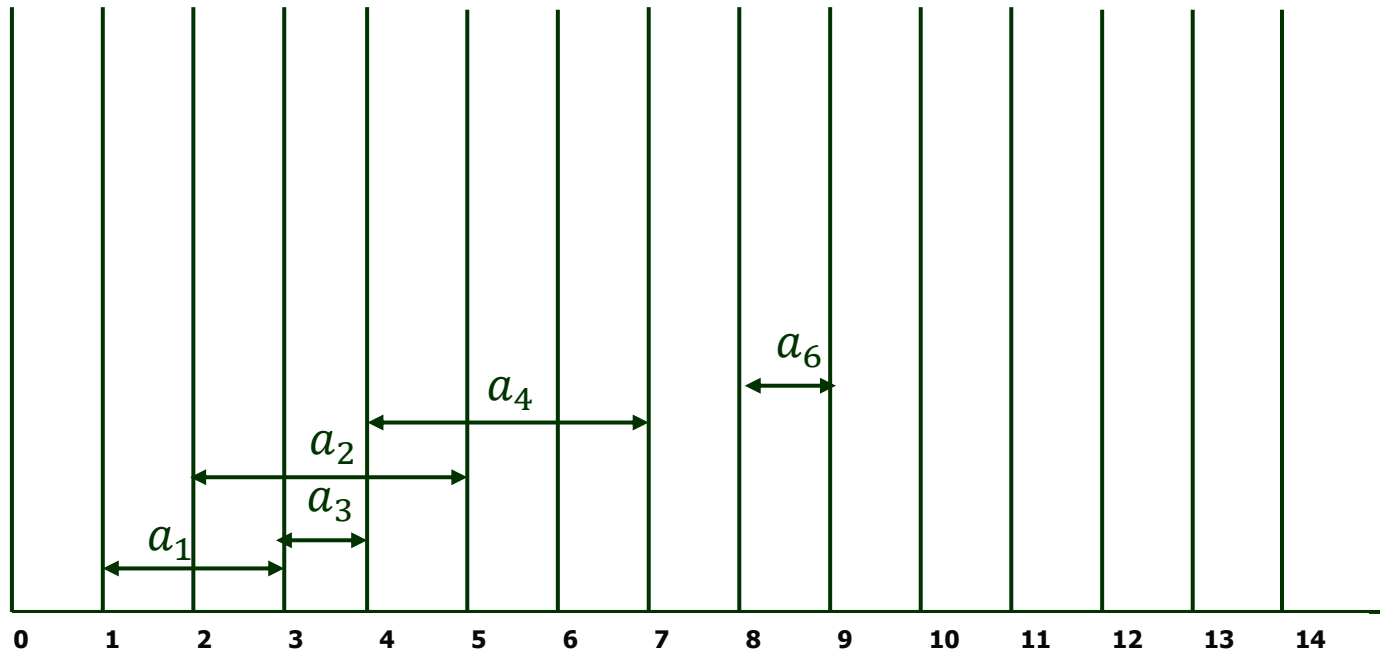


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14



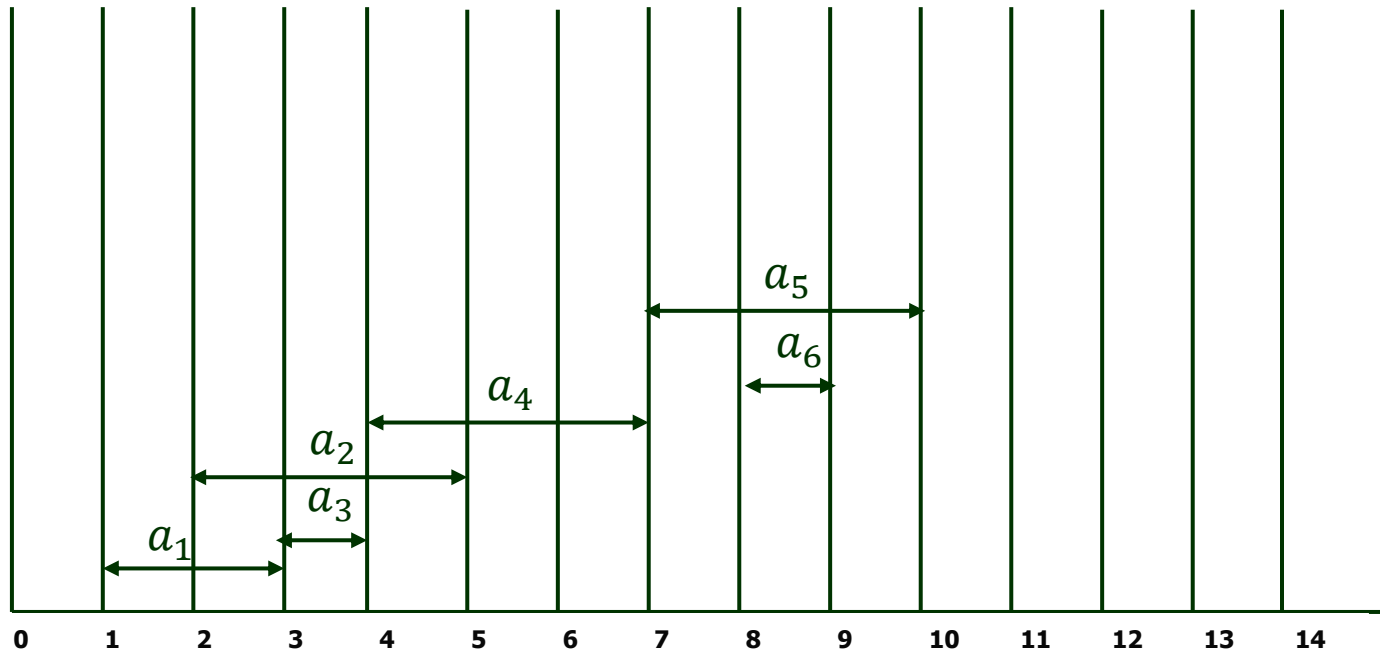


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14

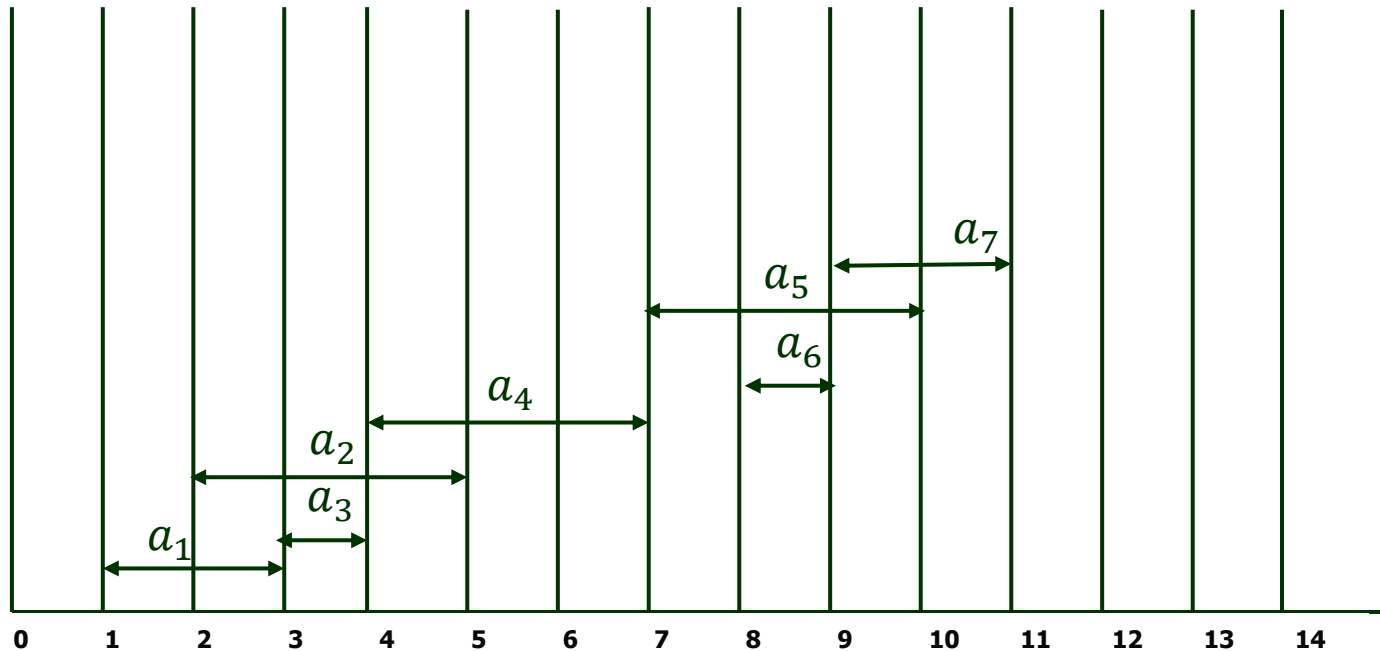


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14

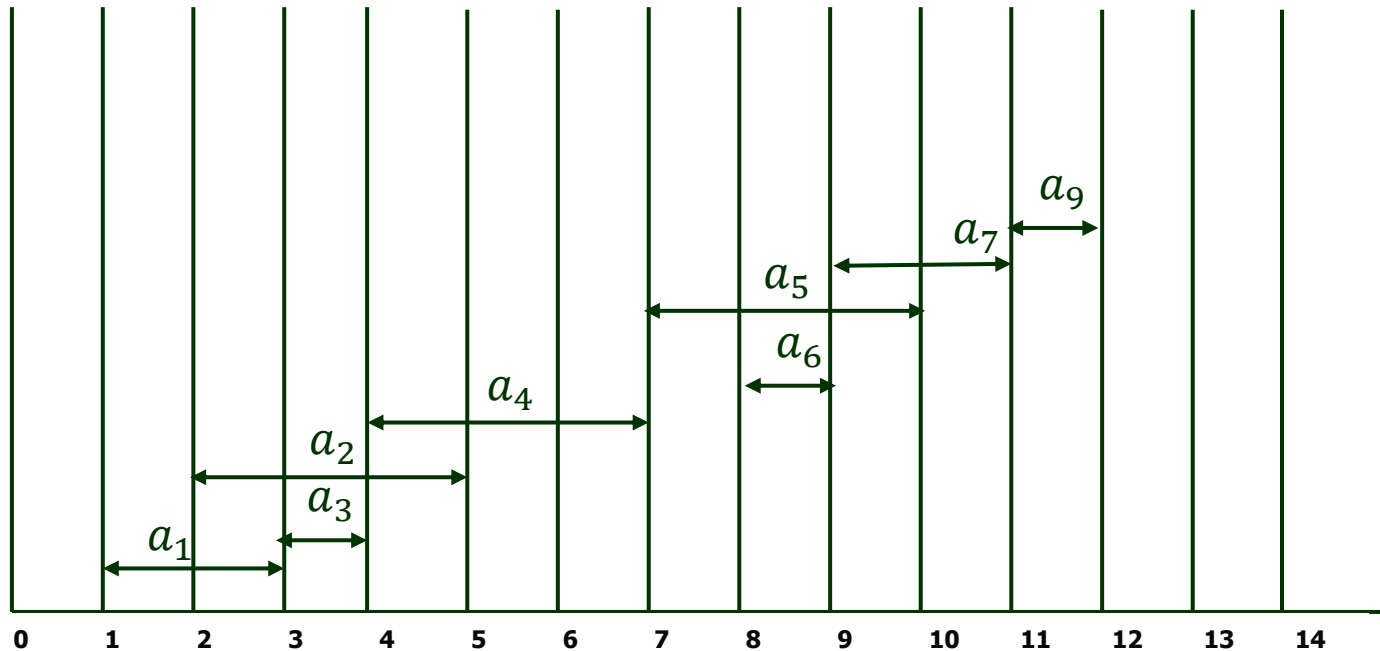


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14

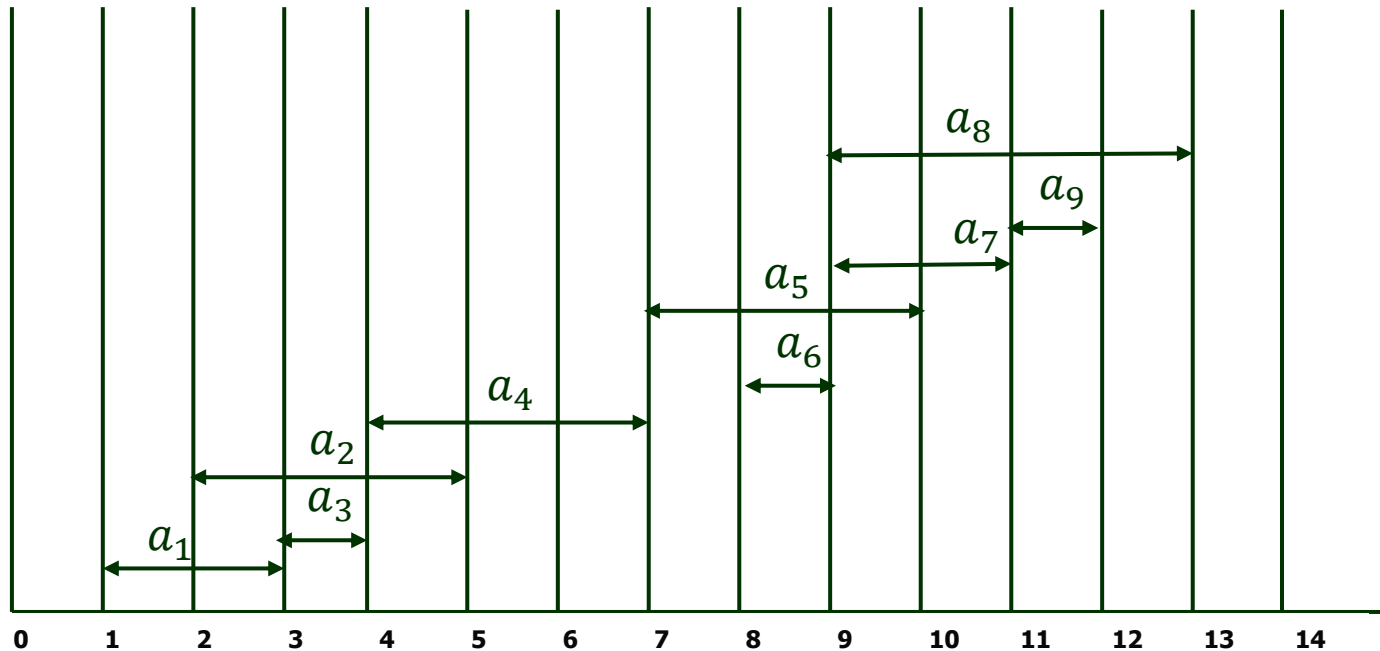


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14

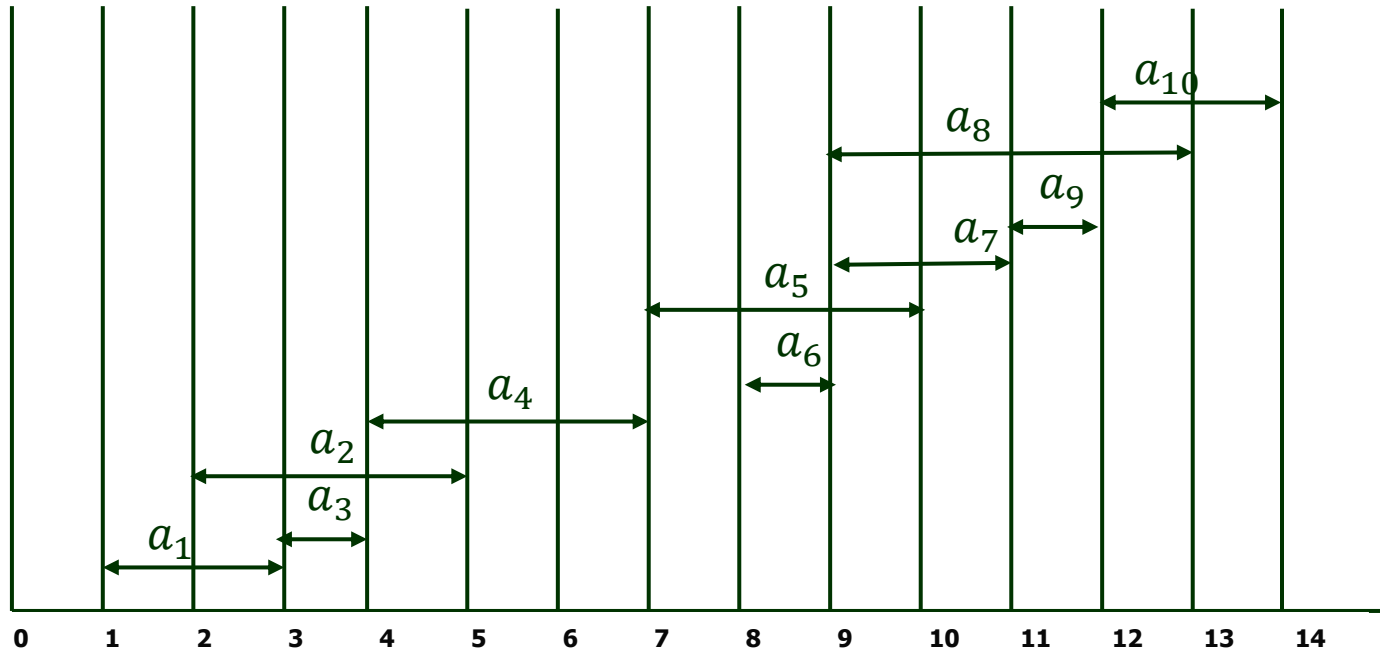


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14

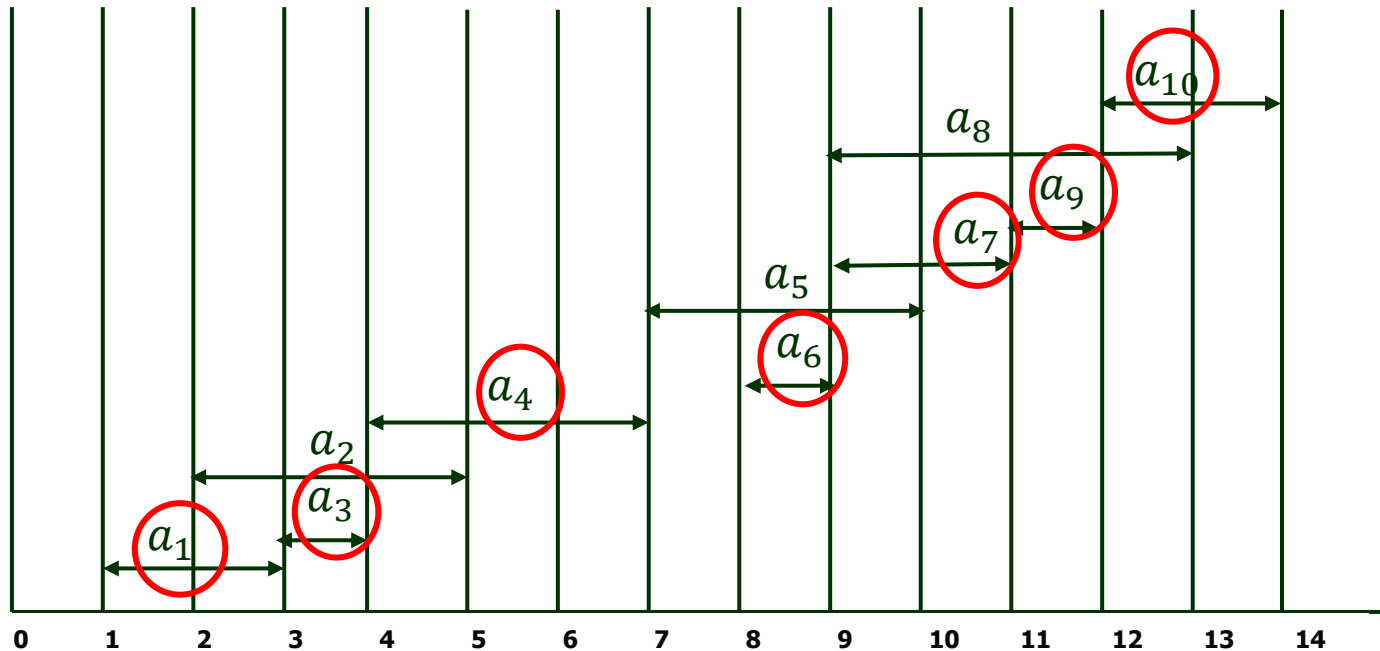


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14

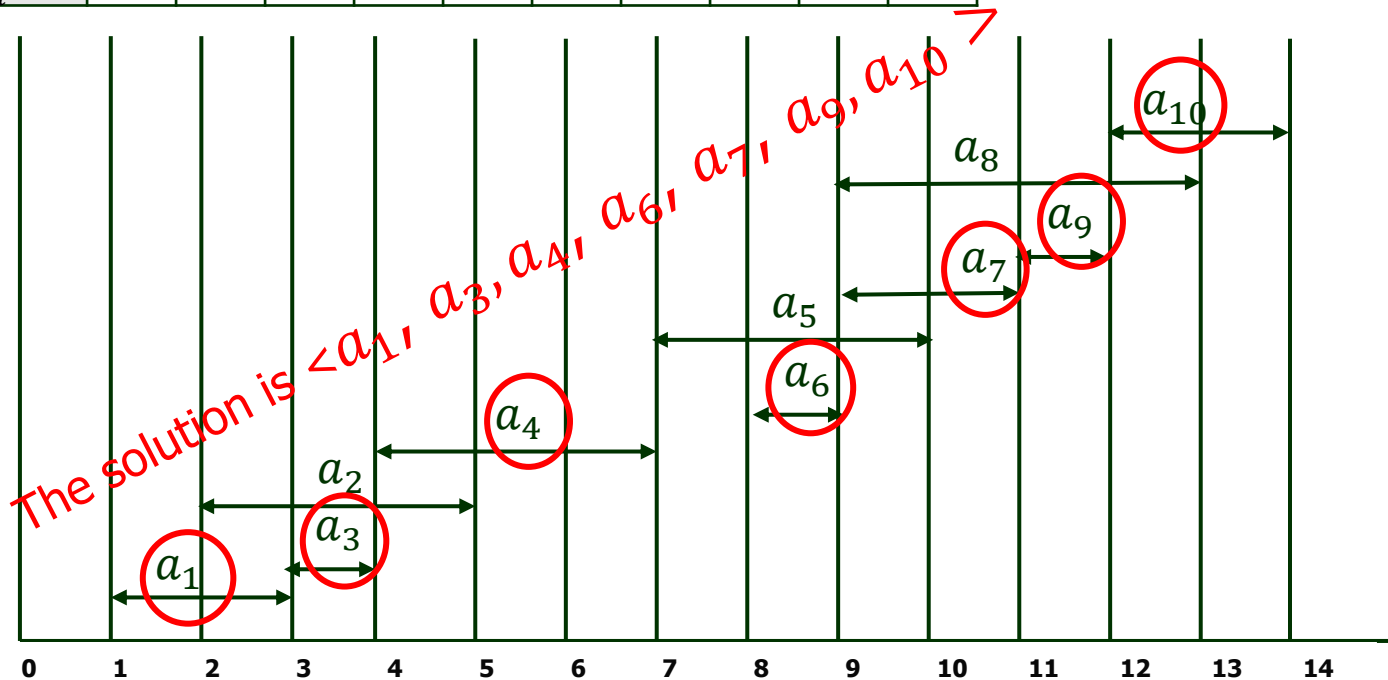


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_1$	$a_3$	$a_2$	$a_4$	$a_6$	$a_5$	$a_7$	$a_9$	$a_8$	$a_{10}$
$s_i$	1	3	2	4	8	7	9	11	9	12
$f_i$	3	4	5	7	9	10	11	12	13	14



# Greedy Algorithm

- **Problem 1: An activity-selection problem**

Example 2: Find the optimal set in the given activity selection problem.

Activity	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$	$a_8$	$a_9$	$a_{10}$
$s_i$	1	2	3	4	7	8	9	9	11	12
$f_i$	5	3	4	6	7	8	11	10	12	13



# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

First arrainging the following activities in increasing order on their finishing

Activity	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$	$a_8$	$a_9$	$a_{10}$
$s_i$	1	2	3	4	7	8	9	9	11	12
$f_i$	5	3	4	6	7	8	11	10	12	13



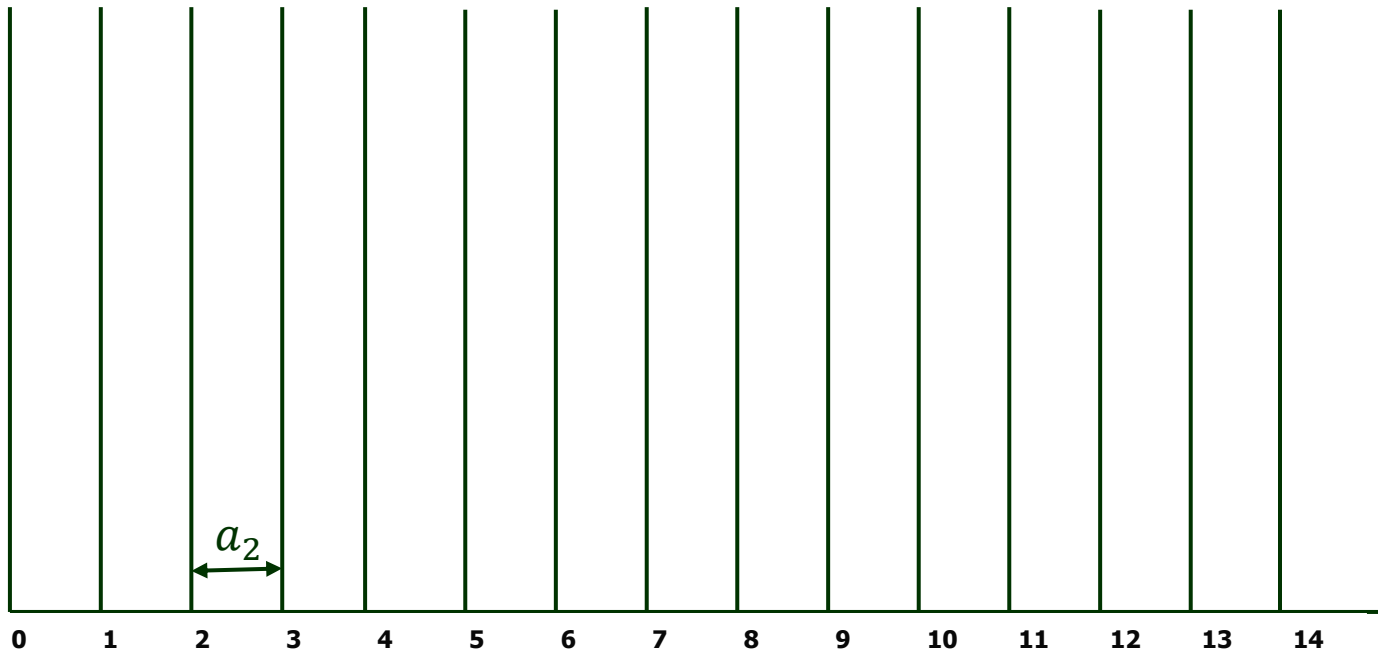
Activity	$a_2$	$a_3$	$a_1$	$a_4$	$a_5$	$a_6$	$a_8$	$a_7$	$a_9$	$a_{10}$
$s_i$	2	3	1	4	5	6	8	7	9	10
$f_i$	3	4	5	6	7	8	10	11	12	13

# Greedy Algorithm

- **Problem 1: An activity-selection problem**

Solution:

Activity	$a_2$	$a_3$	$a_1$	$a_4$	$a_5$	$a_6$	$a_8$	$a_7$	$a_9$	$a_{10}$
$s_i$	2	3	1	4	5	6	8	7	9	10
$f_i$	3	4	5	6	7	8	10	11	12	13

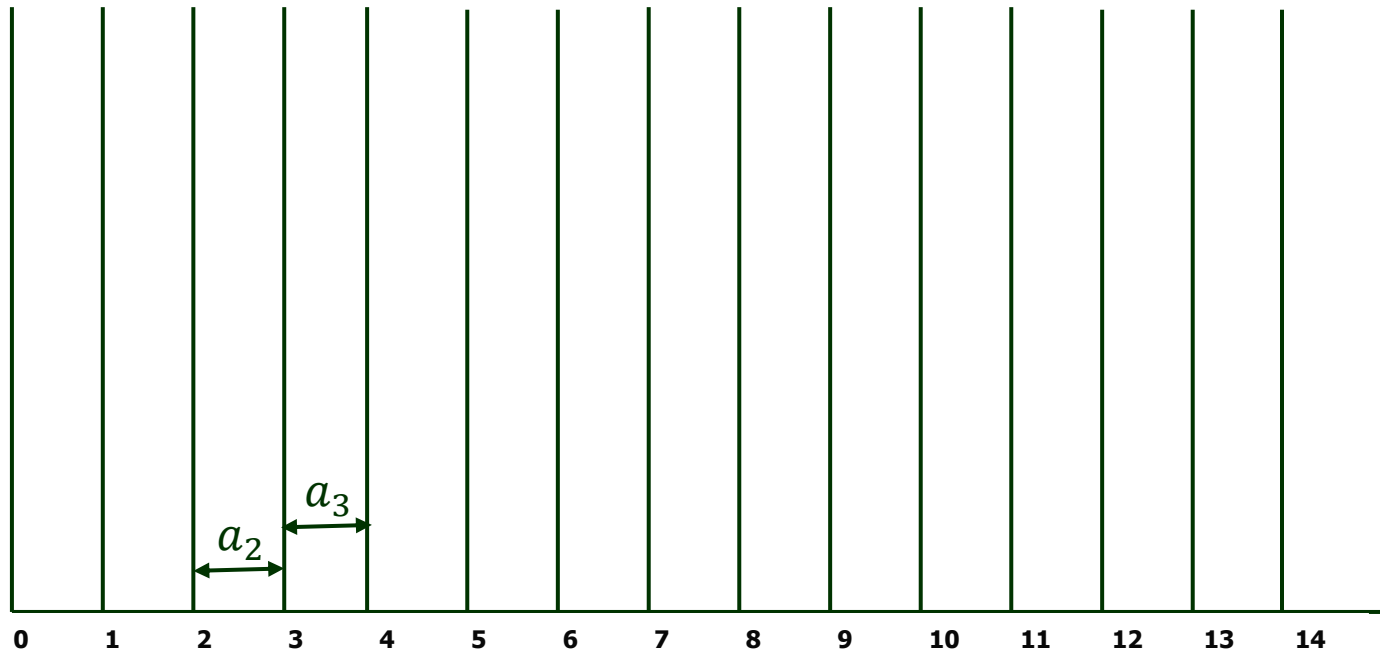


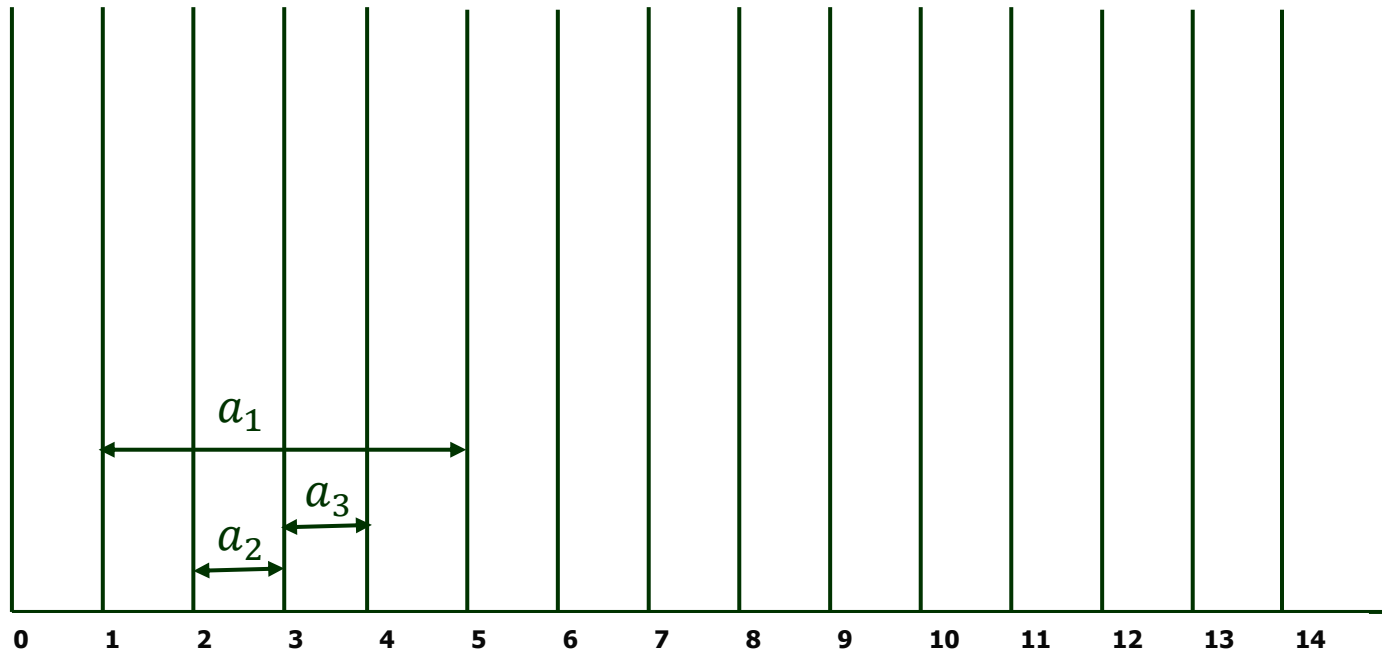
# Greedy Algorithm

- **Problem 1: An activity-selection problem**

Solution:

Activity	$a_2$	$a_3$	$a_1$	$a_4$	$a_5$	$a_6$	$a_8$	$a_7$	$a_9$	$a_{10}$
$s_i$	2	3	1	4	5	6	8	7	9	10
$f_i$	3	4	5	6	7	8	10	11	12	13



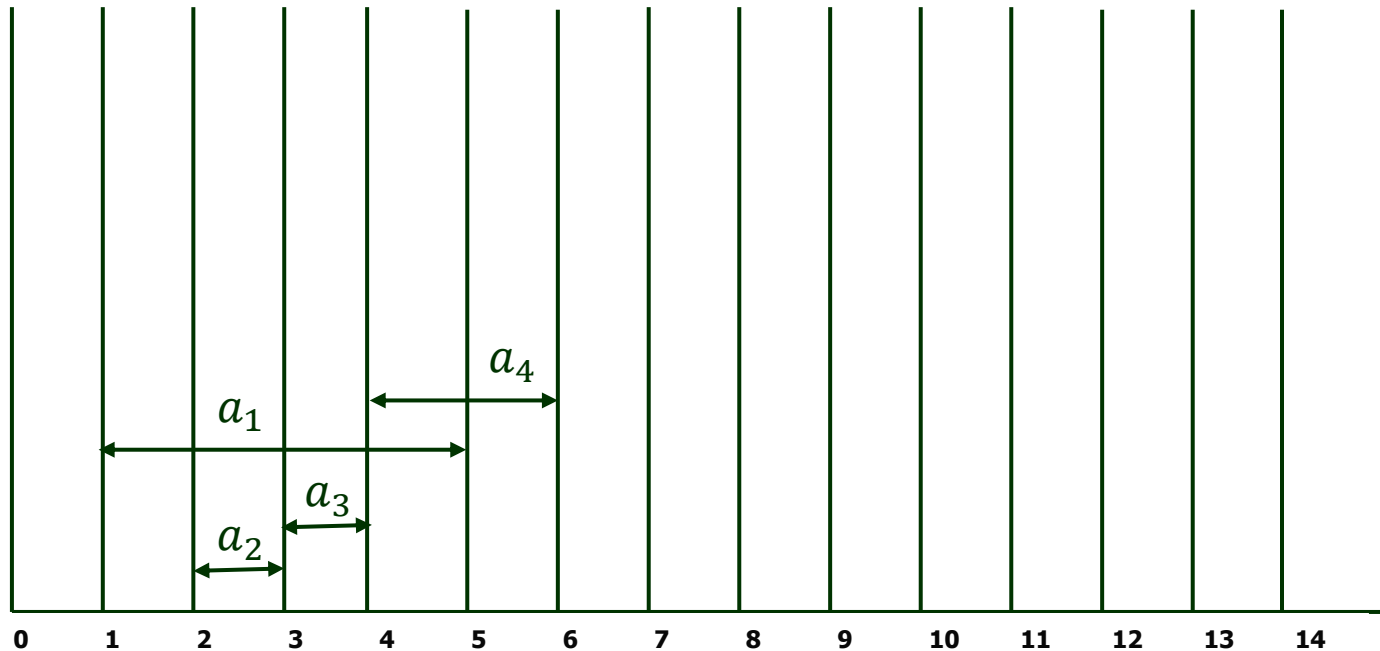


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_2$	$a_3$	$a_1$	$a_4$	$a_5$	$a_6$	$a_8$	$a_7$	$a_9$	$a_{10}$
$s_i$	2	3	1	4	5	6	8	7	9	10
$f_i$	3	4	5	6	7	8	10	11	12	13

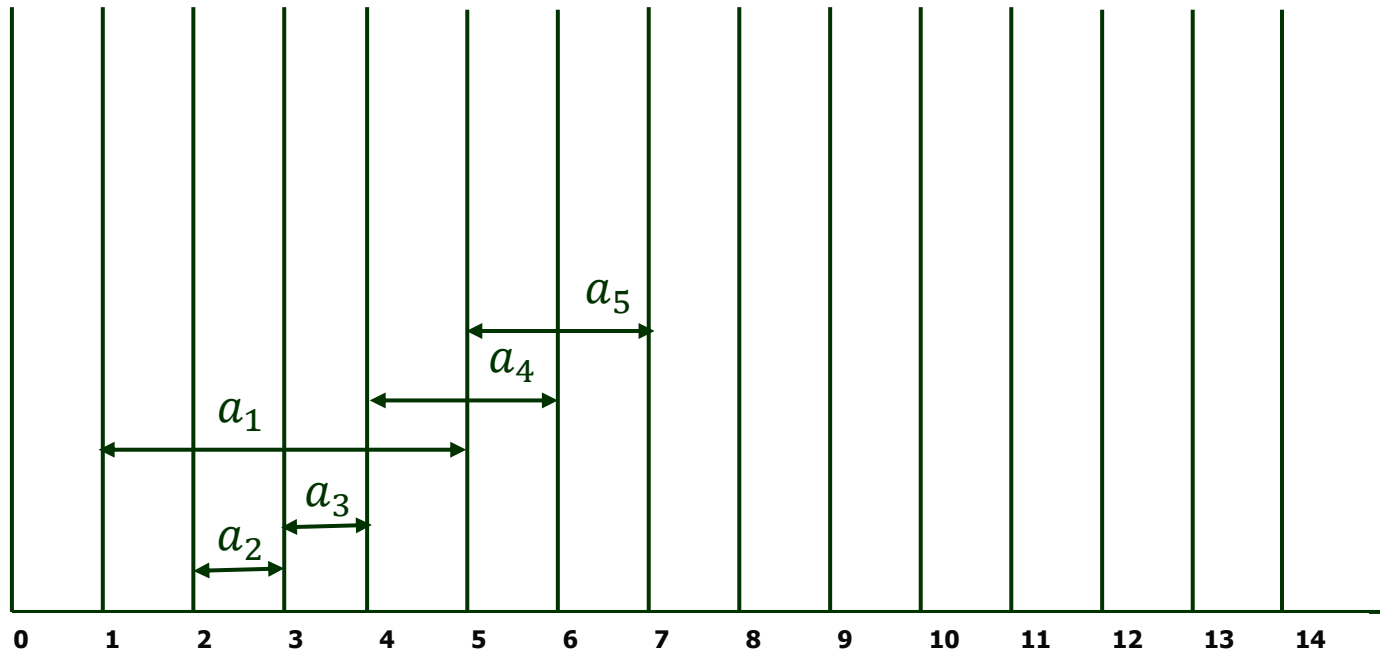


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_2$	$a_3$	$a_1$	$a_4$	$a_5$	$a_6$	$a_8$	$a_7$	$a_9$	$a_{10}$
$s_i$	2	3	1	4	5	6	8	7	9	10
$f_i$	3	4	5	6	7	8	10	11	12	13

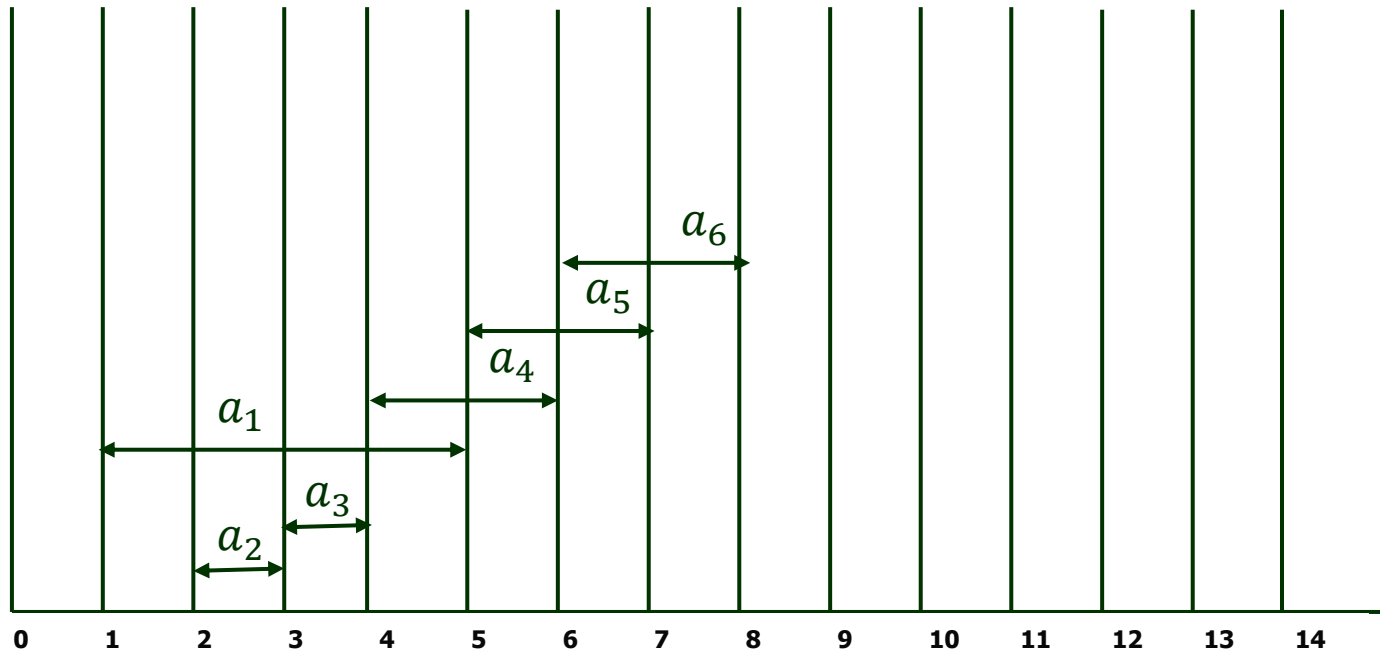


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_2$	$a_3$	$a_1$	$a_4$	$a_5$	$a_6$	$a_8$	$a_7$	$a_9$	$a_{10}$
$s_i$	2	3	1	4	5	6	8	7	9	10
$f_i$	3	4	5	6	7	8	10	11	12	13

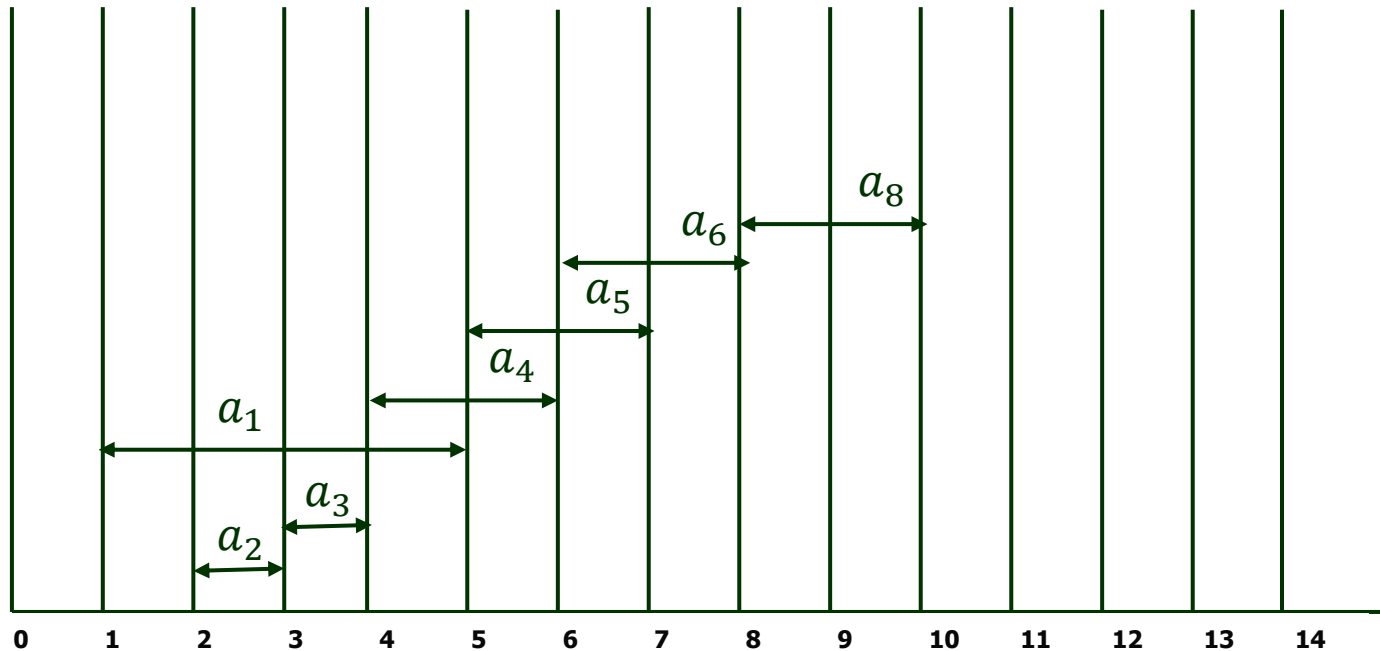


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_2$	$a_3$	$a_1$	$a_4$	$a_5$	$a_6$	$a_8$	$a_7$	$a_9$	$a_{10}$
$s_i$	2	3	1	4	5	6	8	7	9	10
$f_i$	3	4	5	6	7	8	10	11	12	13



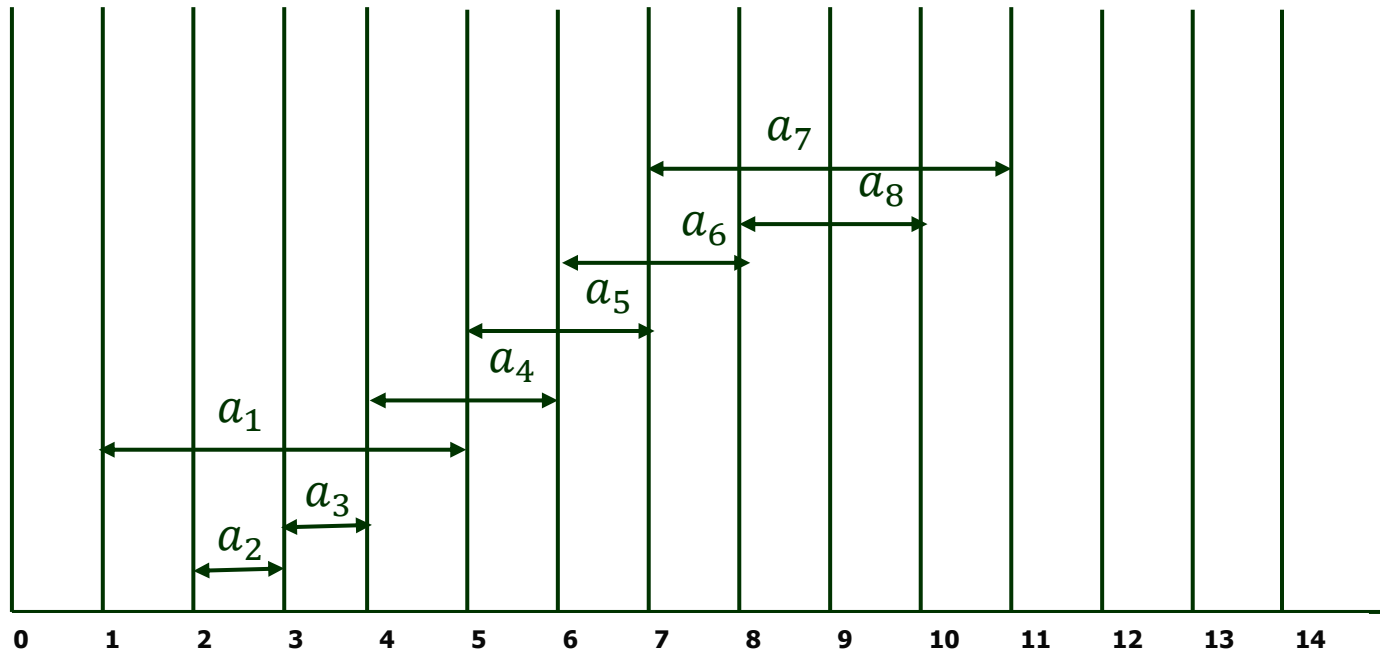


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_2$	$a_3$	$a_1$	$a_4$	$a_5$	$a_6$	$a_8$	$a_7$	$a_9$	$a_{10}$
$s_i$	2	3	1	4	5	6	8	7	9	10
$f_i$	3	4	5	6	7	8	10	11	12	13

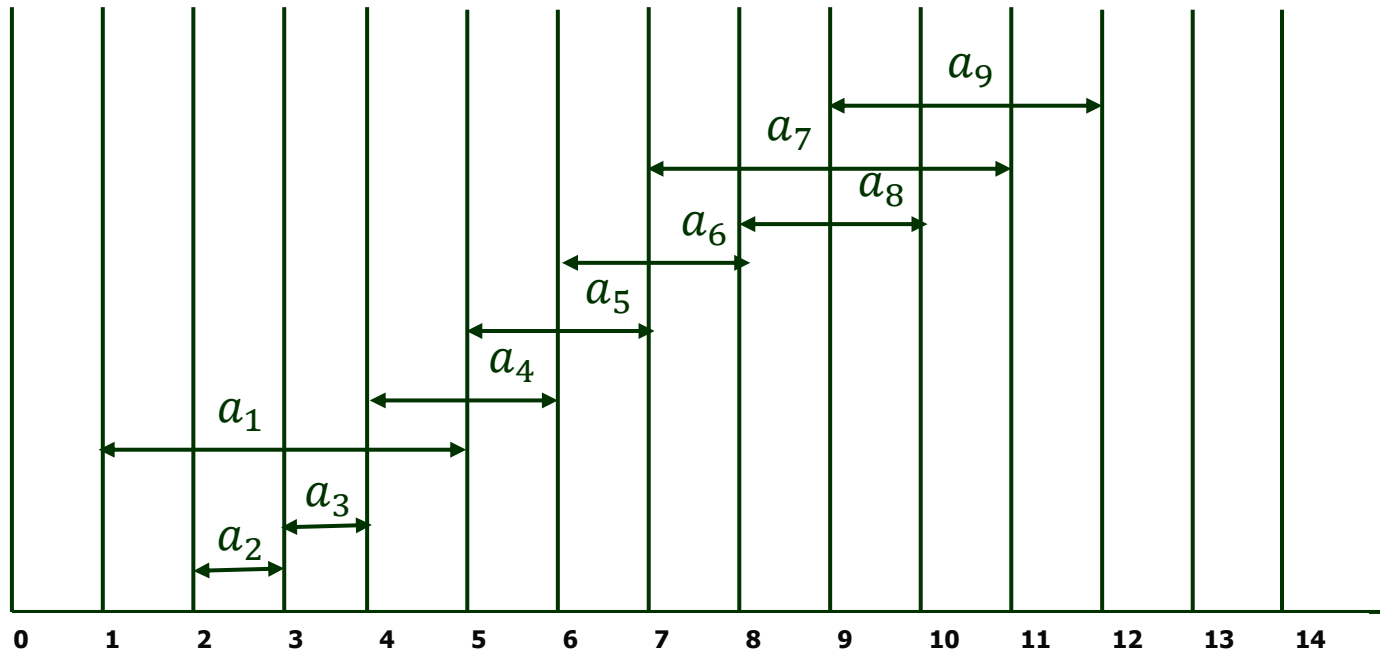


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_2$	$a_3$	$a_1$	$a_4$	$a_5$	$a_6$	$a_8$	$a_7$	$a_9$	$a_{10}$
$s_i$	2	3	1	4	5	6	8	7	9	10
$f_i$	3	4	5	6	7	8	10	11	12	13

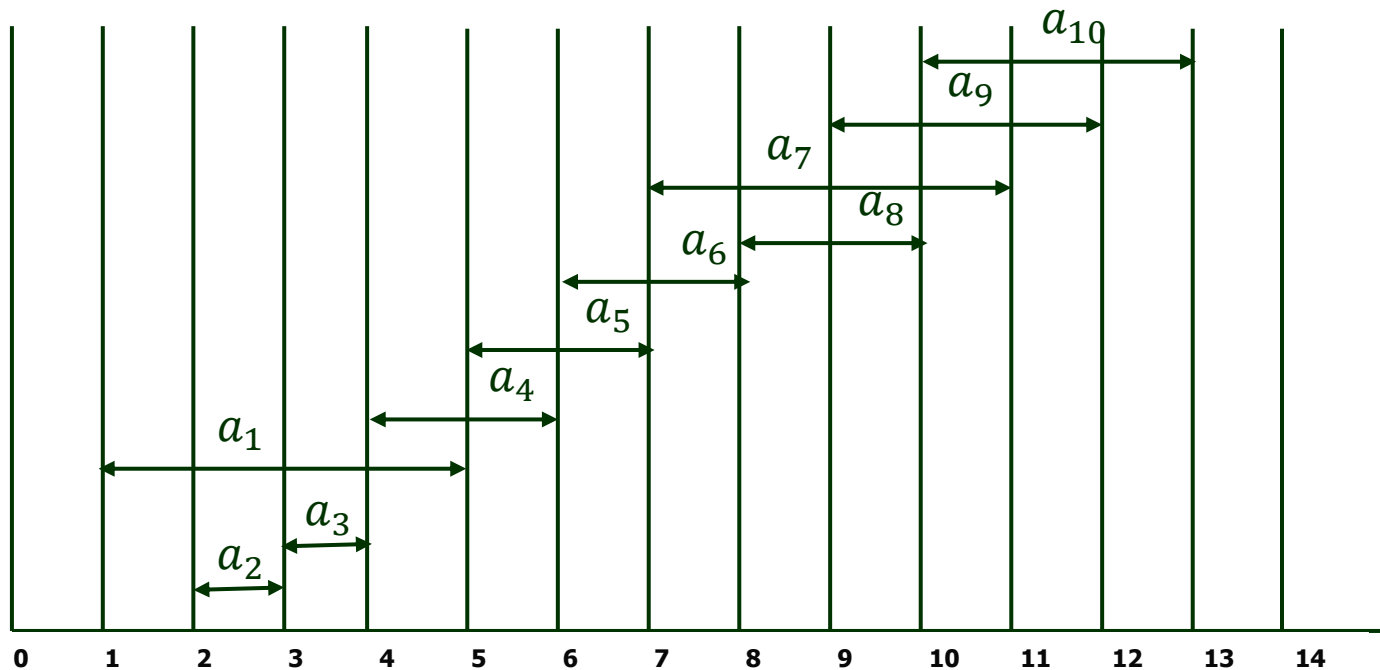


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_2$	$a_3$	$a_1$	$a_4$	$a_5$	$a_6$	$a_8$	$a_7$	$a_9$	$a_{10}$
$s_i$	2	3	1	4	5	6	8	7	9	10
$f_i$	3	4	5	6	7	8	10	11	12	13

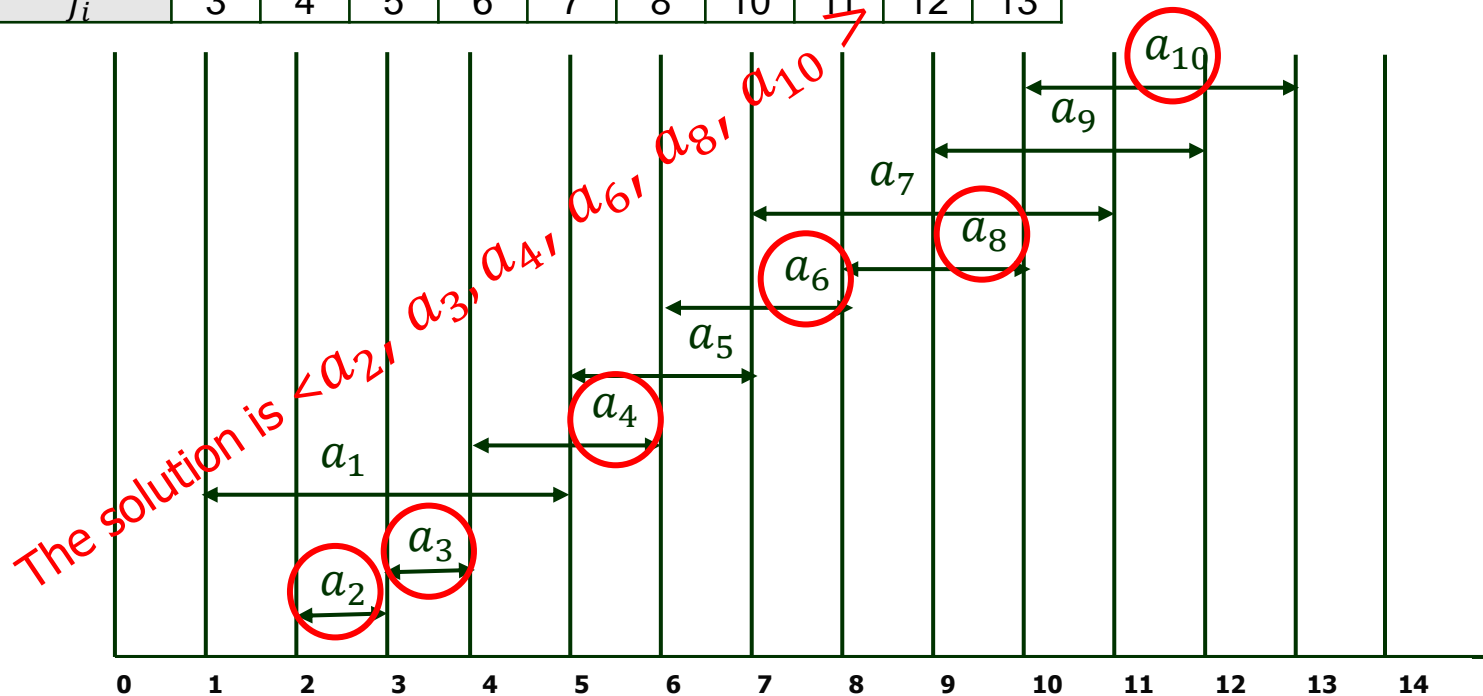


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Activity	$a_2$	$a_3$	$a_1$	$a_4$	$a_5$	$a_6$	$a_8$	$a_7$	$a_9$	$a_{10}$
$s_i$	2	3	1	4	5	6	8	7	9	10
$f_i$	3	4	5	6	7	8	10	11	12	13



# Greedy Algorithm

- **Problem 1: An activity-selection problem**

A recursive greedy algorithm

Recursive-Activity-Selector( $s, f, k, n$ )

1.  $m = k + 1$
2. while  $(m \leq n) \text{ and } (s[m] < f[k])$
3.    $m = m + 1$
4. If  $(m \leq n)$
5.   return  $\{a_m\} \cup \text{Recursive-Activity-Selector}(s, f, k, n)$
6. else return 0

# Greedy Algorithm

- **Problem 1: An activity-selection problem**

A recursive greedy algorithm

Recursive-Activity-Selector( $s, f, k, n$ )

1.  $m = k + 1$
2. while  $(m \leq n) \text{ and } (s[m] < f[k])$
3.    $m = m + 1$
4. If  $(m \leq n)$
5.   return  $\{a_m\} \cup \text{Recursive-Activity-Selector}(s, f, k, n)$
6. else return 0

Complexity without sorting is  $O(n \lg n) + \Theta(n)$ .

Complexity with sorting is  $\Theta(n)$ .

# Greedy Algorithm

- **Problem 1: An activity-selection problem**

A non recursive greedy algorithm

The procedure GREEDY-ACTIVITY-SELECTOR is an iterative version of the procedure RECURSIVE-ACTIVITY-SELECTOR.

Greedy-Activity-Selector ( $s, f$ )

1.  $n = s.length$
2.  $A = \{a_1\}$
3.  $k = 1$
4. for  $m = 2$  to  $n$
5.     If  $(s[m] \geq f[k])$
6.          $A = A \cup \{a_m\}$
7.          $k = m$
8. Return  $A$

# Greedy Algorithm

- **Problem 1: An activity-selection problem**

A non recursive greedy algorithm

The procedure GREEDY-ACTIVITY-SELECTOR is an iterative version of the procedure RECURSIVE-ACTIVITY-SELECTOR.

Greedy-Activity-Selector ( $s, f$ )

1.  $n = s.length$
2.  $A = \{a_1\}$
3.  $k = 1$
4. for  $m = 2$  to  $n$
5.     If ( $s[m] \geq f[k]$ )
6.          $A = A \cup \{a_m\}$
7.          $k = m$
8. Return  $A$

Like the recursive version, Greedy-Activity-Selector schedules a set of  $n$  activities in  $\Theta(n)$  time,



# Greedy Algorithm

- **Problem 2: Task Scheduling problem**

- An interesting problem that are solving using matroids is the **problem of optimally scheduling unit-time tasks on a single processor**, where each task has a deadline, along with a penalty paid if the task misses its deadline.
- This problem looks complicated, when it was solved in a surprisingly simple manner by casting it as a matroid and using a greedy algorithm.

# Greedy Algorithm

- **Problem 2: Task Scheduling problem**

- A unit-time task is a job, such as a program to be run on a computer, that requires exactly one unit of time to complete.
- Given a finite set  $S$  of unit-time tasks, a schedule for  $S$  is a permutation of  $S$  specifying the order in which to perform these tasks.
- The first task in the schedule begins at time 0 and finishes at time 1, the second task begins at time 1 and finishes at time 2, and so on..

# Greedy Algorithm

- **Problem 2: Task Scheduling problem**

- The problem of scheduling unit-time tasks with deadlines and penalties for a single processor has the following inputs:
  - A set  $S = \{a_1, a_2, a_3, \dots, a_n\}$  of  $n$  unit-time tasks;
  - A set of  $n$  integer deadlines  $d_1, d_2, d_3, \dots, d_n$  such that each  $d_i$  satisfies  $1 \leq d_i \leq n$  and task  $a_i$  is supposed to finish by time  $d_i$ .
  - A set of  $n$  nonnegative weights or penalties  $w_1, w_2, w_3, \dots, w_n$ , such that a penalty of  $w_i$  is incurred if task  $a_i$  is not finished by time  $d_i$ , and incurred no penalty if a task finishes by its deadline.

# Greedy Algorithm

- **Problem 2: Task Scheduling problem**

Example 1: Find an optimal schedule from the following table, where the tasks with penalties(weight) and deadlines are given.

Task	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$d_i$	4	2	4	3	1	4	6
$p_i$	70	60	50	40	30	20	10

# Greedy Algorithm

- **Problem 2: Task Scheduling problem**

Example 1:

Solution: As per the greedy algorithm, first sort the tasks in descending order of their penalties. So that minimum penalties will be charged.

Tasks→	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$d_i$	4	2	4	3	1	4	6
$p_i$	70	60	50	40	30	20	10

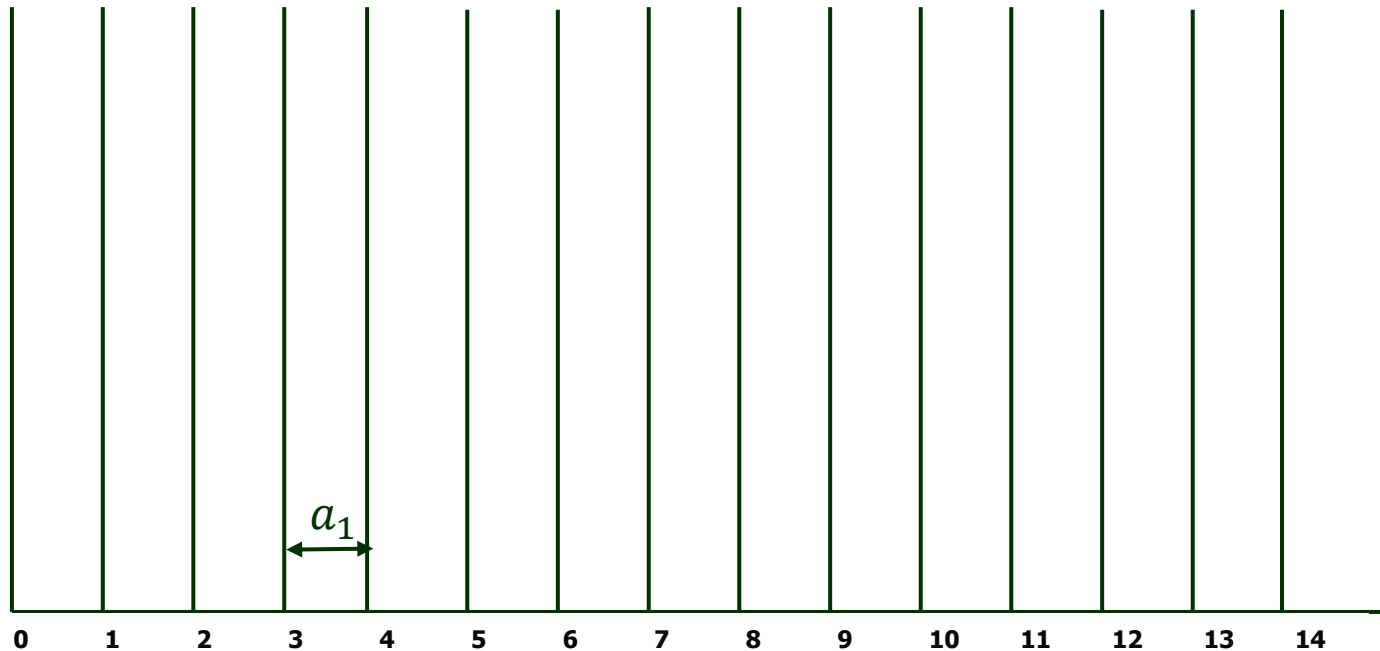
(Note: In this problem the tasks are already sorted)

# Greedy Algorithm

- **Problem 1: An activity-selection problem**

Solution:

Tasks→	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$d_i$	4	2	4	3	1	4	6
$p_i$	70	60	50	40	30	20	10

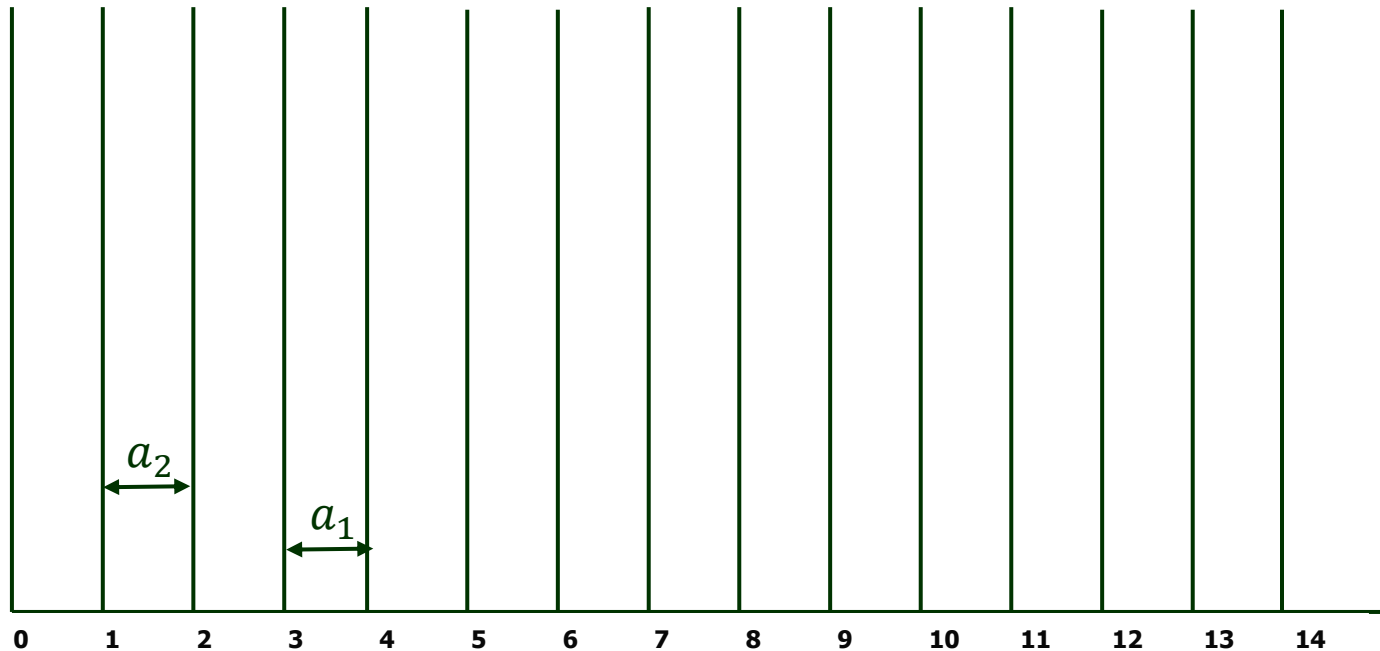


# Greedy Algorithm

- **Problem 1: An activity-selection problem**

Solution:

Tasks→	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$d_i$	4	2	4	3	1	4	6
$p_i$	70	60	50	40	30	20	10

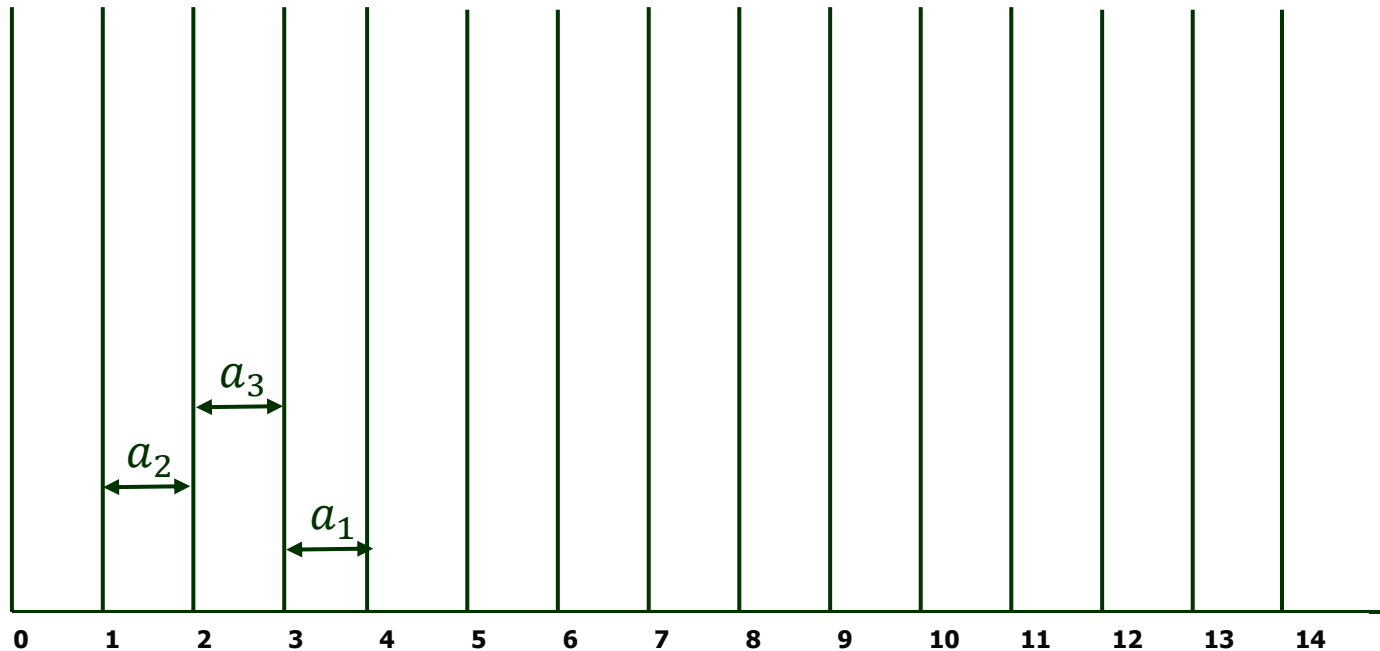


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Tasks→	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$d_i$	4	2	4	3	1	4	6
$p_i$	70	60	50	40	30	20	10



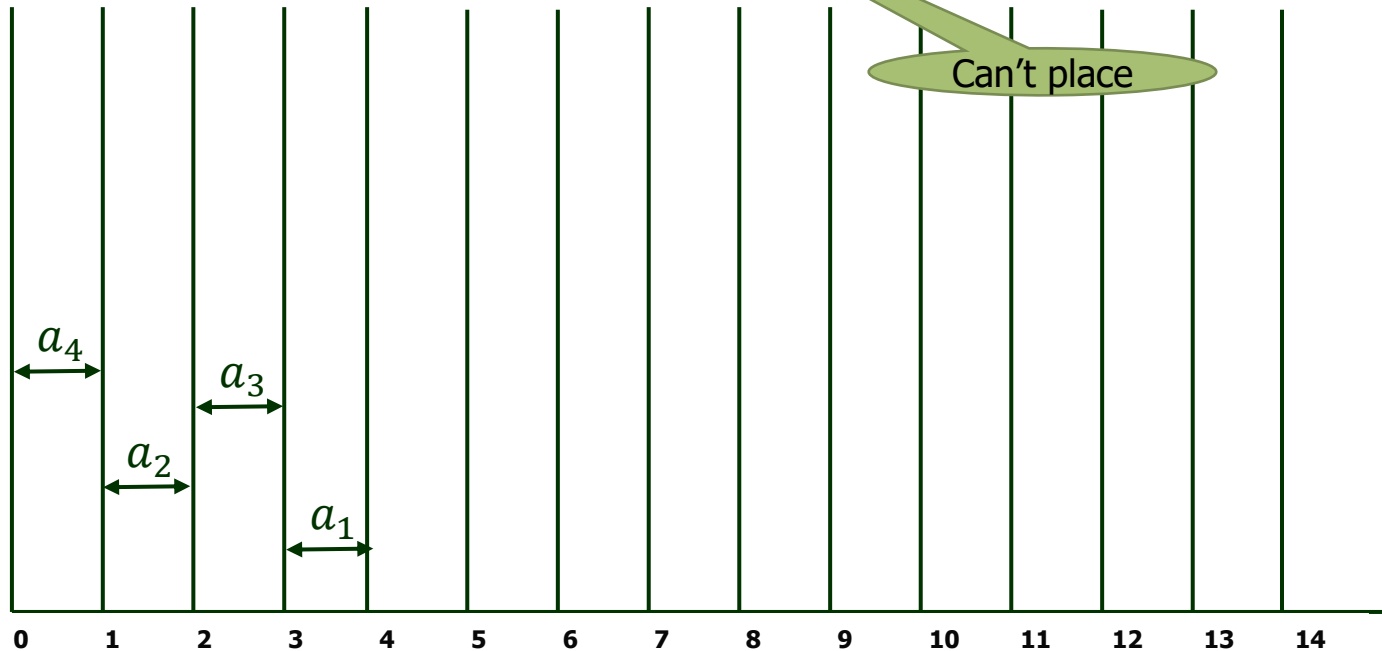


# Greedy Algorithm

- **Problem 1: An activity-selection problem**

Solution:

Tasks→	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$d_i$	4	2	4	3	1	4	6
$p_i$	70	60	50	40	30	20	10

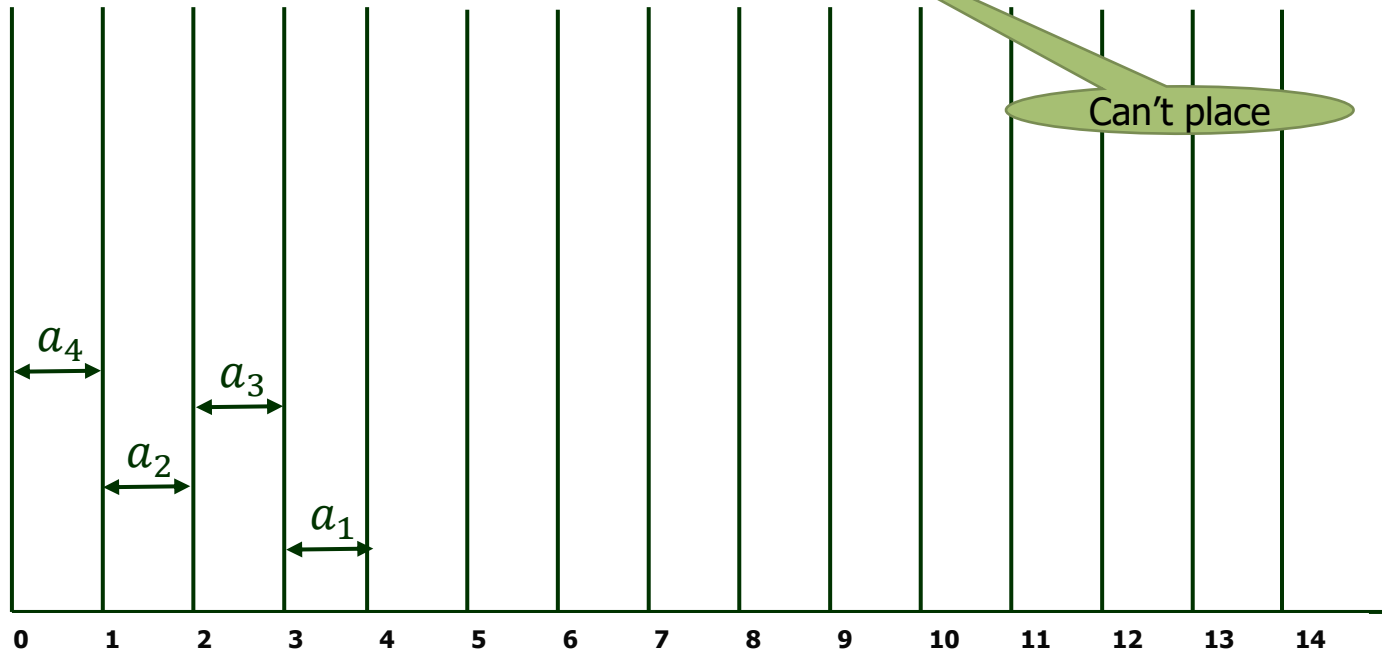


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Tasks→	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$d_i$	4	2	4	3	1	4	6
$p_i$	70	60	50	40	30	20	10

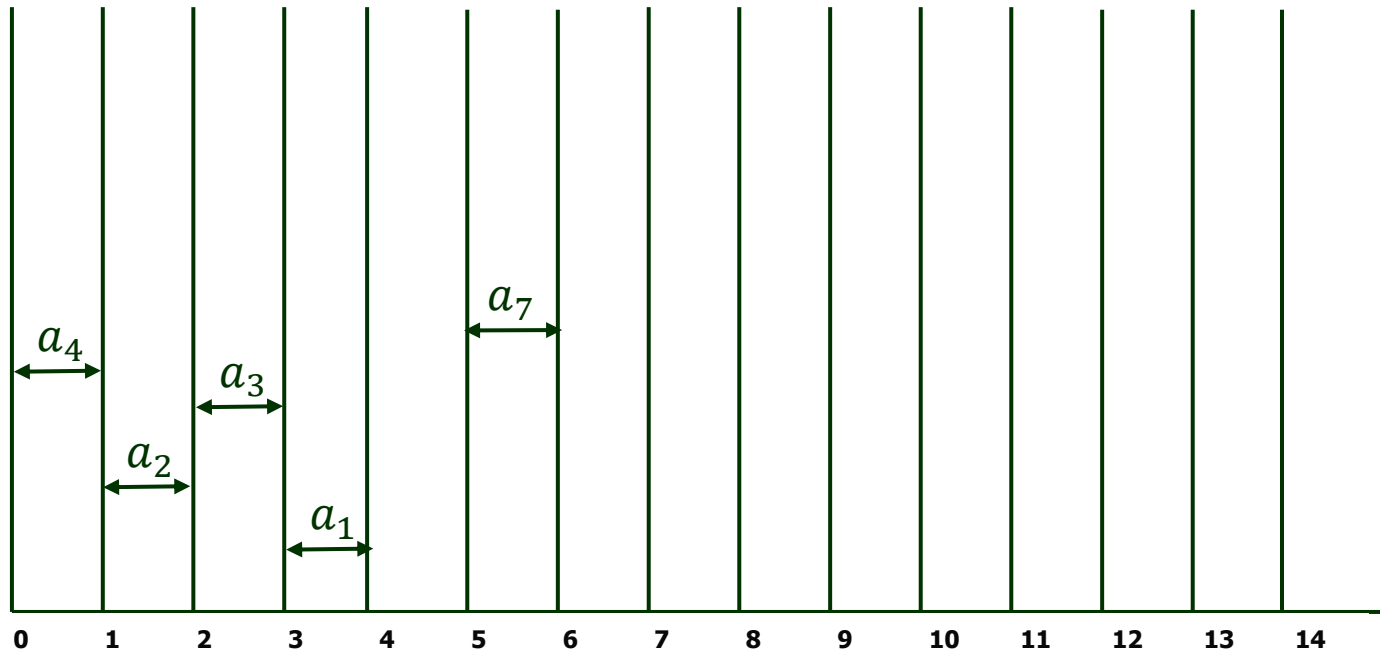


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Tasks→	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$d_i$	4	2	4	3	1	4	6
$p_i$	70	60	50	40	30	20	10

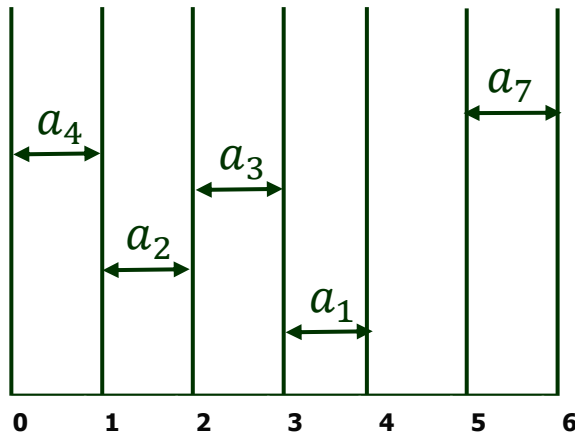


# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Tasks→	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$d_i$	4	2	4	3	1	4	6
$p_i$	70	60	50	40	30	20	10



Gantt Chart

$a_4$	$a_2$	$a_3$	$a_1$		$a_7$
-------	-------	-------	-------	--	-------

The Maximum deadline limit = 6

So total loss is  $p[a_5] + p[a_6] = 30 + 20 = 50$

# Greedy Algorithm

- **Problem 2: Task Scheduling problem**

Example 2: Find an optimal schedule from the following table, where the tasks with penalties(weight) and deadlines are given.

Task	$a_1$	$a_2$	$a_3$	$a_4$
$d_i$	2	1	2	1
$p_i$	100	10	15	27

# Greedy Algorithm


- **Problem 2: Task Scheduling problem**

Example 2:

Solution: As per the greedy algorithm, first sort the tasks in descending order of their penalties. So that minimum penalties will be charged.

Task	$a_1$	$a_2$	$a_3$	$a_4$
$d_i$	2	1	2	1
$p_i$	100	10	15	27

After Sorting



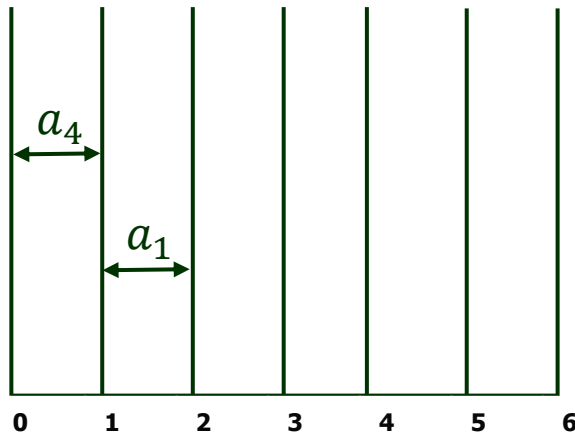
Task	$a_1$	$a_4$	$a_3$	$a_2$
$d_i$	2	1	2	1
$p_i$	100	27	15	10

# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Task	$a_1$	$a_4$	$a_3$	$a_2$
$d_i$	2	1	2	1
$p_i$	100	27	15	10



Gantt Chart

$a_4$	$a_1$
-------	-------

Can't place

The Maximum deadline limit = 2

So total loss is  $p[a_3] + p[a_2] = 15 + 10 = 25$

# Greedy Algorithm

- **Problem 2: Task Scheduling problem**

Example 3: Find an optimal schedule from the following table, where the tasks with penalties(weight) and deadlines are given.

Task	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$d_i$	1	3	3	4	1	2	1
$p_i$	3	5	18	20	6	1	38



# Greedy Algorithm

- **Problem 2: Task Scheduling problem**

Example 3:

Solution: As per the greedy algorithm, first sort the tasks in descending order of their penalties. So that minimum penalties will be charged.

Task	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$d_i$	1	3	3	4	1	2	1
$p_i$	3	5	18	20	6	1	38

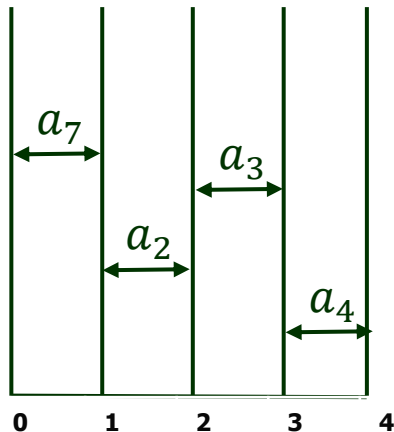
Task	$a_7$	$a_4$	$a_3$	$a_5$	$a_2$	$a_1$	$a_6$
$d_i$	1	4	3	1	3	1	2
$p_i$	38	20	18	6	5	3	1

# Greedy Algorithm

- Problem 1: An activity-selection problem**

Solution:

Task	$a_7$	$a_4$	$a_3$	$a_5$	$a_2$	$a_1$	$a_6$
$d_i$	1	4	3	1	3	1	2
$p_i$	38	20	18	6	5	3	1



Gantt Chart

$a_7$	$a_2$	$a_3$	$a_4$
-------	-------	-------	-------

Can't place

The Maximum deadline limit =4,

So total loss is  $p[a_5] + p[a_1] + p[a_6] = 6 + 3 + 1 = 10$

# Greedy Algorithm

- **Problem 3: Huffman Coding**

- Huffman Coding is a famous Greedy Algorithm.
- It is used for the lossless compression of data.
- It uses variable length encoding.
- It assigns variable length code to all the characters.
- The code length of a character depends on how frequently it occurs in the given text.
- The character which occurs most frequently gets the smallest code.
- The character which occurs least frequently gets the largest code.
- It is also known as Huffman Encoding

# Greedy Algorithm

- **Problem 3: Huffman Coding**

- Huffman Coding implements a rule known as a prefix rule.
- This is to prevent the ambiguities while decoding.
- It ensures that the code assigned to any character is not a prefix of the code assigned to any other character.

# Greedy Algorithm

- **Problem 3: Huffman Coding**

- **Major Steps in Huffman Coding-**

- There are two major steps in Huffman Coding-
    1. Building a Huffman Tree from the input characters.
    2. Assigning code to the characters by traversing the Huffman Tree.

# Greedy Algorithm

- **Problem 3: Huffman Coding**

1. **Building a Huffman Tree from the input characters.**

The steps involved in the construction of Huffman Tree are as follows-

**Step-01:**

- Create a leaf node for each character of the text.
- Leaf node of a character contains the occurring frequency of that character.

**Step-02:**

- Arrange all the nodes in increasing order of their frequency value.

# Greedy Algorithm

- **Problem 3: Huffman Coding**

1. Building a Huffman Tree from the input characters.

Step-03:

- Considering the first two nodes having minimum frequency,
- Create a new internal node.
- The frequency of this new node is the sum of frequency of those two nodes.
- Make the first node as a left child and the other node as a right child of the newly created node.

Step-04:

- Keep repeating **Step-02** and **Step-03** until all the nodes form a single tree.

The tree finally obtained is the desired Huffman Tree.

# Greedy Algorithm

- **Problem 3: Huffman Coding**

1. **Building a Huffman Tree from the input characters.**

Time Complexity-

The time complexity analysis of Huffman Coding is as follows-

- `extractMin( )` is called  $2 \times (n-1)$  times if there are  $n$  nodes.
- As `extractMin( )` calls `minHeapify( )`, it takes  $O(\log n)$  time.

Thus, Overall time complexity of Huffman Coding becomes  $O(n \log n)$ .

*[Note: Here,  $n$  is the number of unique characters in the given text.]*



# Greedy Algorithm

- **Problem 3: Huffman Coding**

- 2. Assigning code to the characters by traversing the Huffman Tree.

- Assign weight to all the edges of the constructed Huffman Tree.
    - Let us assign weight '0' to the left edges and weight '1' to the right edges.

# Greedy Algorithm

- **Problem 3: Huffman Coding**

- 2. Assigning code to the characters by traversing the Huffman Tree.

- Assign weight to all the edges of the constructed Huffman Tree.
    - Let us assign weight '0' to the left edges and weight '1' to the right edges.

- **Rule**

- If you assign weight '0' to the left edges, then assign weight '1' to the right edges.
    - If you assign weight '1' to the left edges, then assign weight '0' to the right edges.
    - Any of the above two conventions may be followed.
    - But follow the same convention at the time of decoding that is adopted at the time of encoding.

# Greedy Algorithm

- **Problem 3: Huffman Coding**

## **Example 1-**

A file contains the following characters with the frequencies as shown. If Huffman Coding is used for data compression, determine-

- Huffman Code for each character
- Average code length
- Length of Huffman encoded message (in bits)

Characters	Frequencies
a	10
e	15
i	12
o	3
u	4
s	13
t	1

# Greedy Algorithm

- **Problem 3: Huffman Coding**

## **Example 1-**

### **Solution**

- First let us construct the Huffman Tree.
- Huffman Tree is constructed in the following steps-

Step 1:



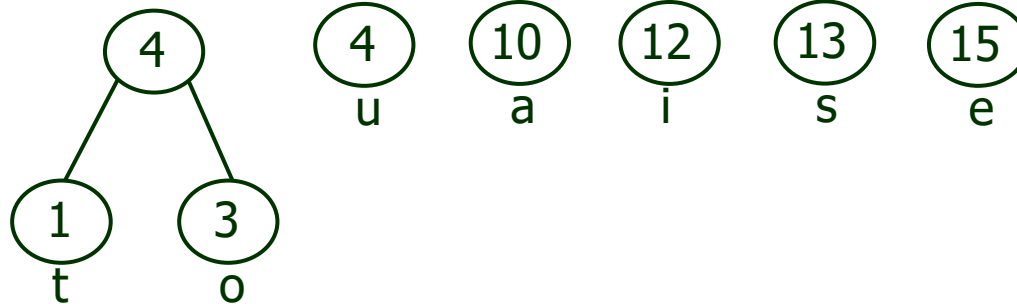
# Greedy Algorithm

- **Problem 3: Huffman Coding**

**Example 1-**

**Solution**

Step 2:



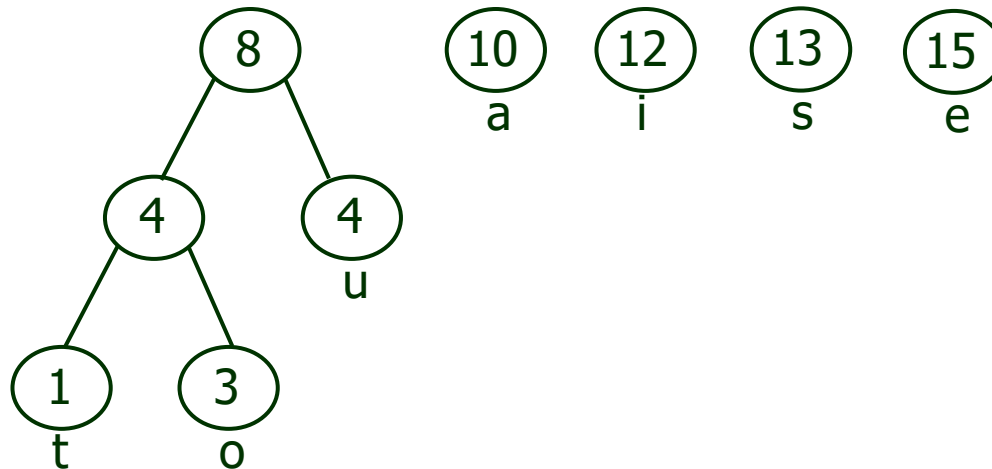
# Greedy Algorithm

- **Problem 3: Huffman Coding**

**Example 1-**

**Solution**

Step 3:

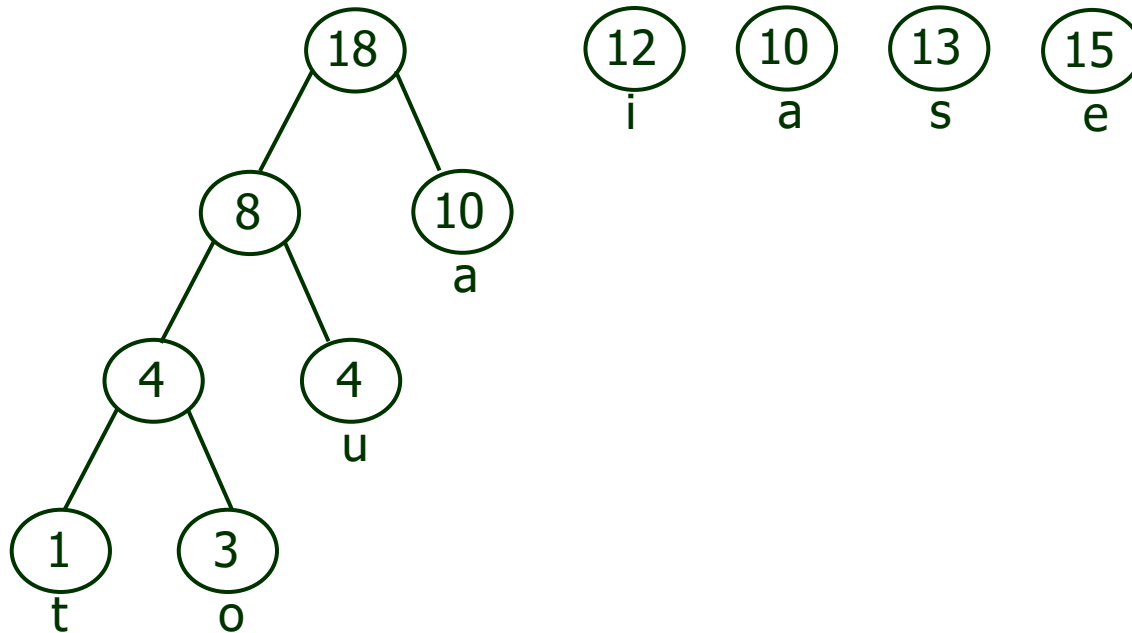


# Greedy Algorithm

- **Problem 3: Huffman Coding**

## **Example 1- Solution**

Step 4:

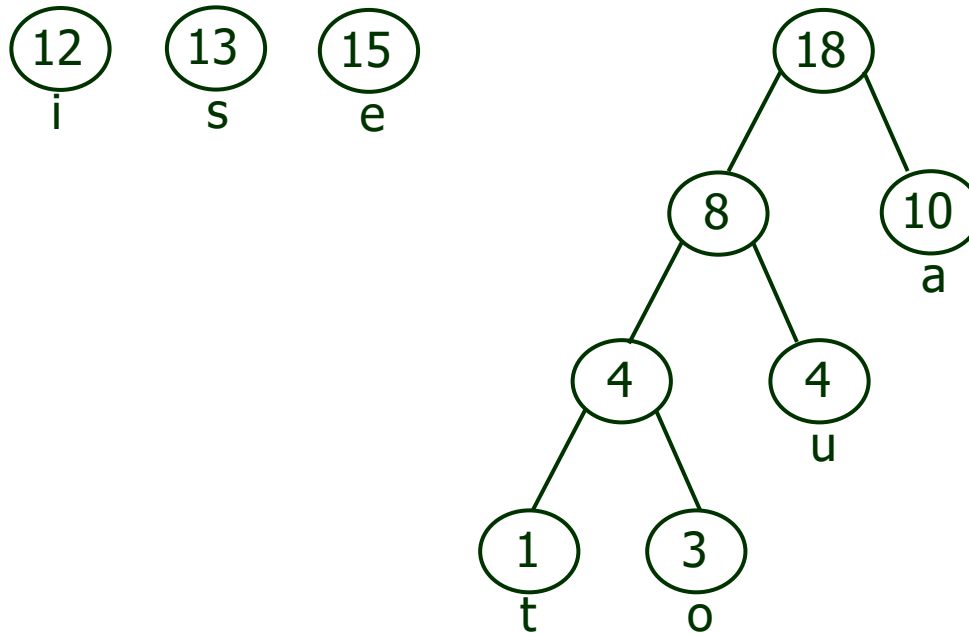


# Greedy Algorithm

- **Problem 3: Huffman Coding**

## **Example 1- Solution**

Step 6:



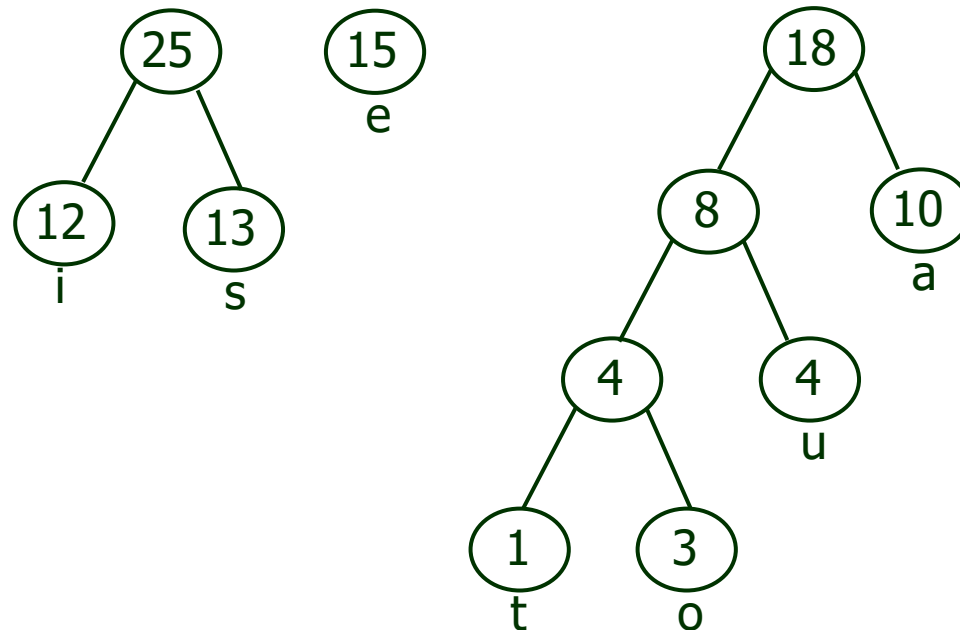


# Greedy Algorithm

- **Problem 3: Huffman Coding**

## **Example 1- Solution**

Step 7:



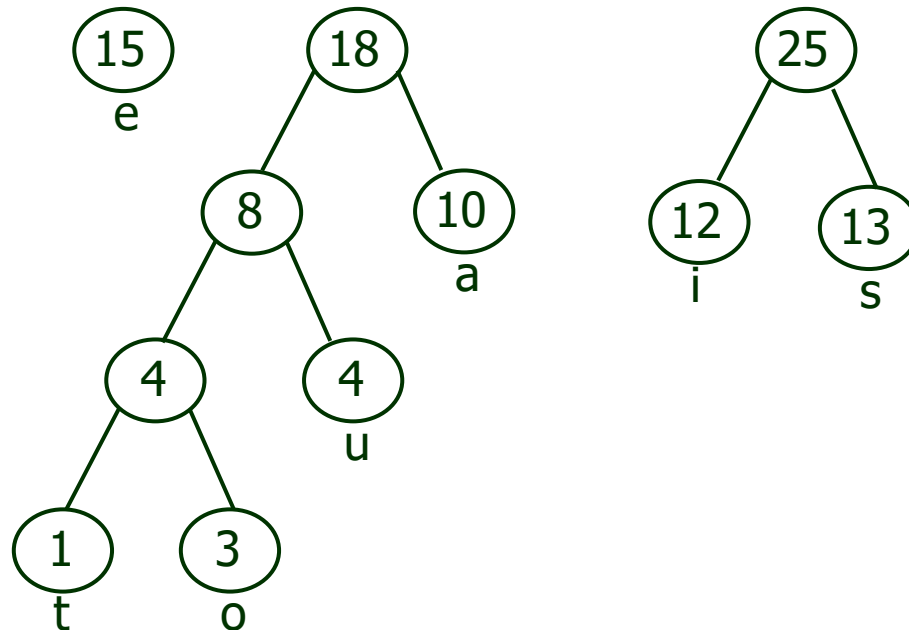
# Greedy Algorithm

- **Problem 3: Huffman Coding**

**Example 1-**

**Solution**

Step 8:



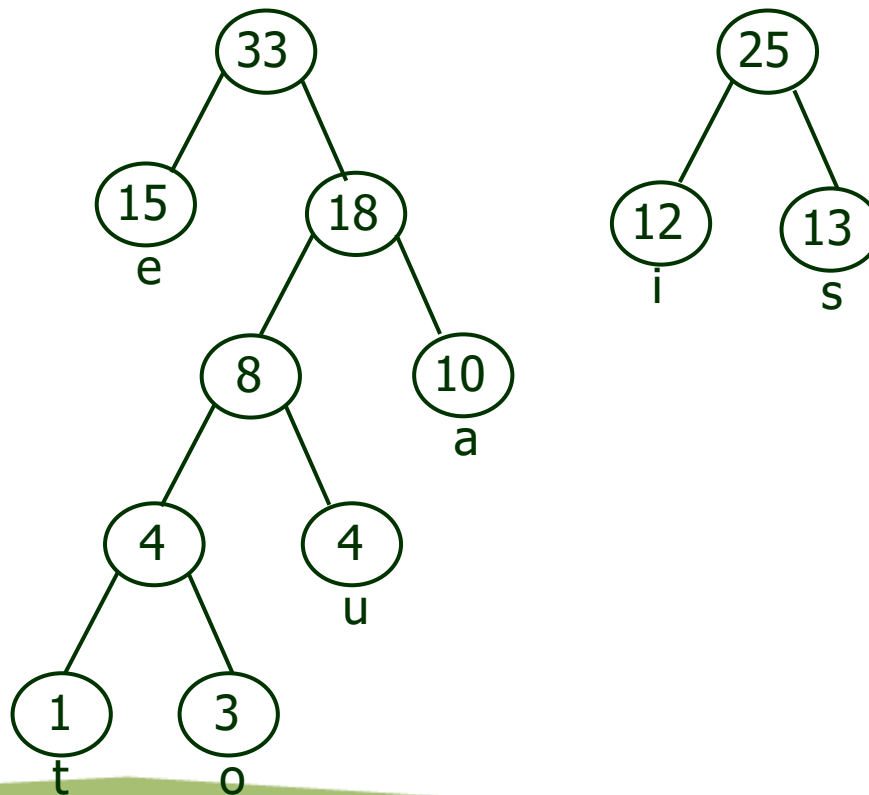
# Greedy Algorithm

- **Problem 3: Huffman Coding**

**Example 1-**

**Solution**

Step 9:



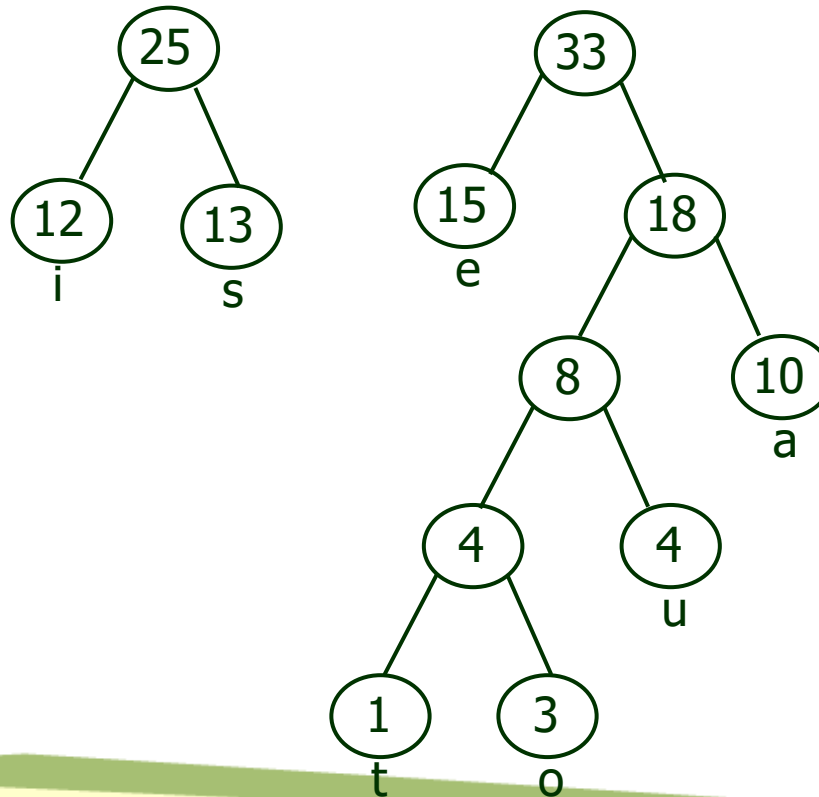
# Greedy Algorithm

- **Problem 3: Huffman Coding**

**Example 1-**

**Solution**

Step 10:



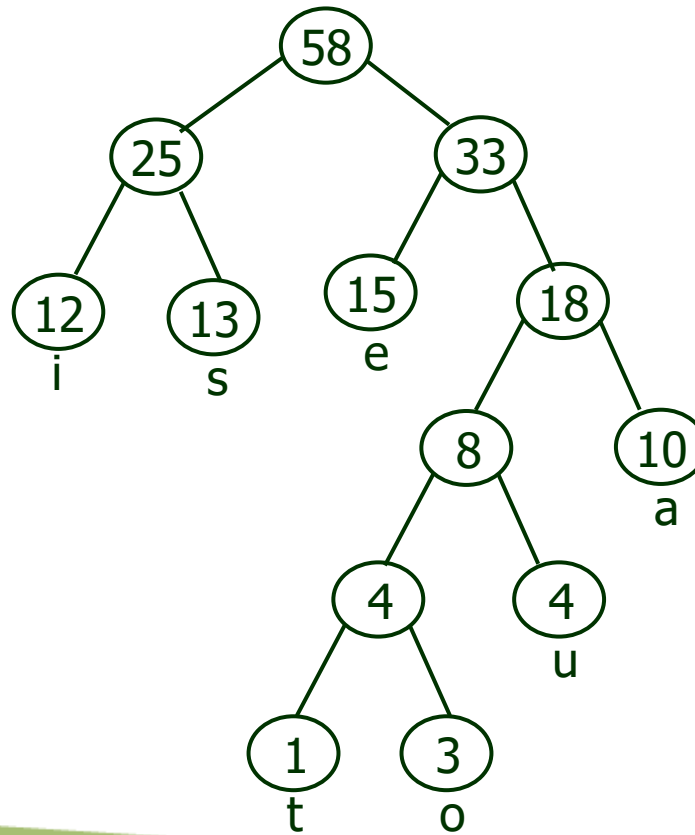
# Greedy Algorithm

- **Problem 3: Huffman Coding**

**Example 1-**

**Solution**

Step 10:



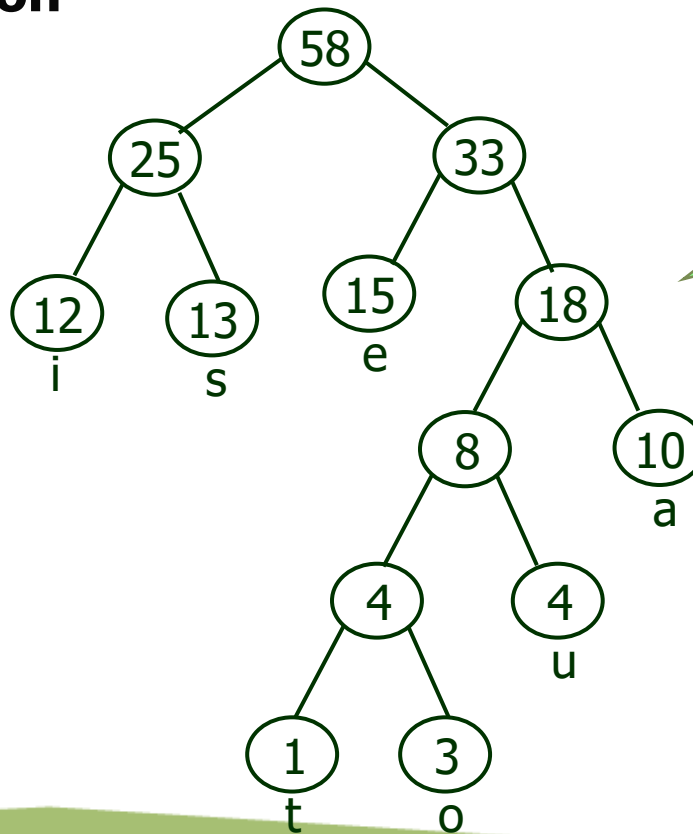
# Greedy Algorithm

- **Problem 3: Huffman Coding**

## Example 1-

## Solution

Step 10:



Now, as per the rule assign weight '0' to the left edges and weight '1' to the right edges.

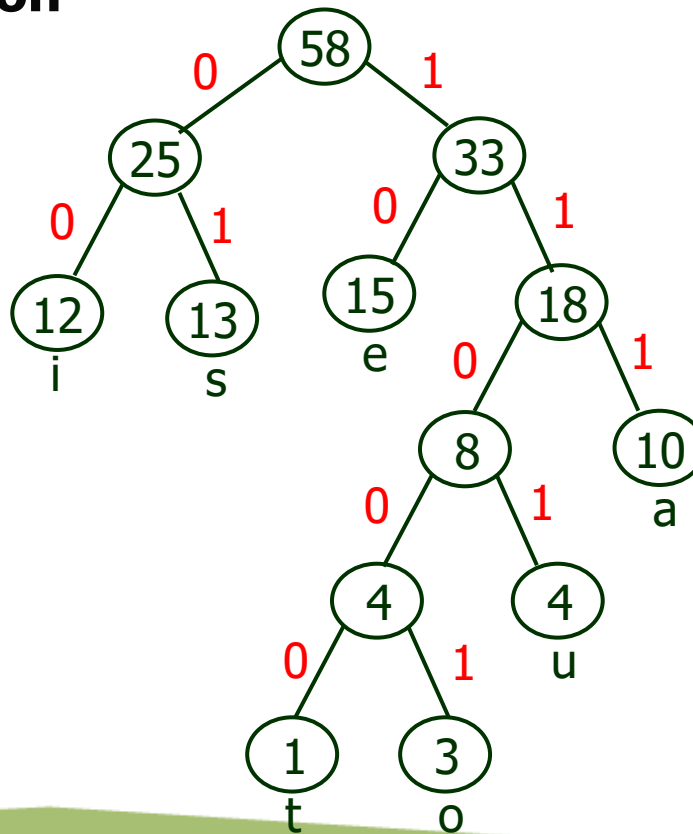
# Greedy Algorithm

- **Problem 3: Huffman Coding**

**Example 1-**

**Solution**

Step 10:



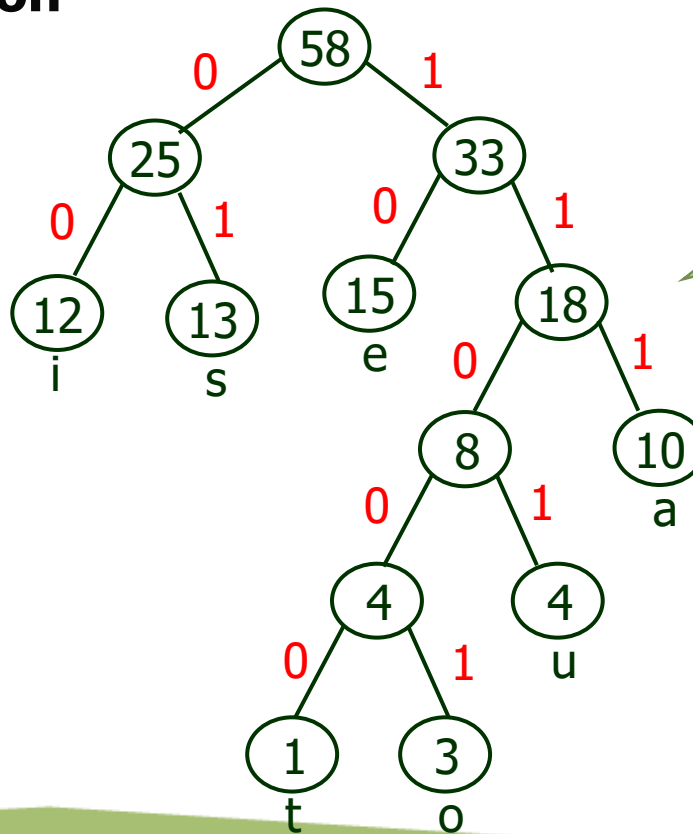
# Greedy Algorithm

- **Problem 3: Huffman Coding**

## Example 1-

## Solution

Step 10:



Now, answer  
all the  
questions

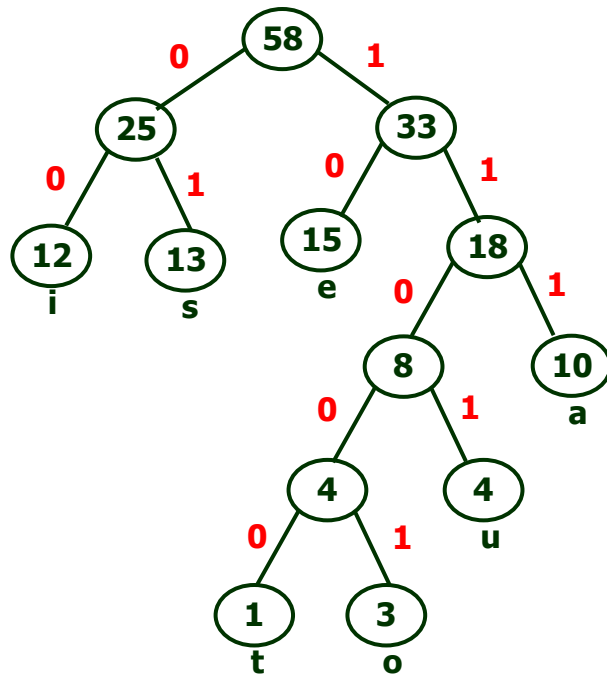


# Greedy Algorithm

- **Problem 3: Huffman Coding**

**Example 1-  
Solution**

Question 1: Huffman Code for each character.



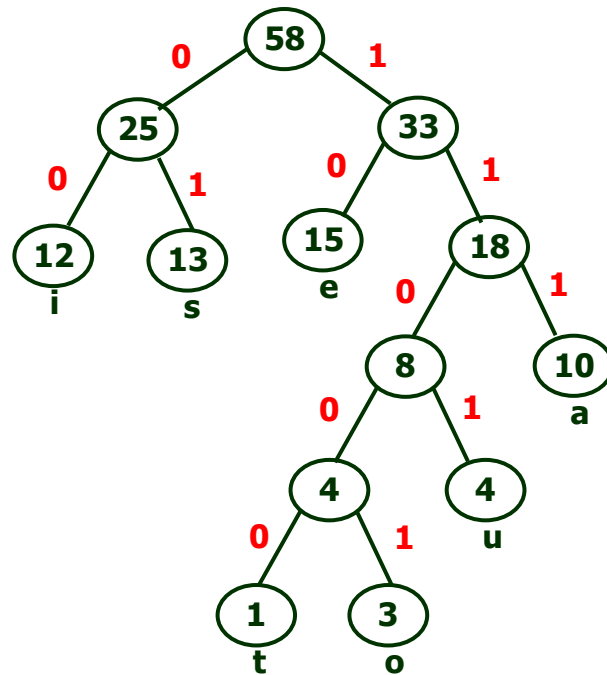
Huffman Tree

Characters	Frequencies	Huffman Code
a	10	111
e	15	10
i	12	00
o	3	11001
u	4	1101
s	13	01
t	1	11000

# Greedy Algorithm

## • Problem 3: Huffman Coding

### Example 1- Solution



Huffman Tree

Question 1: Huffman Code for each character.

Characters	Frequencies	Huffman Code
a	10	111
e	15	10
i	12	00
o	3	11001
u	4	1101
s	13	01
t	1	11000

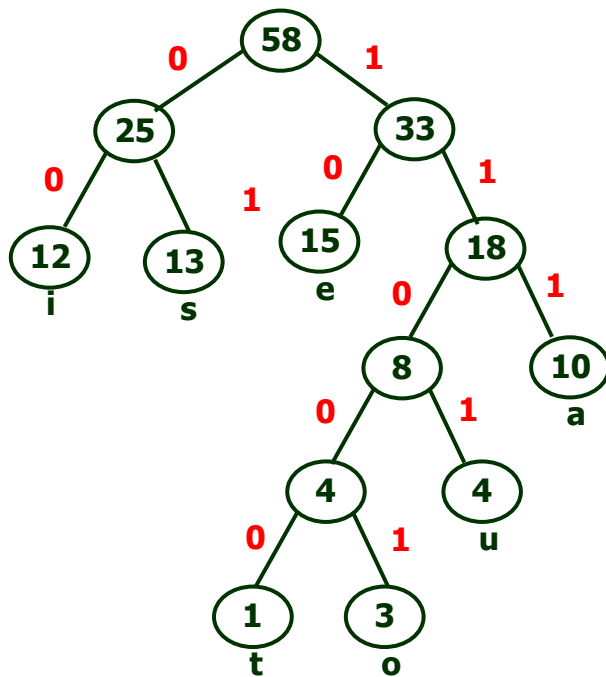
### Observation:

- Characters occurring less frequently in the text are assigned the larger code.
- Characters occurring more frequently in the text are assigned the smaller code.

# Greedy Algorithm

- Problem 3: Huffman Coding

## Example 1- Solution



Huffman Tree

Question 2: Average code length

Characters	Frequencies	Huffman Code
a	10	111
e	15	10
i	12	00
o	3	11001
u	4	1101
s	13	01
t	1	11000

Average code length

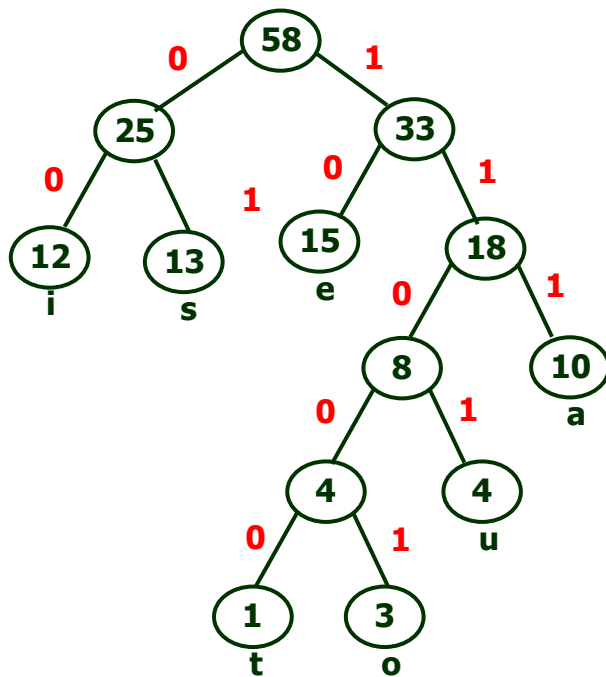
$$\begin{aligned} &= \sum (\text{frequency}_i \times \text{code length}_i) / \sum (\text{frequency}_i) \\ &= \{ (10 \times 3) + (15 \times 2) + (12 \times 2) + (3 \times 5) + \\ &\quad (4 \times 4) + (13 \times 2) + (1 \times 5) \} / (10 + 15 + 12 + \\ &\quad 3 + 4 + 13 + 1) \\ &= 2.52 \end{aligned}$$

# Greedy Algorithm

- **Problem 3: Huffman Coding**

## Example 1-

### Solution



Huffman Tree

Question 3: Length of Huffman encoded message (in bits)

Characters	Frequencies	Huffman Code
a	10	111
e	15	10
i	12	00
o	3	11001
u	4	1101
s	13	01
t	1	11000

Total number of bits in Huffman encoded message

= Total number of characters in the message x Average code length per character

=  $58 \times 2.52$

= 146.16

$\cong 147$  bits

# Greedy Algorithm

- **Problem 3: Huffman Coding (Algorithm)**

HUFFMAN(C)

```
1  n ← |C|
2  Q ← C
3  for i 1 to n - 1
4      do allocate a new node z
5          left[z] ← x ← EXTRACT-MIN (Q)
6          right[z] ← y ← EXTRACT-MIN (Q)
7          f [z] ← f [x] + f [y]
8          INSERT(Q, z)
9  return EXTRACT-MIN(Q)
```

# Greedy Algorithm

- **Problem 3: Huffman Coding (Algorithm)**

HUFFMAN(C)

```
1  n ← |C|
2  Q ← C
3  for i 1 to n - 1
4      do allocate a new node z
5          left[z] ← x ← EXTRACT-MIN (Q)
6          right[z] ← y ← EXTRACT-MIN (Q)
7          f [z] ← f [x] + f [y]
8          INSERT(Q, z)
9  return EXTRACT-MIN(Q)
```

The total running time of HUFFMAN on a set of  $n$  characters is  $O(n \lg n)$ .

# Greedy Algorithm

- **Problem 3: Huffman Coding**

## **Example 2-**

A file contains the following characters with the frequencies as shown. If Huffman Coding is used for data compression, determine-

- Huffman Code for each character
- Average code length
- Length of Huffman encoded message (in bits)

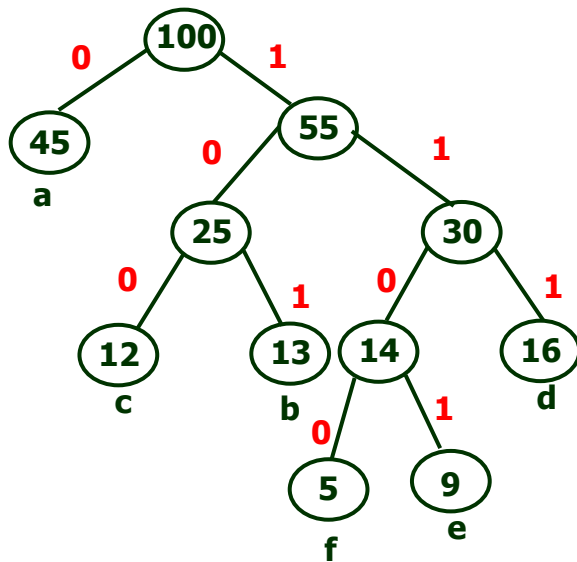
Characters	Frequencies
a	45
b	13
c	12
d	16
e	9
f	5

# Greedy Algorithm

- **Problem 3: Huffman Coding**

## **Example 2- Solution**

Question 1: Huffman Code for each character.



Huffman Tree

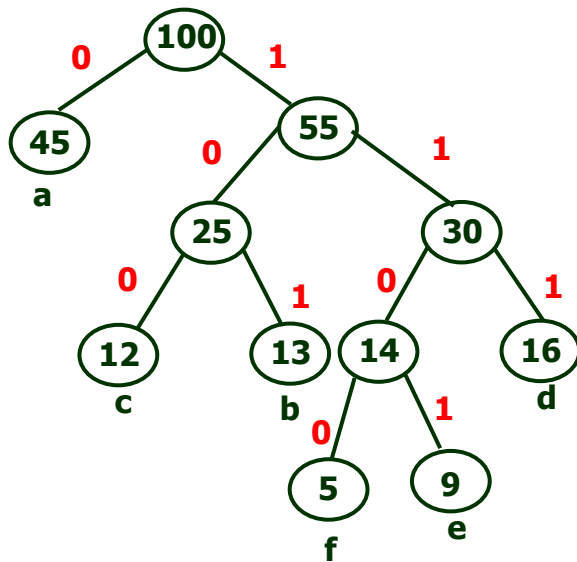
Characters	Frequencies	Huffman Code
a	45	0
b	13	101
c	12	100
d	16	111
e	9	1101
f	5	1100



# Greedy Algorithm

- **Problem 3: Huffman Coding**

**Example 1-  
Solution**



Huffman Tree

Question 1: Huffman Code for each character.

Characters	Frequencies	Huffman Code
a	45	0
b	13	101
c	12	100
d	16	111
e	9	1101
f	5	1100

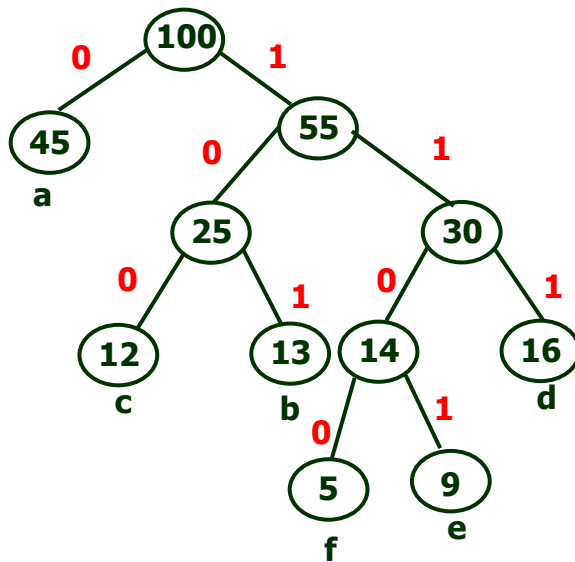
**Observation:**

- Characters occurring less frequently in the text are assigned the larger code.
- Characters occurring more frequently in the text are assigned the smaller code.

# Greedy Algorithm

- **Problem 3: Huffman Coding**

## Example 1- Solution



Huffman Tree

Question 2: Average code length

Characters	Frequencies	Huffman Code
a	45	0
b	13	101
c	12	100
d	16	111
e	9	1101
f	5	1100

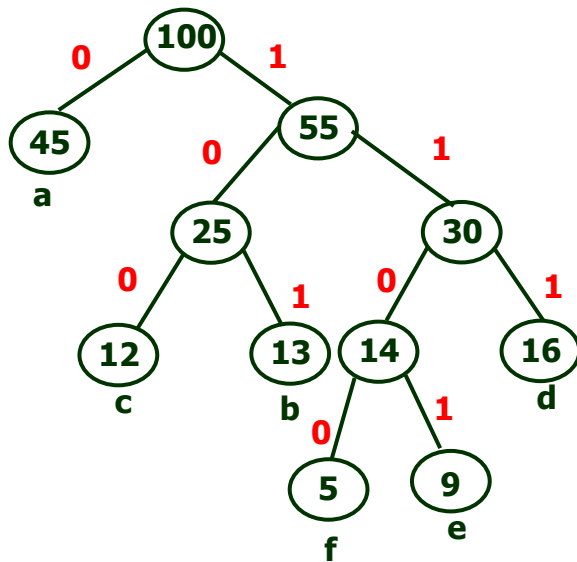
Average code length

$$\begin{aligned} &= \sum (\text{frequency}_i \times \text{code length}_i) / \sum (\text{frequency}_i) \\ &= \{ (45 \times 1) + (13 \times 3) + (12 \times 3) + (16 \times 3) + \\ &\quad (9 \times 4) + (5 \times 4) \} / (45 + 13 + 12 + 16 + 9 + 5) \\ &= 2.24 \end{aligned}$$

# Greedy Algorithm

- **Problem 3: Huffman Coding**

## Example 1- Solution



Huffman Tree

Question 3: Length of Huffman encoded message (in bits)

Characters	Frequencies	Huffman Code
a	45	0
b	13	101
c	12	100
d	16	111
e	9	1101
f	5	1100

Total number of bits in Huffman encoded message

= Total number of characters in the message x Average code length per character

= 100 x 2.24

= 224 bits

# Greedy Algorithm

- **Problem 3: Huffman Coding**

**Example 3- (Practice yourself)**

A file contains the following characters with the frequencies as shown. If Huffman Coding is used for data compression, determine-

- Huffman Code for each character
- Average code length
- Length of Huffman encoded message (in bits)

Characters	a	b	c	d	e	f	g	h
Frequencies	1	1	2	3	5	8	13	21

Thank u