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:

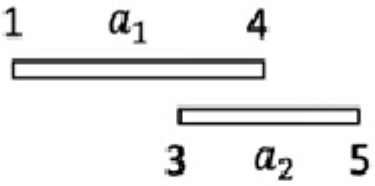
1. Smallest activity first
2. Activity that overlaps with smallest number of other activities.
3. Arrange the activities according to earliest finish time.

Steps of correct solution are:

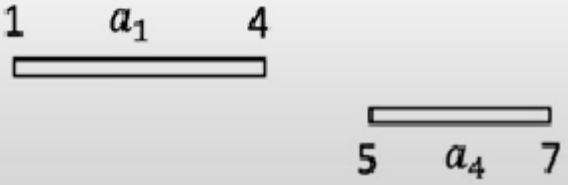
- Pick activity that ends first
- Eliminate activities that overlap with the chosen activity
- Repeat until no more activities left.

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

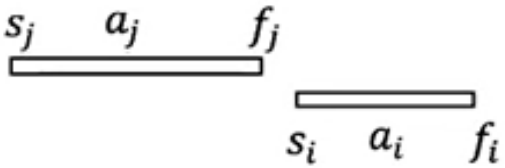
- E.g., a_1 and a_2 are *incompatible*



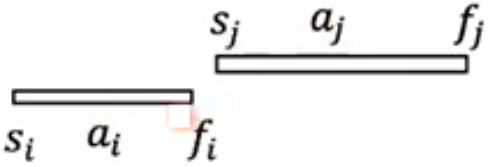
- E.g., a_1 and a_4 are *compatible*

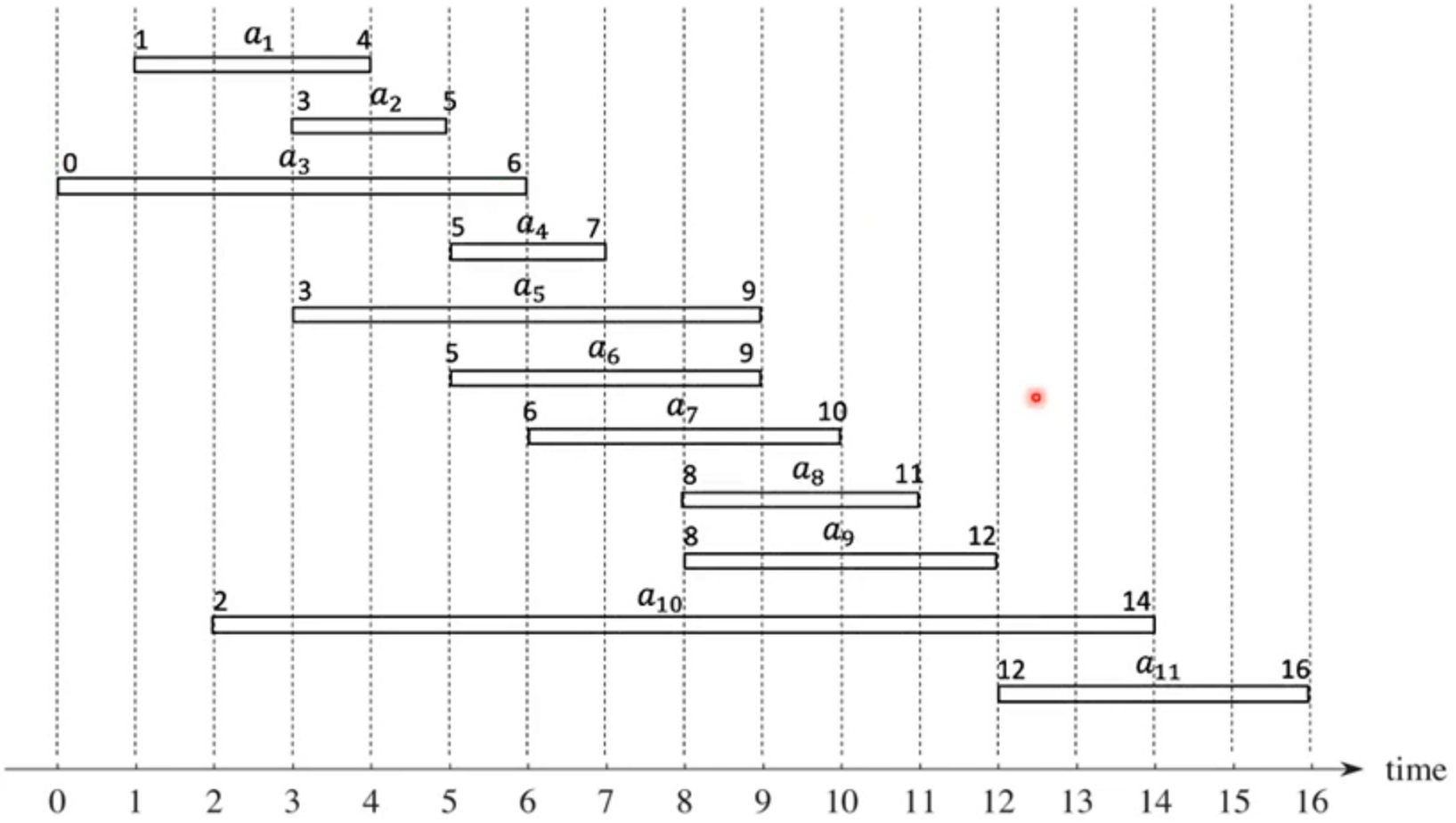


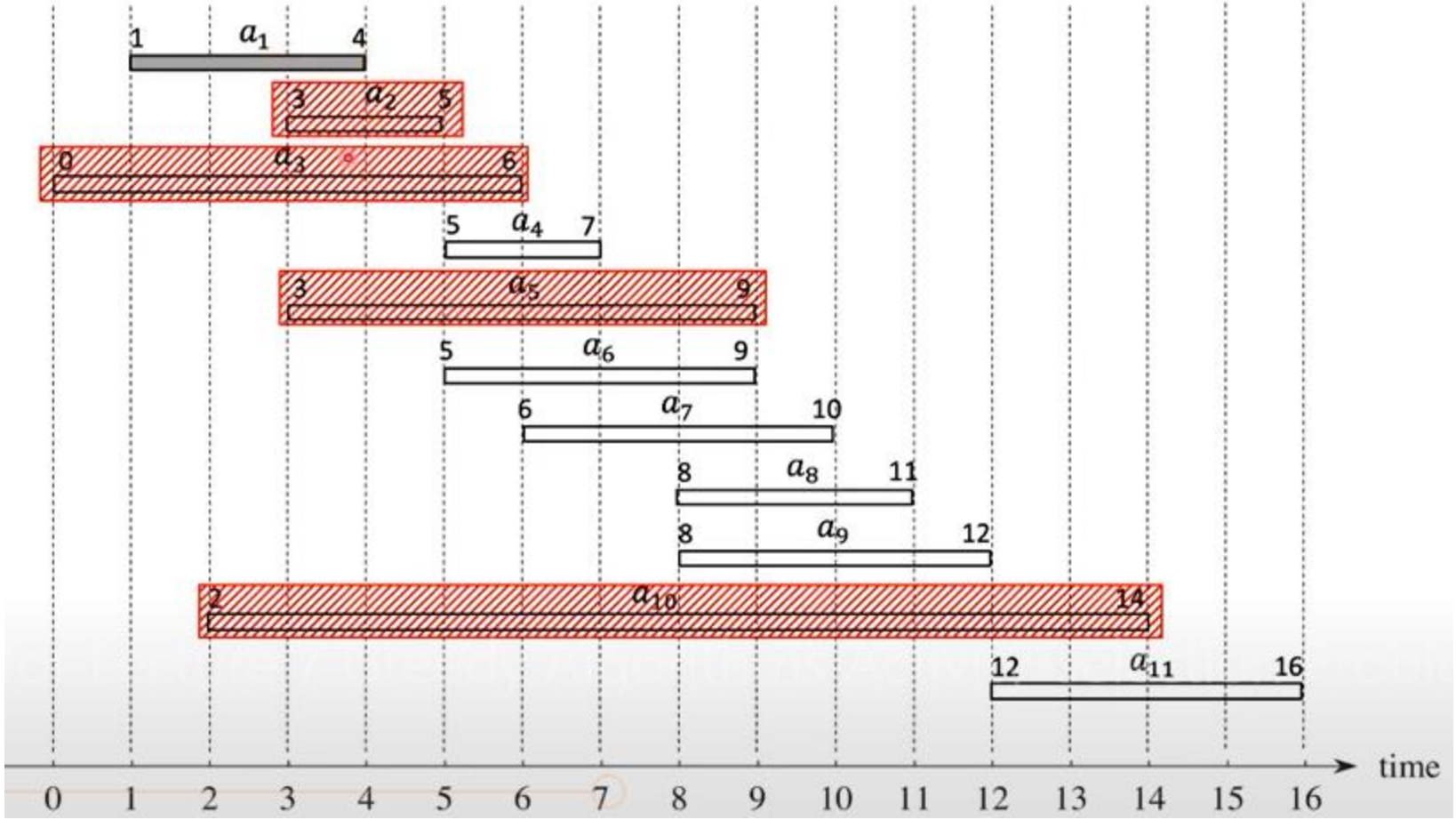
Activities a_i and a_j are compatible if $s_i \geq f_j$:



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

- In job sequencing problem, the objective is to find a sequence of jobs, which is completed within their deadlines and gives maximum profit..

- Let us consider, a set of ***n given jobs which are associated with deadlines and profit is*** earned, if a job is completed by its deadline.

- These jobs need to be ordered in such a way that there is maximum profit.

- It may happen that all of the given jobs may not be completed within their deadlines.

- Assume, deadline of **ith job** ***Ji is di and the profit received from this job is pi. Hence, the*** optimal solution of this algorithm is a feasible solution with maximum profit.
- Thus, $\boldsymbol{D(i) > 0}$ for $\boldsymbol{1 \leq i \leq n}$.
- Initially, these jobs are ordered according to profit, i.e. $\boldsymbol{p1 \geq p2 \geq p3 \geq \ldots \geq pn}$.

Ex: J=[j1,j2,j3,j4] P=[100,27,15,10], D= [2,1,2,1]

J=[j1,j2,j3,j4,j5] P=[20,15,10,5,1], D= [2,2,1,3,3]

- Example: Let us consider a set of given jobs as shown in the following table. We have to find a sequence of jobs, which will be completed within their deadlines and will give maximum profit. Each job is associated with a deadline and profit.

INDEX	1	2	3	4	5
JOB	J1	J2	J3	J4	J5
DEADLINE	2	1	3	2	1
PROFIT	60	100	20	40	20

Solution:

- To solve this problem, the given jobs are sorted according to their profit in a descending order. Hence, after sorting, the jobs are ordered as shown in the following table.

INDEX	1	2	3	4	5
JOB	J2	J1	J4	J3	J5
DEAD LINE	1	2	2	3	1
PROFIT	100	60	40	20	20

- From this set of jobs, first we select ***J2, as it can be completed within its deadline and*** contributes maximum profit.
- Next, ***J1 is selected as it gives more profit compared to J4.***

- In the next clock, ***J4 cannot be selected as its deadline is over, hence J3 is selected***

  as it executes within its deadline.

- The job ***J5 is discarded as it cannot be executed within its deadline.***

- Thus, the solution is the sequence of jobs (***J2, J1, J3), which are being executed within*** their deadline and gives maximum profit.

- Total profit of this sequence is **100 + 60 + 20 = 180**

Ex2:-n=4, (p1,p2,p3,p4)=(100,10,15,27)
(d1,d2,d3,d4)=(2,1,2,1)

The maximum deadline is 2 units, hence the feasible solution set must have <=2 jobs.

	feasible solution	processing sequence	value
1.	(1,2)	2,1	110
2.	(1,3)	1,3 or 3,1	115
3.	(1,4)	4,1	127
4.	(2,3)	2,3	25
5.	(3,4)	4,3	42
6.	(1)	1	100
7.	(2)	2	10
8.	(3)	3	15
9.	(4)	4	27

Solution 3 is optimal.

```
AlgorithmJS(d,j,n)
//d[i]≥1, 1 ≤ i ≤ n are the deadlines.
//The jobs are ordered such that p[1] ≥p[2] …… ≥p[n]
// j[i] is the i^th job in the optimal solution, 1≤ i ≤ k
{
        d[0]=j[0]=0;    // Initialize
        j[1]=1; // Include job 1 which is having highest profit
        k=1; // no of jobs considered till now

        for i=2 to n do
    {   //Consider jobs in Descending order of p[i].
        // Find position for i and  check feasibility of  insertion.
        r=k;
```

```
while( ( d[ j[r] ] > d[ i ] ) and ( d[ j[ r ] ] != r ) ) do
//checking present job with existing jobs in array and try to shift job towards
    upwards in list
    {
        r = r-1;
    }
    if( d[ j[r] ] ≤ d[i] and d[ i ] > r )) then //job can be inserted or not
        {
            // Insert i into j[ ].
            for q=k to (r+1) step -1 do // shift existing jobs downwards
            {
                j[q+1] = j[q];
            }
            j[r+1] :=i; // insert job in sequence
            k:=k+1; // increase job count
    }   }
    return k; }
```

Time taken by this algorithm is $o(n^2)$

Assignment

- Let n=4 (p1...p4)=(20,15,10,5,1)
  (d1...d4)=(2,2,1,3,3)

0/1 knapsack problem

- The Greedy algorithm could be understood very well with a well-known problem referred to as Knapsack problem.

- Although the same problem could be solved by employing other algorithmic approaches.

- Greedy approach solves **Fractional Knapsack problem** reasonably in a good time.

- There are items i1, i2, ..., in each having weight w1, w2, … w_n and some benefit (value or profit) associated with it p1, p2, ..., p_n

- Our objective is to maximize the benefit or profit such that the total weight inside the knapsack is at most M, and we are also allowed to take an item in fractional part.

- There are **n items in the store**

  -weight of **ith item** $w_i > 0$

  - Profit for **ith item** $p_i > 0$ **and**
- Capacity of the Knapsack is **M.**

- In this version of Knapsack problem, items can be broken into smaller pieces, take fraction x_i ***of ith item.***

$$0 \leq x_i \leq 1$$

- The ith item contributes the weight $x_i * w_i$ to the total weight in the knapsack and profit $x_i * p_i$ to the total profit.

Hence, the objective of this algorithm is to

$$\textbf{maximize} \sum_{n=1 \text{ to } n} (x_i . p_i)$$

- subject to constraint,

$$\Sigma_{n=1 \text{ to } n}(\boldsymbol{xi}.\ \boldsymbol{wi}) \leq M$$

- It is clear that an optimal solution must fill the knapsack exactly, otherwise we could add a fraction of one of the remaining items and increase the overall profit.

- Thus, an optimal solution can be obtained by

$$\Sigma_{n=1 \text{ to } n}(\boldsymbol{xi}.\ \boldsymbol{wi}) = M$$

- In this context, first we need to sort those items according to the value of $\boldsymbol{pi/wi}$, **so that** $(\boldsymbol{pi}+1)/(\boldsymbol{wi}+1) \leq \boldsymbol{pi/wi}$.
- **Here,** ***x is an array to store the fraction of items.***

- Example 1:

  Let us consider that the capacity of the knapsack M = **60** and the list of provided item are shown in the following table:

ITEM	A	B	C	D
PROFIT	280	100	120	120
WEIGHT	40	10	20	24
RATIO (Pi/Wi)				

- As the provided items are not sorted based on **pi / wi. After sorting, the items are as** shown in the following table.

ITEM	B	A	C	D
PROFIT	100	280	120	120
WEIGHT	10	40	20	24
RATIO (Pi/Wi)	10	7	6	5

- After sorting all the items according to pi/wi. **First all of *B is chosen as weight of B is less*** than the capacity of the knapsack.
- Next, item ***A is chosen, as the available capacity of the*** knapsack is greater than the weight of ***A.***
- ***Now, C is chosen as the next item. However,*** the whole item cannot be chosen as the remaining capacity of the knapsack is less than the weight of ***C.***

  Hence, fraction of ***C (i.e. (60 – 50)/20) is chosen.***

- Now, the capacity of the Knapsack is equal to the selected items. Hence, no more item

  can be selected.

- The total weight of the selected items is $\mathbf{10 + 40 + 20 * (10/20) = 60}$

- And the total profit is $\mathbf{100 + 280 + 120 * (10/20) = 380 + 60 = 440}$
- This is the optimal solution. We cannot gain more profit selecting any different combination of items.

Example 2: KANPSACK SIZE IS M=16

ITEM	WEIGHT	PROFIT
1	6	6
2	10	2
3	3	1
4	5	8
5	1	3
6	3	5