# **ASSIGNMENT 3**

* Question 1:

**Implementation of Merge sort.**

**TC: O(n Log n)**

* Solution:
* Source Code:

def merge(left, right):

    merged = []

    i = j = 0

    while i < len(left) and j < len(right):

        if left[i] <= right[j]:

            merged.append(left[i])

            i += 1

        else:

            merged.append(right[j])

            j += 1

    while i < len(left):

        merged.append(left[i])

        i += 1

    while j < len(right):

        merged.append(right[j])

        j += 1

    return merged

def merge\_sort(arr):

    if len(arr) <= 1:

        return arr

    mid = len(arr) // 2

    left\_half = merge\_sort(arr[:mid])

    right\_half = merge\_sort(arr[mid:])

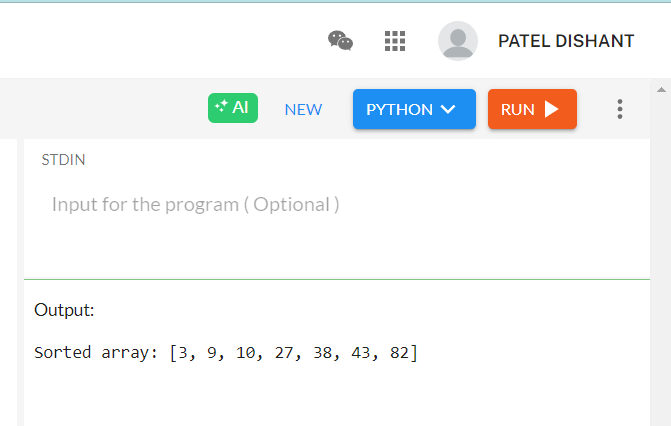
    return merge(left\_half, right\_half)

arr = [38, 27, 43, 3, 9, 82, 10]

sorted\_arr = merge\_sort(arr)

print("Sorted array:", sorted\_arr)

* Output:



* Question 2:

**Implementation of Max-Min by using Divide and Conquer principal**

**TC: O(n)**

* Solution:
* Source code:

def find\_max\_min(arr, low, high):

if low == high:

return arr[low], arr[low]

elif high == low + 1:

if arr[low] > arr[high]:

return arr[low], arr[high]

else:

return arr[high], arr[low]

mid = (low + high) // 2

max1, min1 = find\_max\_min(arr, low, mid)

max2, min2 = find\_max\_min(arr, mid + 1, high)

overall\_max = max(max1, max2)

overall\_min = min(min1, min2)

return overall\_max, overall\_min

arr = [3, 5, 1, 8, 9, 2, 7, 6]

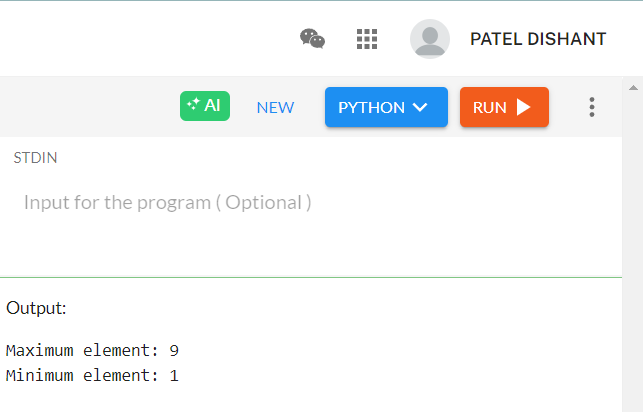
n = len(arr)

maximum, minimum = find\_max\_min(arr, 0, n - 1)

print(f"Maximum element: {maximum}")

print(f"Minimum element: {minimum}")

* Output:



* Question 3:

**Fractional Knapsack GeeksForGeeks Implementation of Fractional KnapSack TC: O(n log n) (Problem Statement: The weight of N items and their corresponding values are given. We have to put these items in a knapsack of weight W such that the total value obtained is maximized.)**

* Solution:
* Source code:

class Item:

def \_\_init\_\_(self,val,w):

self.value = val

self.weight = w

class Solution:

def fractionalknapsack(self, w,arr,n):

prof = [arr[i].value / arr[i].weight for i in range(n)]

items = [[prof[i], arr[i].value, arr[i].weight] for i in range(n)]

items.sort(key=lambda x: x[0], reverse=True)

profit = 0

i = 0

while w > 0 and i < n:

if items[i][2] <= w:

profit += items[i][1]

w -= items[i][2]

else:

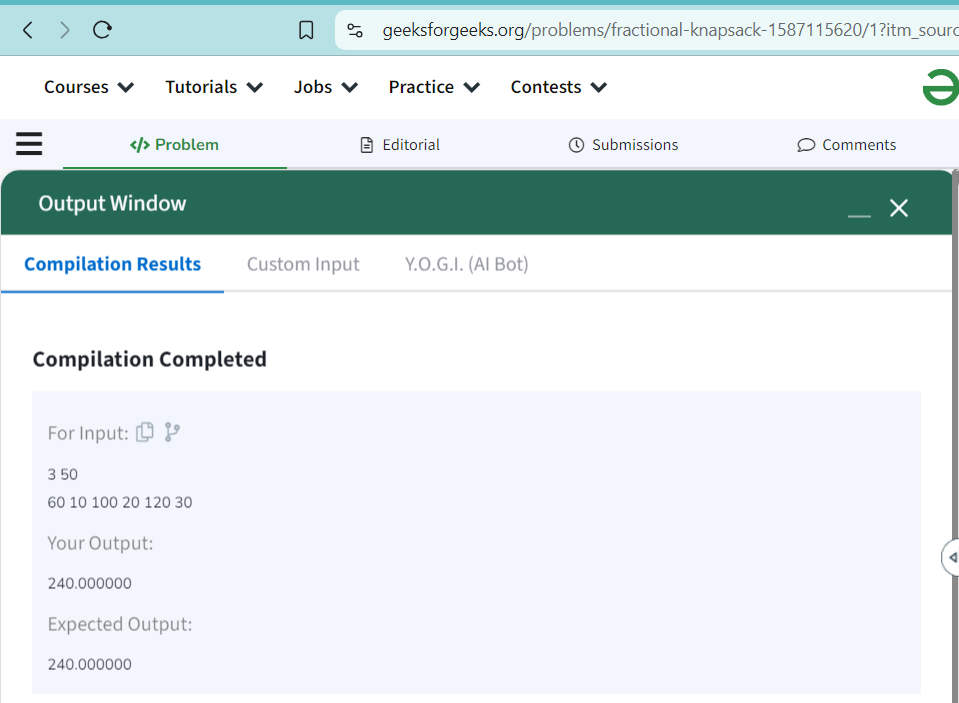
profit += items[i][0] \* w

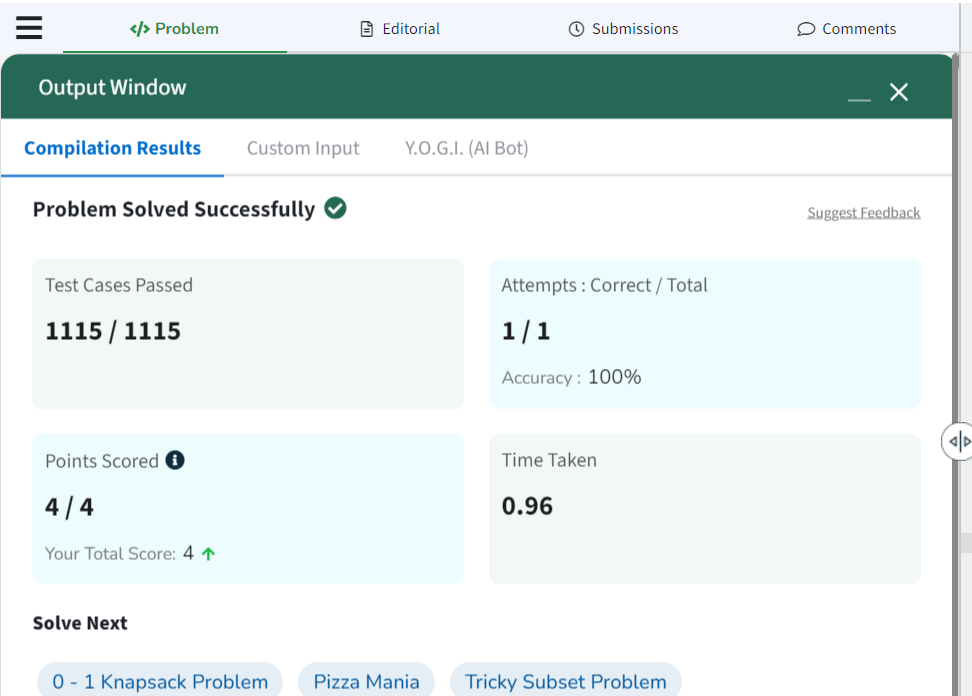
w = 0

i += 1

return profit

* Output:





* Question 4:

**Implementation of Prim’s Algorithm.**

* Solution:
* Source code:

import heapq

def prim(graph, start):

mst = []

visited = set()

min\_heap = [(0, start)]

total\_cost = 0

while min\_heap:

cost, node = heapq.heappop(min\_heap)

if node in visited:

continue

visited.add(node)

total\_cost += cost

mst.append((node, cost))

for neighbor, weight in graph[node]:

if neighbor not in visited:

heapq.heappush(min\_heap, (weight, neighbor))

return mst, total\_cost

graph = {

0: [(1, 2), (3, 6)],

1: [(0, 2), (2, 3), (3, 8), (4, 5)],

2: [(1, 3), (4, 7)],

3: [(0, 6), (1, 8)],

4: [(1, 5), (2, 7)]

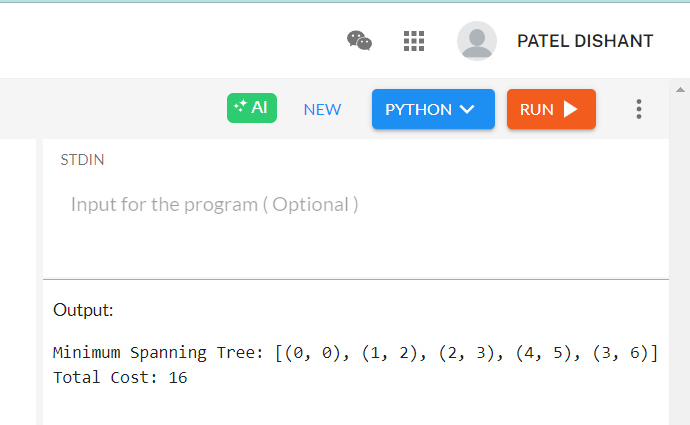
}

mst, total\_cost = prim(graph, 0)

print("Minimum Spanning Tree:", mst)

print("Total Cost:", total\_cost)

* Output:



* Question 5:

**Assign Cookies. (Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie.) Leetcode problem number: 455**

* Solution:
* Source code:

def find\_content\_children(g, s):

g.sort()

s.sort()

i = j = 0

while i < len(g) and j < len(s):

if s[j] >= g[i]:

i += 1

j += 1

return i

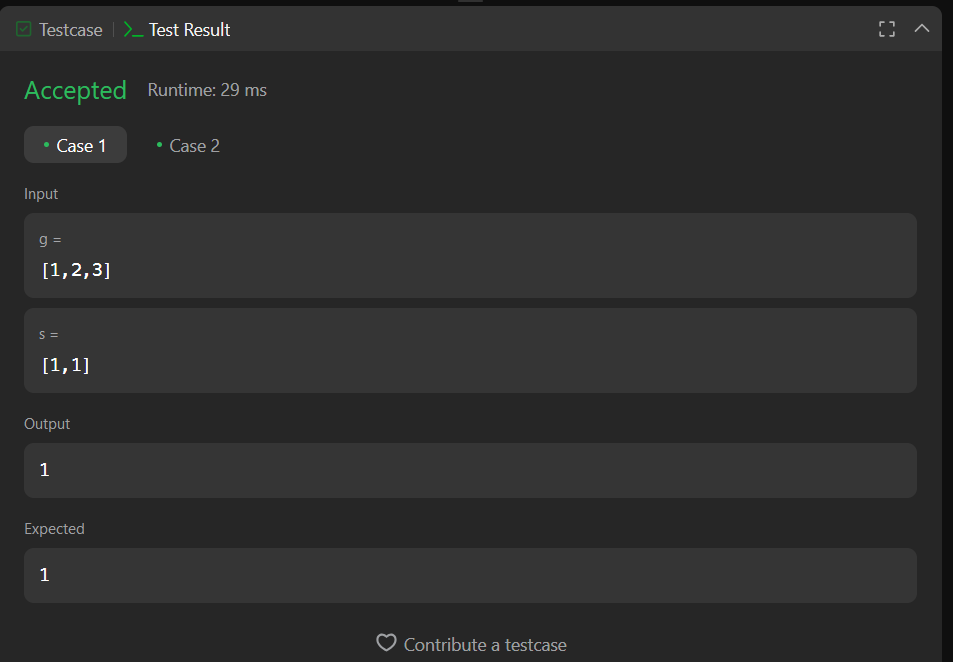
g = [1, 2, 3]

s = [1, 1]

result = find\_content\_children(g, s)

print(result)

* Output:



* Question 6:

**Maximum Units on a Truck. Leetcode problem number: 1710**

* Solution:
* Source Code:

class Solution:

def maximumUnits(self, boxTypes: List[List[int]], truckSize: int) -> int:

boxTypes.sort(key=lambda x: x[1], reverse=True)

total\_units = 0

for box\_count, units in boxTypes:

if truckSize == 0:

break

if box\_count <= truckSize:

total\_units += box\_count \* units

truckSize -= box\_count

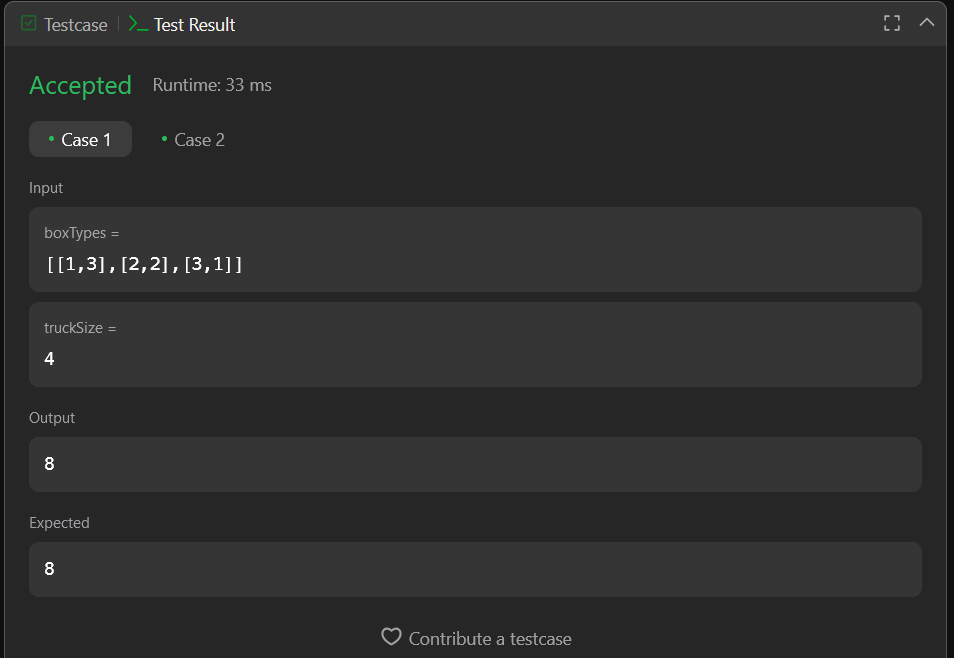
else:

total\_units += truckSize \* units

truckSize = 0

return total\_units

* Solution:



* Question 7:

**Lemonade Change. Leetcode problem number: 860**

* Solution:
* Source Code:

class Solution:

def lemonadeChange(self, bills: List[int]) -> bool:

five, ten = 0, 0

for bill in bills:

if bill == 5:

five += 1

elif bill == 10:

if five > 0:

five -= 1

ten += 1

else:

return False

elif bill == 20:

if ten > 0 and five > 0:

ten -= 1

five -= 1

elif five >= 3:

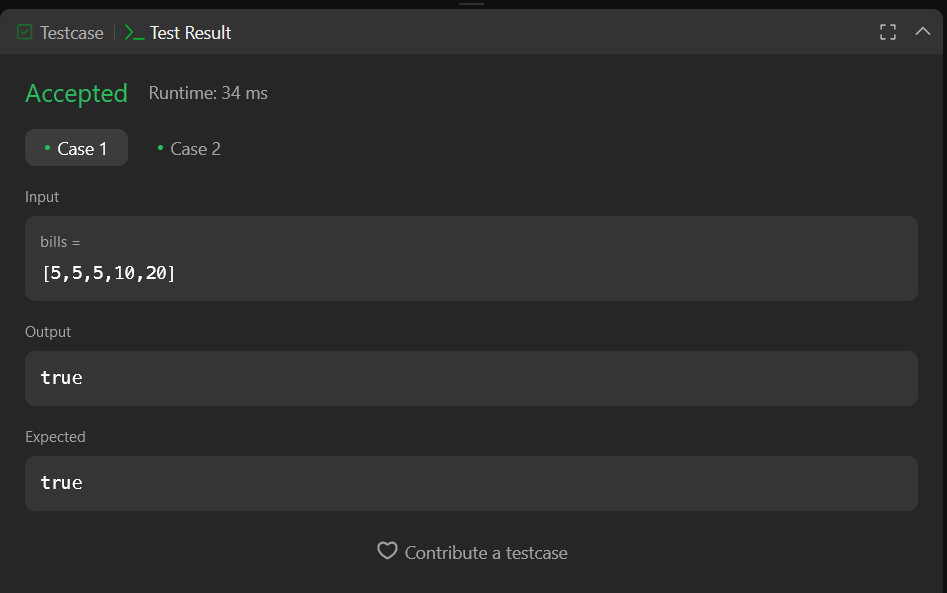
five -= 3

else:

return False

return True

* Solution:



* Question 8:

**Merge Intervals Leetcode problem number: 56**

* Solution:
* Source Code:

class Solution:

def merge(self, intervals: List[List[int]]) -> List[List[int]]:

intervals.sort(key=lambda x: x[0])

merged = []

for interval in intervals:

if not merged or merged[-1][1] < interval[0]:

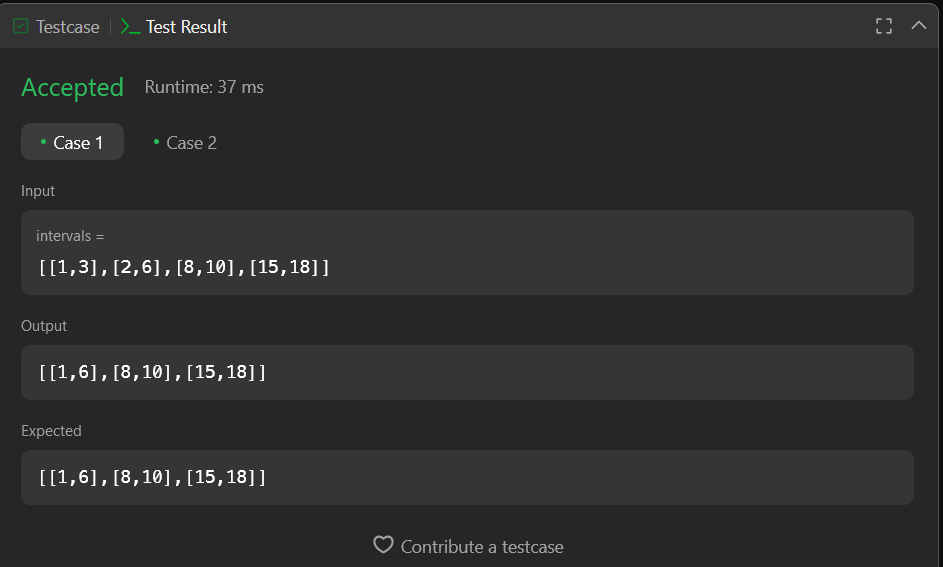
merged.append(interval)

else:

merged[-1][1] = max(merged[-1][1], interval[1])

return merged

* Output:



* Question 9:

**LCS LeetCode problem number 1143**

* Solution:
* Source Code:

class Solution:

def longestCommonSubsequence(self, text1: str, text2: str) -> int:

m, n = len(text1), len(text2)

dp = [[0] \* (n + 1) for \_ in range(m + 1)]

for i in range(1, m + 1):

for j in range(1, n + 1):

if text1[i - 1] == text2[j - 1]:

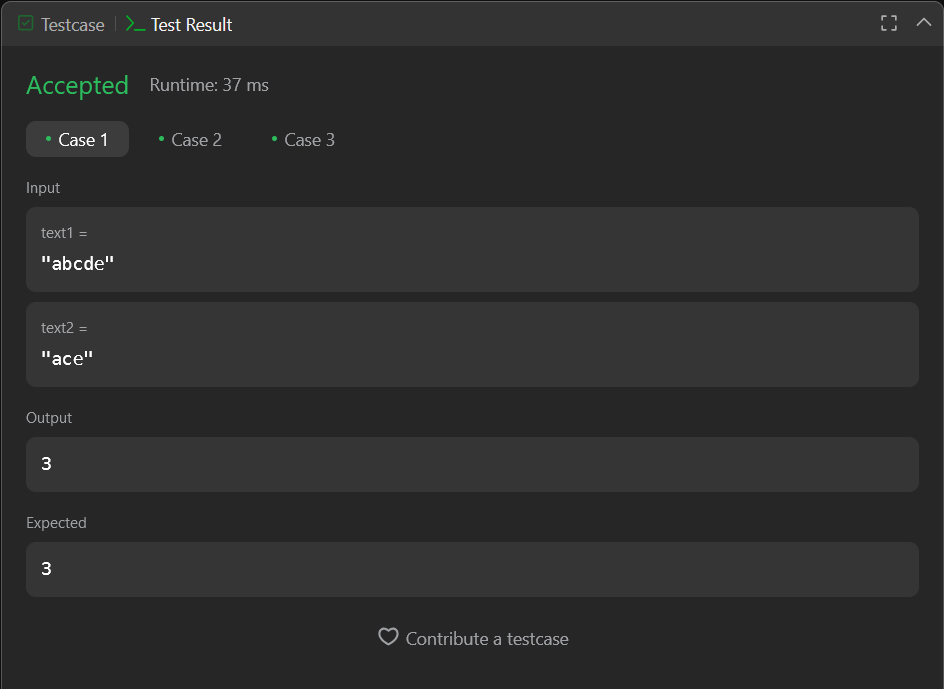
dp[i][j] = dp[i - 1][j - 1] + 1

else:

dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

return dp[m][n]

* Output:



* Question 10:

**Numberof Coins GeeksForGeeks**

* Solution:
* Source Code:

class Solution:

def minCoins(self, coins, M, sum):

k = float("inf")

dp = [[k for \_ in range(sum + 1)] for \_ in range(M + 1)]

dp[0][0] = 0

for i in range(1, M + 1):

for j in range(1, sum + 1):

if coins[i - 1] <= j:

dp[i][j] = min(dp[i][j - coins[i - 1]] + 1, dp[i - 1][j])

else:

dp[i][j] = dp[i - 1][j]

if dp[M][sum] == k:

return -1

return dp[M][sum]

# Driver code

if \_\_name\_\_ == "\_\_main\_\_":

T = int(input())

for i in range(T):

v, m = input().split()

v, m = int(v), int(m)

coins = [int(x) for x in input().split()]

ob = Solution()

ans = ob.minCoins(coins, m, v)

print(ans)

* Output:

