Objectives

- Introduction to Greedy Algorithms
  - Optimization problem
  - Feasible solutions
  - Optimal solution
  - Activity Selection Problem
  - Job Sequencing with deadlines
  - Fractional knapsack problem

Greedy Algorithms:

1. Algorithms for solving optimization problems that is a problem which requires minimum result or maximum result.
2. Go through a sequence of steps with a set of choices at each step. [Algorithms tries to optimize the outcomes based on the sequence of operations]
3. Characteristic of greedy algorithms is that it makes the choice that look best at the moment.
4. It does not guarantee for global optimal solution, but it makes a locally optimal choice with the hope that it will lead to global optimal.

Points related to Greedy Approach:

1. Greedy approach is designed to achieve optimal solution of a given problem.
2. In greedy approach, decisions are made from the given solution domain. In greedy approach we consider the nearest optimal solution and then move onto next step.
3. Most of the problems in greedy contains n-number of inputs and our objective is to finding a subset which satisfy our constraint is called feasible solution.
4. We are required to find a feasible solution that either maximize or minimize a given objective function. A feasible solution that does this is called optimal solution.
5. There can be more than one solution of a given problem. Also there can be more than one feasible solution. But there can be only one optimal solution that is there cannot be more than one optimal solution.

Elements of Greedy strategy:

1. Greedy choice property
2. Optimal substructure

A global optimal solution can be arrived at by making a locally optimal (greedy) choice.

Greedy algorithms make whatever choice seems best at the moment. The choice depends on so far, not depend on any future choice.

Activity Selection Problem:

- A set of n proposed activities $S = \{a_1, a_2, ..., a_n\}$
- Our goal is to select a limited no. of activities that can happen in same place. Goal is to use maximum no. of activities to make use of proper utilization of resource.
- Select the maximum size subset of mutually compatible activities.
- Each activity a_i has start time s_i and finish time f_i.
- Activities are sorted according to increasing order of finish time.
- If a_i is selected, it takes place during the time interval $[s_i, f_i)$.

Activity Selection Problem: schedule several competing activities that require exclusive use of a common resource, with a goal of selecting maximum number of mutual compatible activities.

Compatible activities:

Activities a_i and a_j are compatible if their intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

Possible approaches:

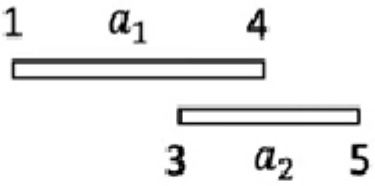
1. Smallest activity first
2. Activity that overlaps with smallest number of other activities.
3. Arrange the activities according to earliest finish time.

Steps of correct solution are:

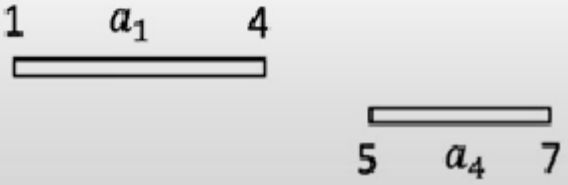
- Pick activity that ends first
- Eliminate activities that overlap with the chosen activity
- Repeat until no more activities left.

Activity a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Start time s_i	1	3	0	5	3	5	6	8	8	2	12
Finish time f_i	4	5	6	7	9	9	10	11	12	14	16

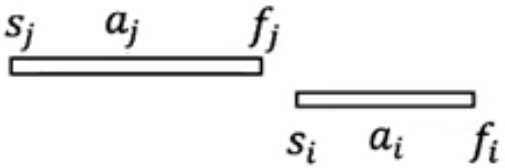
- E.g., a_1 and a_2 are *incompatible*



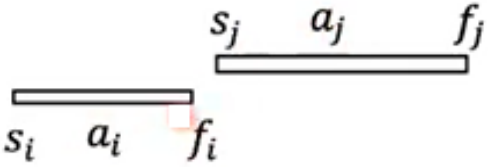
- E.g., a_1 and a_4 are *compatible*

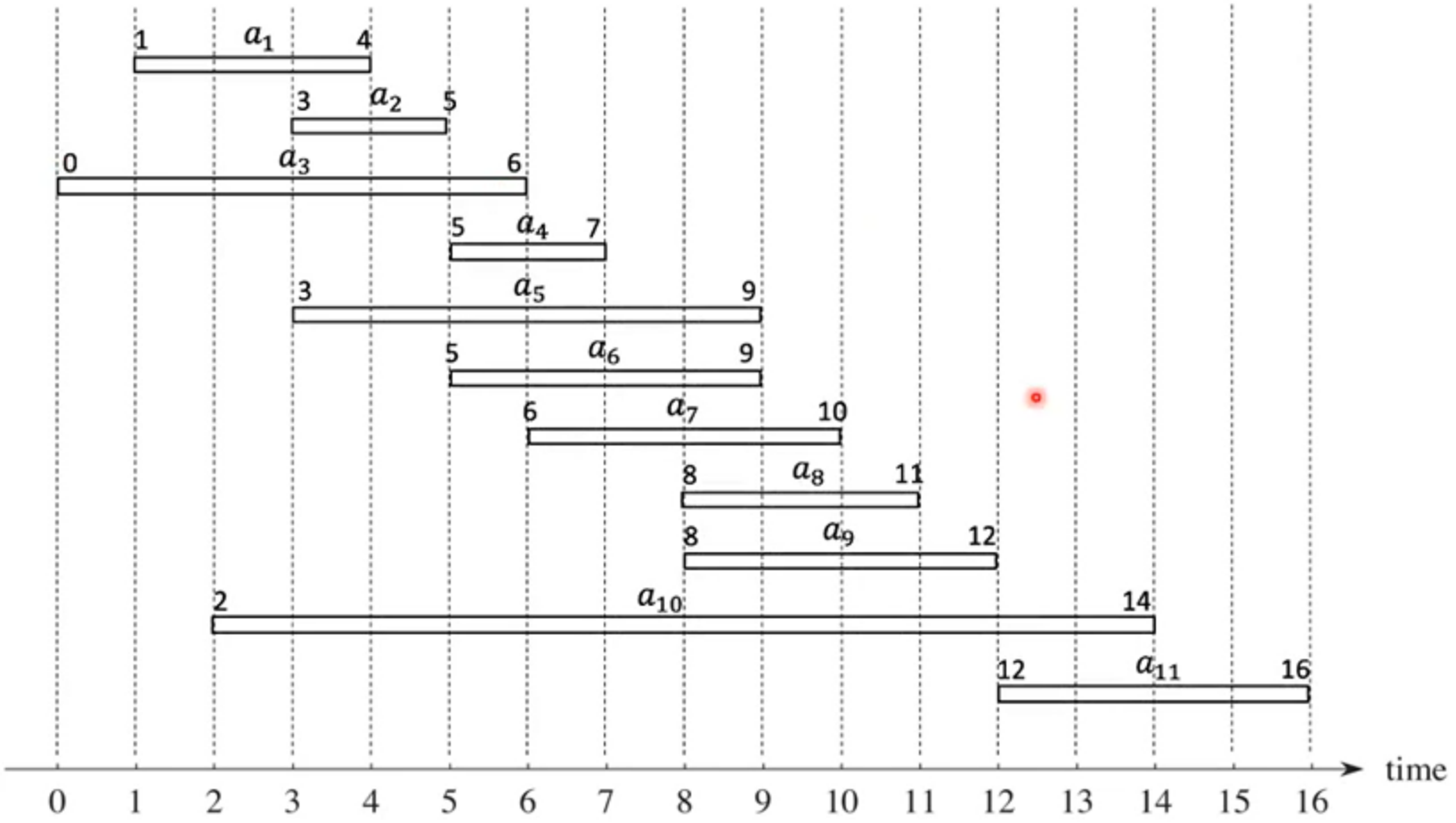


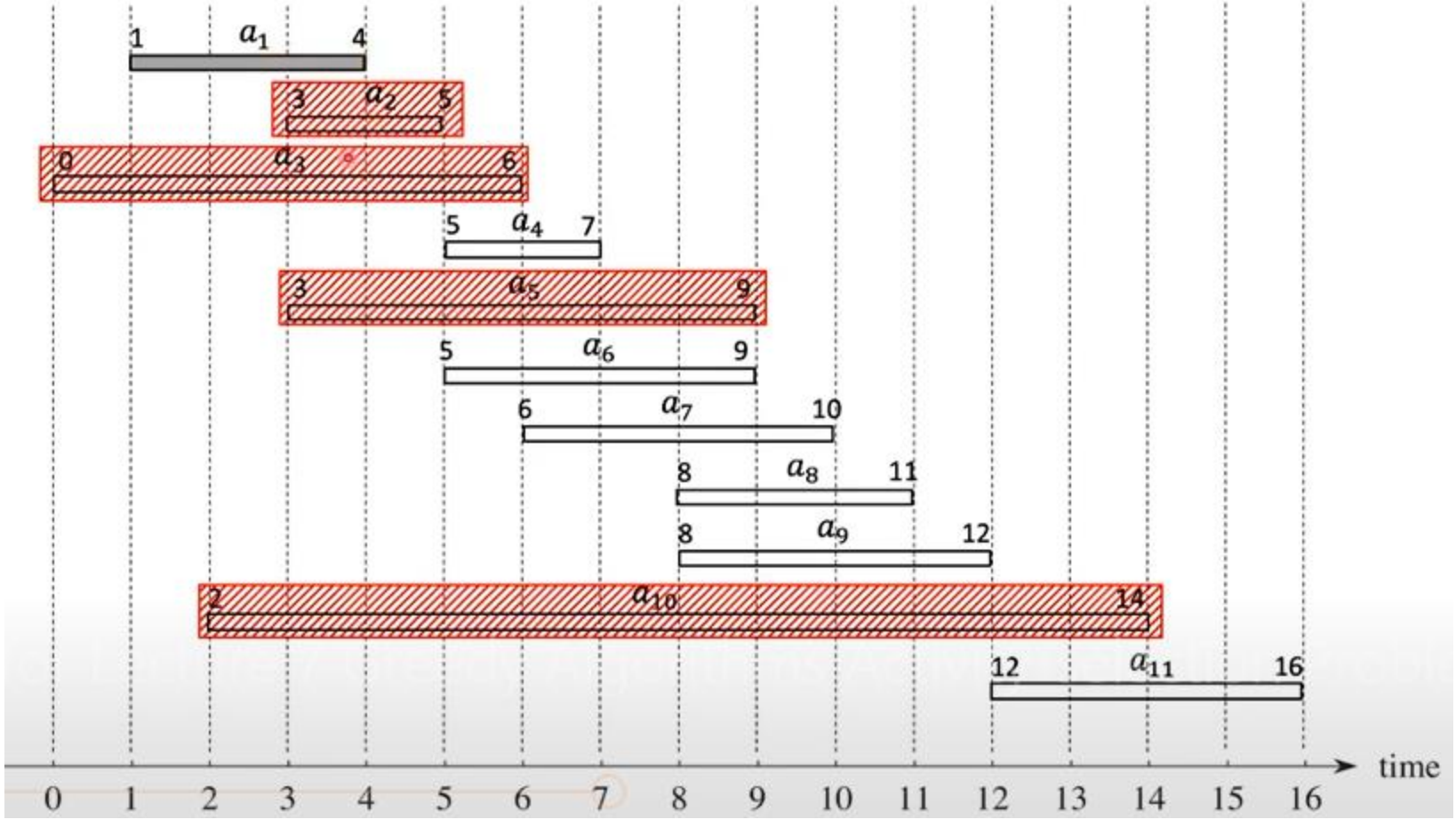
Activities a_i and a_j are compatible if $s_i \geq f_j$:



or $s_j \geq f_i$:




1
a_1
4
3
a_2
5
0
a_3
6
5
a_4
7
3
a_5
9
5
a_6
9
6
a_7
10
8
a_8
11
8
a_9
12
2
a_{10}
14
12
a_{11}
16
time
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16


1
a_1
4
3
a_2
5
0
a_3
6
5
a_4
7
3
a_5
9
5
a_6
9
6
a_7
10
8
a_8
11
8
a_9
12
2
a_{10}
14
12
a_{11}
16
time
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

$A = \{a_1\} \quad k = 1$

$s[4] \geq f[1]$

$A = \{a_1, a_4\} \quad k = 4$

$s[8] \geq f[4]$

$A = \{a_1, a_4, a_8\} \quad k = 8$

$s[11] \geq f[8]$

$A = \{a_1, a_4, a_8, a_{11}\} \quad k = 11$

Iterative implementation of Activity Selection Problem:

$Algo\ \ Greedy\_Activity\_Selector(s, f)$

{

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

}

Other heuristics:

- Chose the activity with the latest start time. Then the solution is $\{a_2, a_4, a_9, a_{11}\}$.
- Sort activities in the increasing order of start time.

Running time complexity of this problem is $\theta(n)$ without considering the time for sorting.

Job sequencing with deadlines:

Given an array of jobs with every job has deadline and associated profit if the job is finished before the deadline. It is also given the every job takes single unit of time. How to maximize total profit if only one job can be scheduled at a time. That is we have to schedule these jobs in such a way so as to maximize the total profit.

Three steps greedy algorithm is:

1. Sort all the jobs in decreasing order of profit.
2. Initialize the result sequence as first job in sorted jobs.
3. Do following for the remaining $n-1$ jobs:
    a. If the current job can fit in the current result sequence without missing the deadline, add current job to the result, and else ignore the current job.

```
Algo JS_deadline(d, j, n)
{
d[0] = j[0] = 0;
j[1] = 1; // Assign first job
k = 1; // No. of jobs chosen
for i = 2 to n
{
r = k;
while(d[j[r]] > d[i] && d[j[r] ≠ r) // Shift in upward
{r = r − 1;}
if (d[j[r]] ≤ d[i] &&d[i] > r) // Check the feasibility
{
 for(q = k to r + 1 step − 1) do
 {
 j[q + 1] = j[q]; // to shift the existing job downwards
 }
 j[r + 1] = i; // insert the job in sequence
 k = k + 1; // increment the job count
 }
 }
 return k;}
```

THANKS !