GREEDY

Q: N-meetings in one room/ Activity selection problem:

Intuition: sort according to the end time of it and then choose it to maximize the number of meetings.

```python
class Solution:

    #Function to find the maximum number of meetings that can
    #be performed in a meeting room.
    def maximumMeetings(self,n,start,end):
        endtime=float('-inf')
        se=[]
        for i in range(n):
            se.append([start[i],end[i]])
        se.sort(key=lambda x: x[1])
        c=0
        # print(se)
        for i in range(n):
            if se[i][0] > endtime:
                c+=1
                endtime=se[i][1]
        return c
```

```
    #Function to find the maximum number of meetings that can
    #be performed in a meeting room.
    def maximumMeetings(self, n, start, end):
        l = [[i, j] for i, j in zip(start, end)]
        # Sort the list according to end time
        l.sort(key=lambda x: x[1])

        c = 1
        e = l[0][1]

        for i in range(1, len(l)):
            if l[i][0] > e:
                c += 1
                e = l[i][1]

        return c
```

Q: Fractional Knapsack:

Step1: Sort the array according to value/weight.

Step2: simply use the concept of fractional knapsack.

```
class Solution:
    #Function to get the maximum total value in the knapsack.
    def fractionalknapsack(self, W,arr,n):
        arr.sort(key=lambda x: x.value/x.weight, reverse=True)
        profit=0
        remaining_capacity=W

        for i in range(n):
            if arr[i].weight<=remaining_capacity:
                profit+=arr[i].value
                remaining_capacity=remaining_capacity-arr[i].weight
            else:
                profit+= arr[i].value*(remaining_capacity/arr[i].weight)
                break

        return profit
```

Q:Job Sequencing Problem:

Given a set of N jobs where each jobi 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 associated with job if and only if the job is completed by its deadline.Find the number of jobs done and the maximum profit.

First sort the Jobs according to Profit.

Find max deadline and create that much slots.

Now try to Put max Profit Job as much far as possible basically on d day or just before it.

```
class Solution:

    #Function to find the maximum profit and the number of jobs done.
    def JobScheduling(self,Jobs,n):

        # code here
        Jobs.sort(key=lambda i:i.profit, reverse=True)
        maxi=Jobs[0].deadline
        for i in range(1,n):
            maxi=max(maxi,Jobs[i].deadline)

        slot = [-1]*(maxi+1)
        countJobs=0
        jobProfit=0

        for i in range(n):
            for j in range(Jobs[i].deadline,0,-1):
                if slot[j]==-1:
                    slot[j]=i
                    countJobs+=1
                    jobProfit+=Jobs[i].profit
                    break
        return countJobs,jobProfit
```

Note : Here slot = -1*(maxi+1) is used which will create one extra slot. for eg Suppose we have the following jobs:

```
Job 1: Deadline = 4, Profit = 20
```

```
Job 2: Deadline = 2, Profit = 10
```

```
Job 3: Deadline = 3, Profit = 40
```

```
Job 4: Deadline = 1, Profit = 30
```

Using slot = [-1] * (maxi + 1):
With (maxi + 1), the slot list will have 5 elements: [-1, -1, -1, -1, -1]. Each element represents a slot for a specific deadline. Initially, all slots are set to -1, indicating that they are available.

Using slot = [-1] * maxi:
With maxi, the slot list will have 4 elements: [-1, -1, -1, -1]. Here, the length of the slot list is equal to the maximum deadline. However, this approach will lead to a problem.

In the given example, Job 1 has a deadline of 4. If we try to access slot[4] to check if it's available, we would get an IndexError because the slot list has only 4 elements, and the valid indices range from 0 to 3. So, using slot = [-1] * maxi would result in an error when trying to access slot[j] where j is equal to the maximum deadline.

Activity Selection: Same as n-meeting question1

```python
#User function Template for python3

class Solution:

    #Function to find the maximum number of activities that can
    #be performed by a single person.
    def activitySelection(self,n,start,end):
        endtime=float('-inf')
        se=[]
        for i in range(n):
            se.append([start[i],end[i]])
        se.sort(key=lambda x: x[1])
        c=0
        # print(se)
        for i in range(n):
            if se[i][0] > endtime:
                c+=1
                endtime=se[i][1]
        return c
```

Q: Minimum no of platforms required:

```python
class Solution:
    #Function to find the minimum number of platforms required at the
    #railway station such that no train waits.
    def minimumPlatform(self,n,arr,dep):
        arr.sort()
        dep.sort()
        i=1
        j=0
        plat=1
        res=1
        while(i<n):
            if arr[i]<=dep[j]:
                plat+=1
                i+=1
            elif arr[i]>dep[j]:
                plat-=1
                j+=1
            res= max(res,plat)
        return res
```

ARRAYS:

Q: Set matrices to zero:
The solution is taking extra space to store which rows and columns index contain the zero and then after filling that row and column to zero.

```
class Solution:
    def setZeroes(self, matrix: List[List[int]]) -> None:
        m= len(matrix) #length of rows
        n= len(matrix[0])   #length of column
        #store those indexes for rows and column where there is 0
        row=[]
        col=[]
        for i in range(m):
            for j in range(n):
                if matrix[i][j]==0:
                    row.append(i)
                    col.append(j)
        for i in row:
            for j in range(n):
                matrix[i][j]=0
        for j in col:
            for i in range(m):
                matrix[i][j]=0
```

Optimal solution : without using extra space : to do

Q: Pascal Triangle:

```python
import math
class Solution:
    def generate(self, numRows: int) -> List[List[int]]:
        ans=[]
        for i in range(0,numRows):
            temp=[]
            for j in range(0,i+1):
                temp.append(math.comb(i,j))
            ans.append(temp)
        return ans
```