Agenda:

- What are Greedy Algorithms?
- Problem-1: Maximizing the number of tasks
- Problem-2: Fractional Knapsack
- Problem-3: Meeting rooms
- Problem-4: Job Sequencing problem
- Problem-5: Majority Element

What are Greedy Algorithms?

A greedy algorithm makes the choice that appears best at that instance of time with the hope of finding the best possible result.
It basically takes decisions based upon the concept of local optimality. The current best is considered the global best and the decisions are taken on the basis of it.

Problem-1: Maximizing the number of tasks

Imagine you are a very busy person and you have lots of interesting things to be done within a short span of time T. You would want to do maximum of those interesting to-do things in the short time you have.
You have an integer array A, where each element Ai represents the time taken to complete a task. Your main task is now to compute the maximum number of things that can be done in the limited time T.

Example: A = {4, 2, 1, 2, 5}, fixed time T = 8
Output: 3

Explanation: You can do the task numbered 1, 2, and 3 (with time 4, 2, and 1 respectively).

Problem-2: Fractional Knapsack

Given weights and values of N items, we need to put these items in a knapsack of capacity W to get the maximum total value in the knapsack. You are allowed to break the items in whatever ratio you want to.

```
Input:
N = 3, W = 50
values[] = {60,100,120}
weight[] = {10,20,30}
Output:
240.00
```

Explanation:
Take the 1st item fully. Current value = 60, weight left = 40
Take the 2nd item fully. Current value = 160, weight left = 20
Take the 3rd item in (⅔)ratio. Current value = 240, weight left=0

Problem-3: Meeting rooms

Given an 2D integer array **A** of size `N x 2` denoting time intervals of different meetings.

Where:

- **A[i][0]** = start time of the i^{th} meeting.
- **A[i][1]** = end time of the i^{th} meeting.

Find the **minimum number of conference rooms** required so that all meetings can be done.

Input 1:

```
A = [   [0, 30]
        [5, 10]
        [15, 20]
    ]
```

Input 2:

```
A =  [   [1, 18]
         [18, 23]
         [15, 29]
         [4, 15]
         [2, 11]
         [5, 13]
     ]
```

Output 1:

```
2
```

Output 2:

```
4
```

Problem-4: Job Sequencing problem

Given a set of **N** jobs where each **job_i** has a deadline and profit associated with it. Each job takes ***1*** unit of time to complete and only one job can be scheduled at a time. We earn the profit if and only if the job is completed by its deadline. The task is to find the number of jobs done and the **maximum profit**.

Note: Jobs will be given in the form (Job_{id}, Deadline, Profit) associated with that Job.

```
Input:
N = 4
Jobs = {(1,4,20),(2,1,10),(3,1,40),(4,1,30)}
Output:
2 60
Explanation:
Job1 and Job3 can be done with
maximum profit of 60 (20+40).
```

```
Input:
N = 5
Jobs = {(1,2,100),(2,1,19),(3,2,27),
        (4,1,25),(5,1,15)}
Output:
2 127
Explanation:
2 jobs can be done with
maximum profit of 127 (100+27).
```

Problem-5: Majority Element

Given an array **arr**. Find the majority element in the array. If no majority exists, return -1.

Note: A majority element in an array is an element that appears **strictly** more than **arr.size()/2 times** in the array.

Examples:

Input: arr[] = [3, 1, 3, 3, 2]
Output: 3
Explanation: Since, 3 is present more than n/2 times, so it is the majority element.

Input: arr[] = [7]
Output: 7
Explanation: Since, 7 is single element and present more than n/2 times, so it is the majority element.

Input: arr[] = [2, 13]
Output: -1
Explanation: Since, no element is present more than n/2 times, so there is no majority element.

Moore’s Voting Algorithm:

This is a two-step process:

- *The first step gives the element that may be the majority element in the array. If there is a majority element in an array, then this step will definitely return majority element, otherwise, it will return candidate for majority element.*
- *Check if the element obtained from the above step is the majority element. This step is necessary as there might be no majority element.*