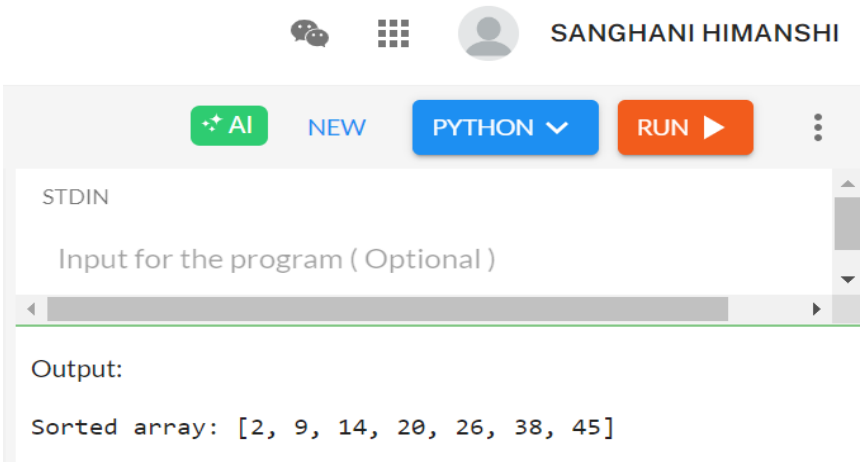
ASSIGNMENT : 3
PRACTICAL : 1

AIM : Implementation of Merge Sort. TC: O(n log n)

- **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 = [38,26,9,45,2,14,20]
sorted_arr = merge_sort(arr)
print("Sorted array:", sorted_arr)
```

- **Output :**



Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

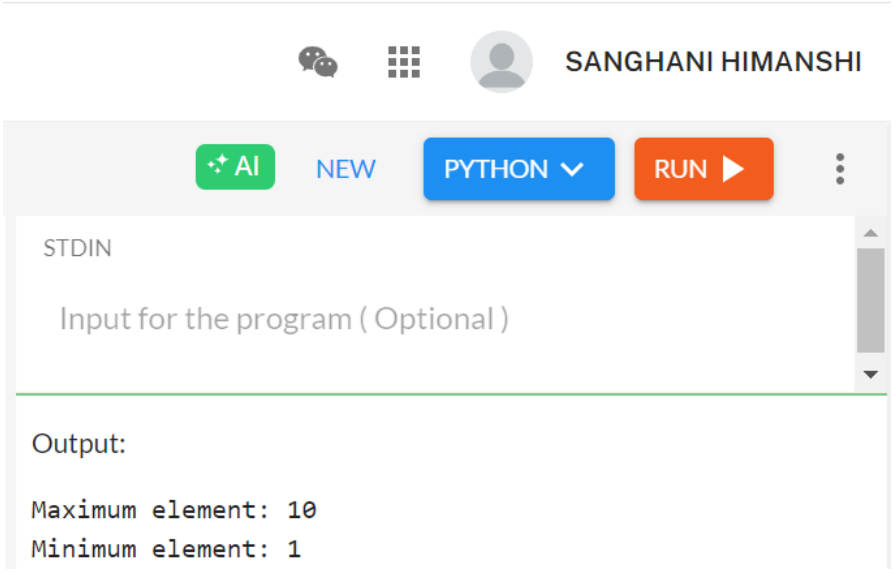
PRACTICAL : 2

AIM : Implementation of Max-Min by using Divide and Conquer principal TC: O(n)

- **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 = [2,7,5,3,9,10,1]
n = len(arr)
maximum, minimum = find_max_min(arr, 0, n - 1)
print("Maximum element:",maximum)
print("Minimum element:",minimum)
```

- **Output :**



Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

PRACTICAL : 3

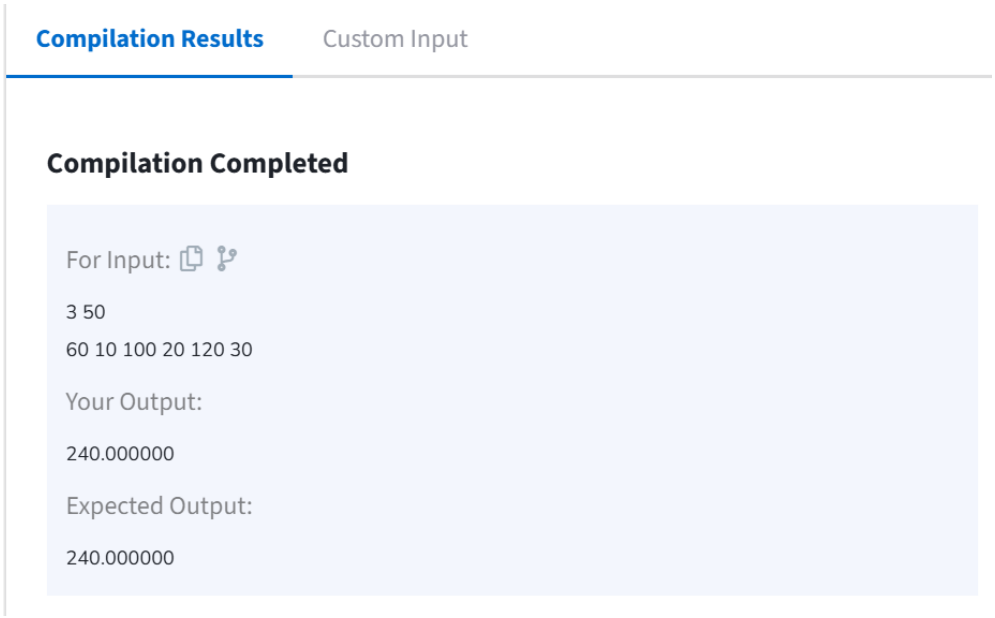
AIM : Fractional Knapsack GeeksForGeeks Implementation of Fractional KnapSack TC: O(n log n) (Problem Statement: The weight of N items and their corresponding values are given. We have to put these items in a knapsack of weight W such that the total value obtained is maximized.)

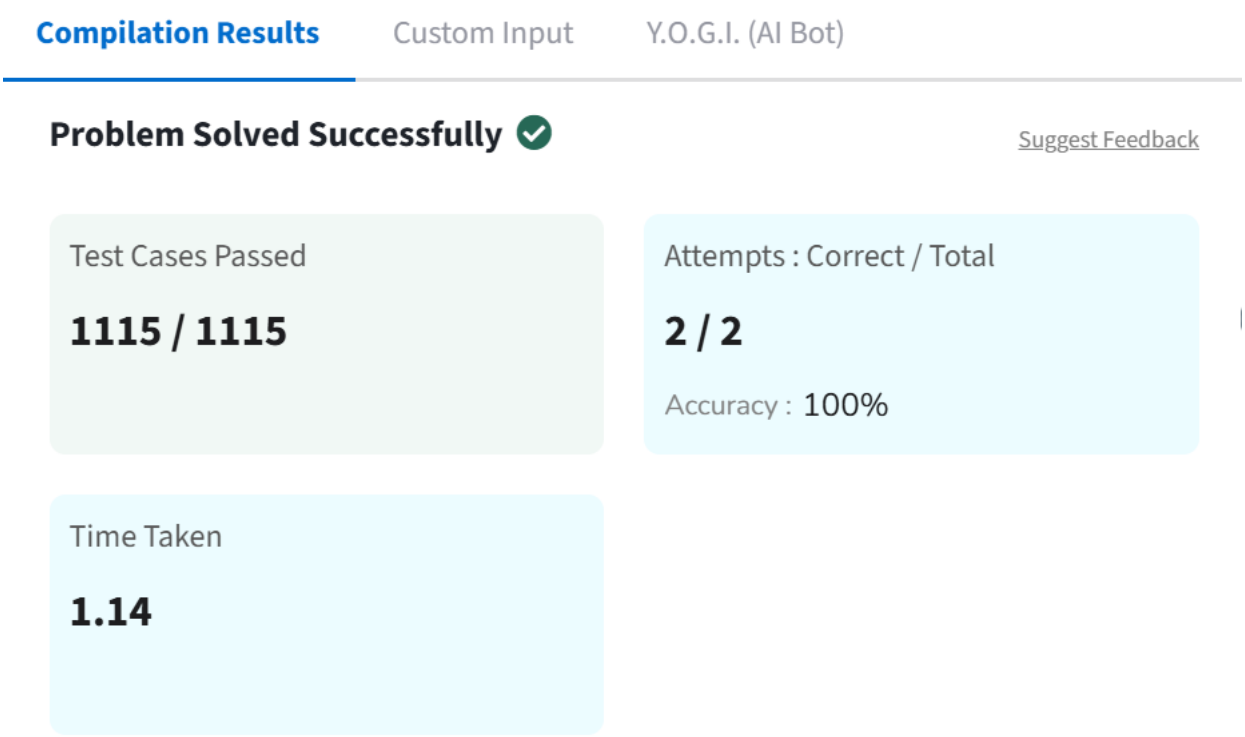
- **Code :**

```
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
```


Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

➢ **Output :**





Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

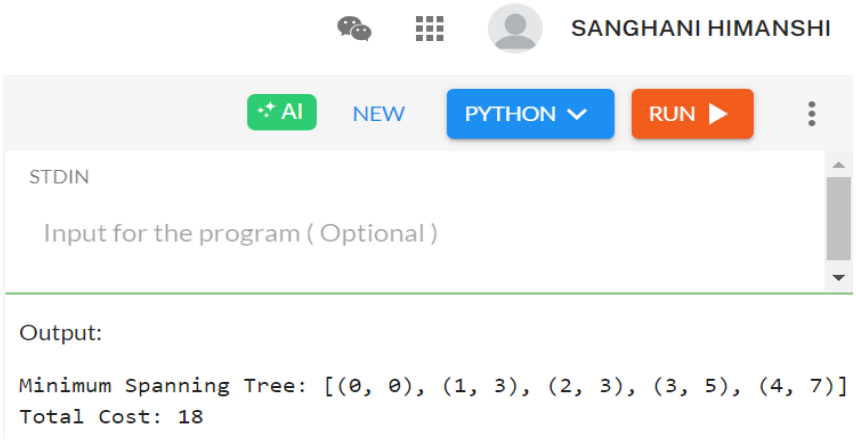
PRACTICAL : 4

AIM : Implementation of Prim's Algorithm.

- **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, 3), (3, 6)],
    1: [(0, 2), (2, 3), (4, 8), (2, 5)],
    2: [(3, 5), (4, 7)],
    3: [(0, 6), (2, 8)],
    4: [(1, 5), (3, 7)]  }
mst, total_cost = prim(graph, 0)
print("Minimum Spanning Tree:", mst)
print("Total Cost:", total_cost)
```

- **Output :**



Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

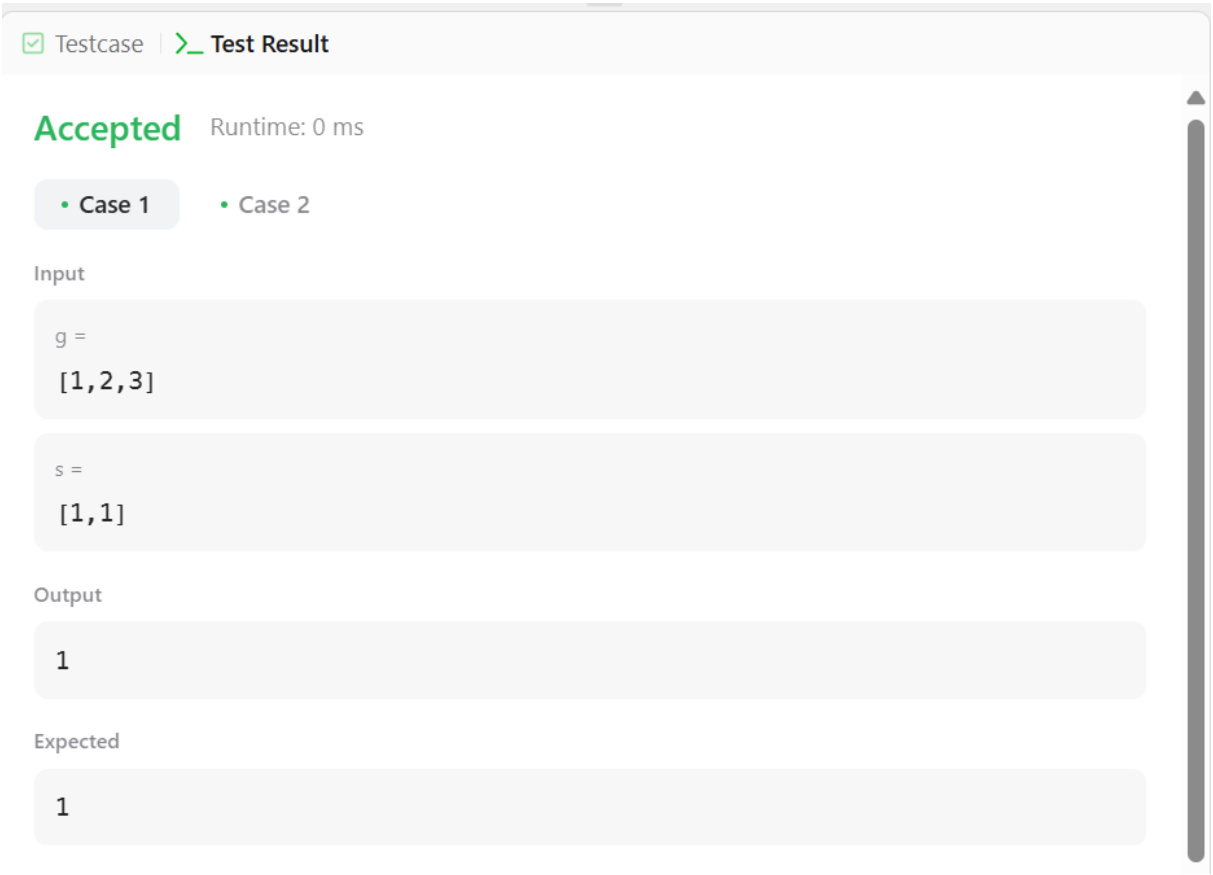
PRACTICAL : 5

AIM : 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

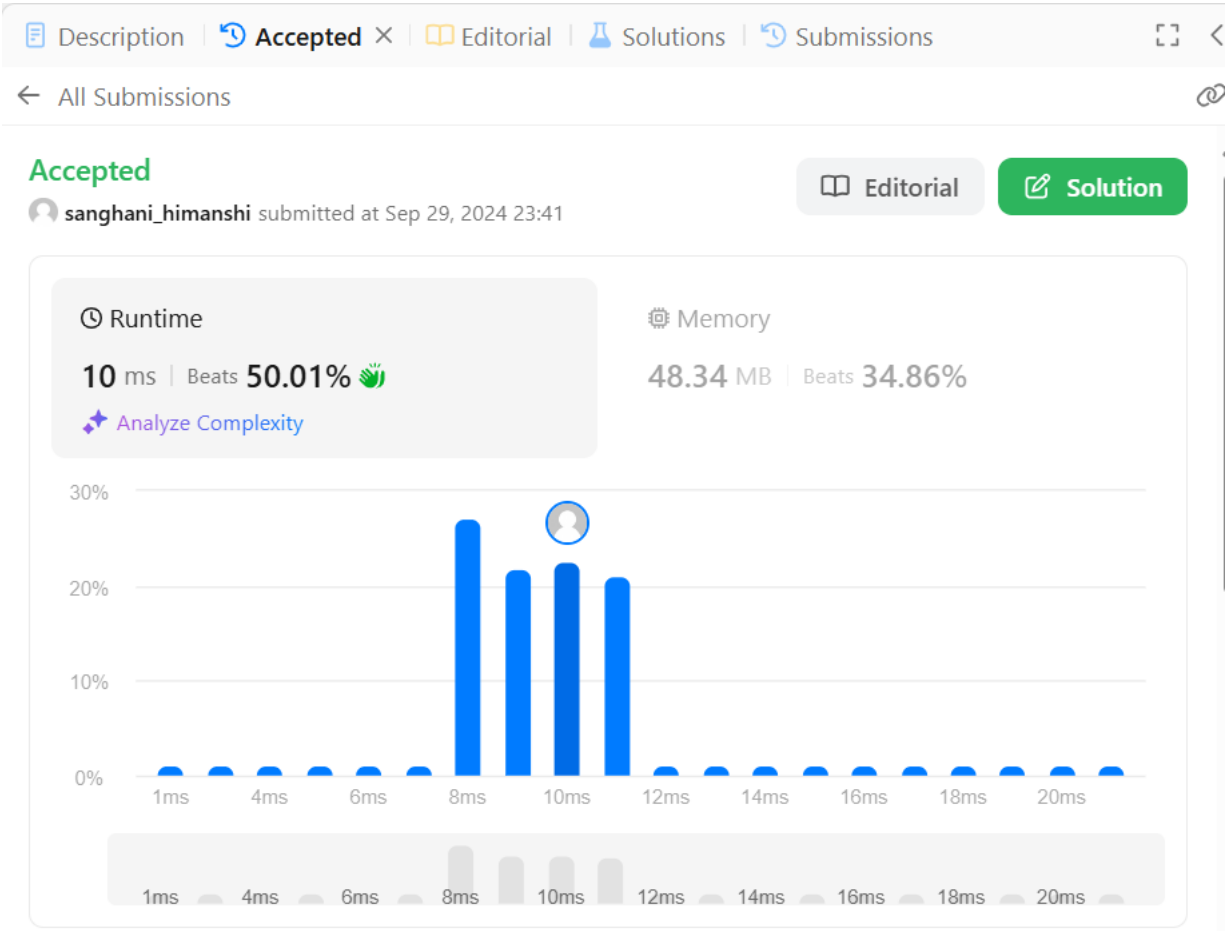
- **Code :**

```
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 :**



Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32



Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

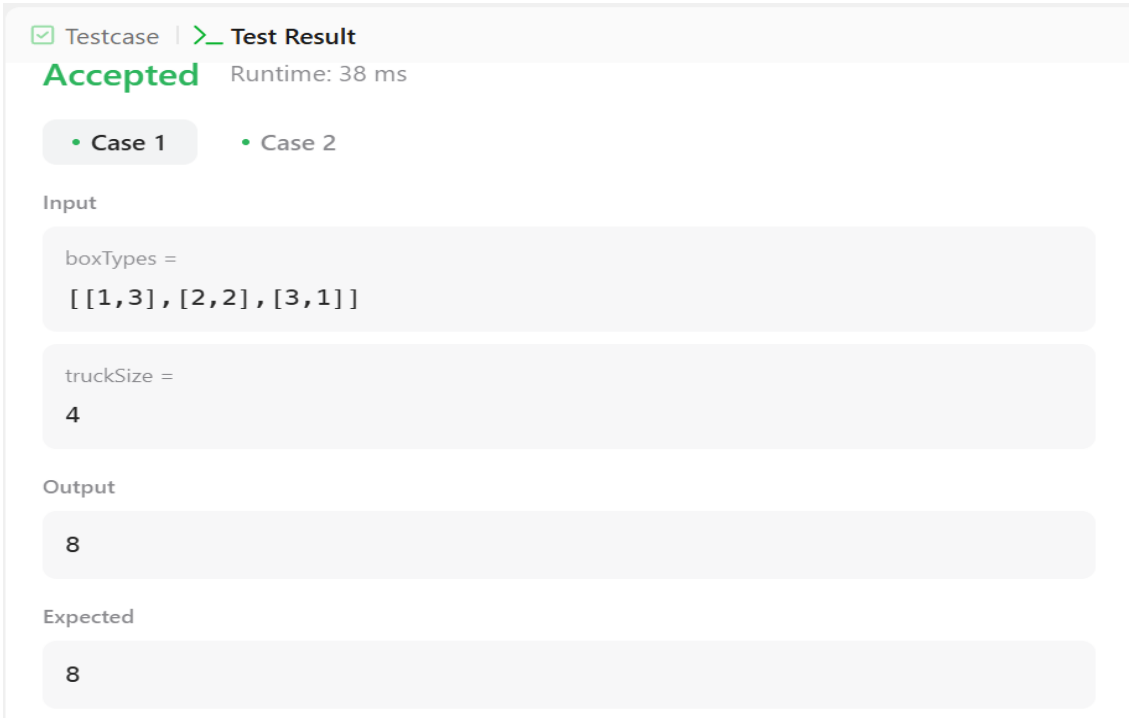
PRACTICAL : 6

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

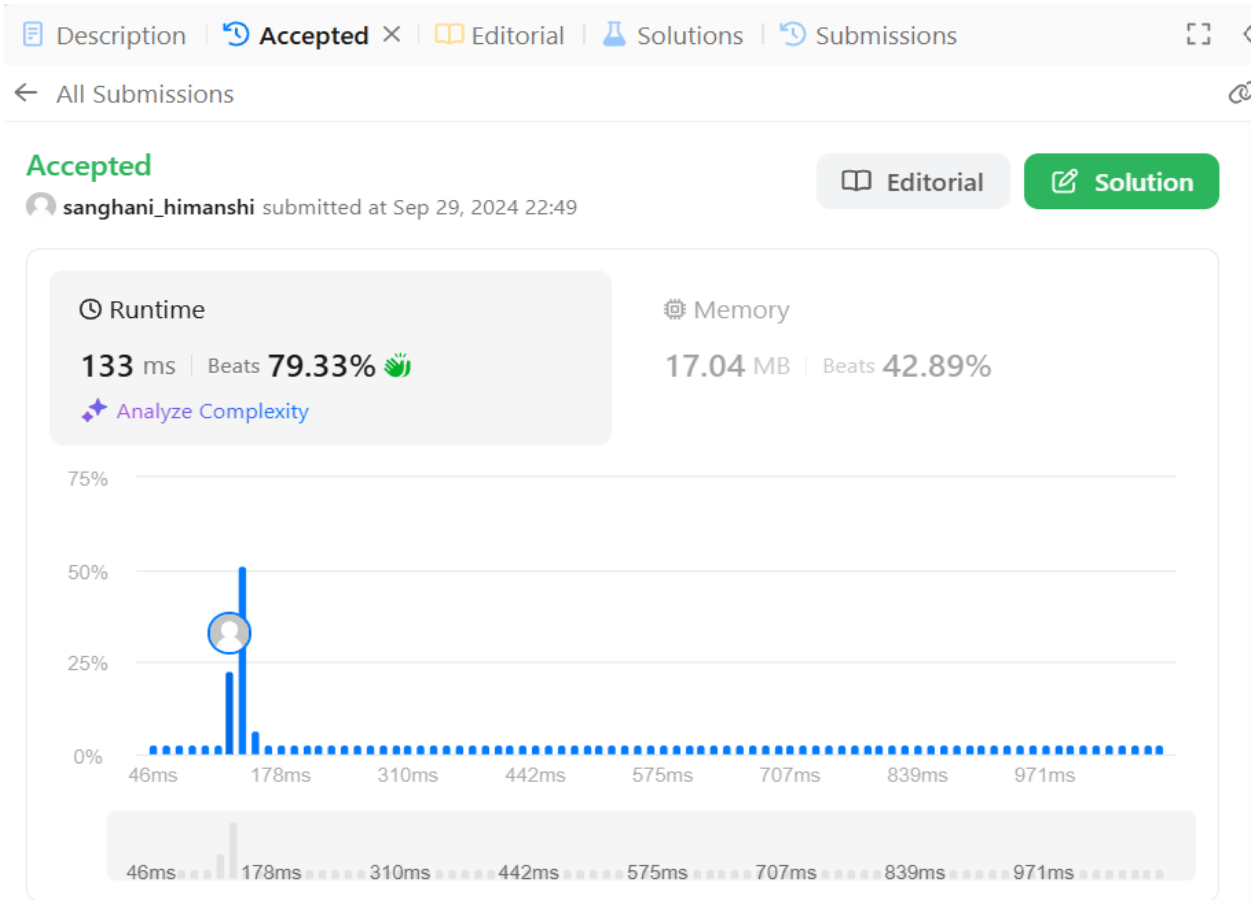
- **Code :**

```
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
```

- **Output :**



Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32



Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

PRACTICAL : 7

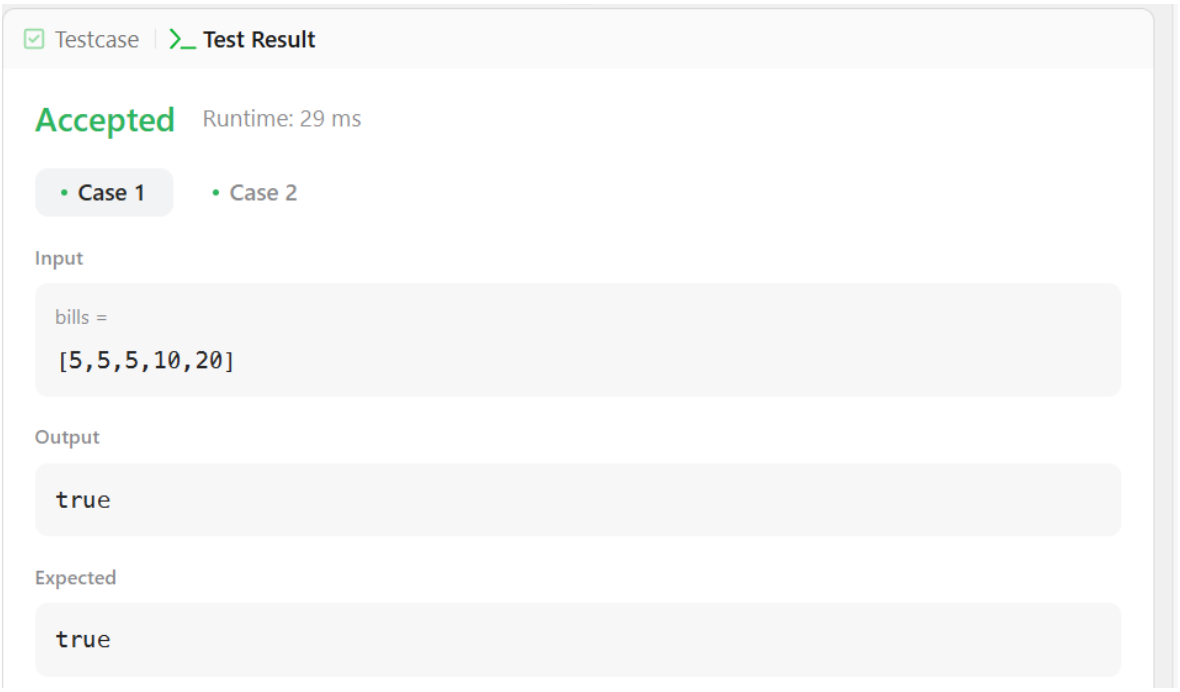
AIM : Lemonade Change. Leetcode problem number: 860

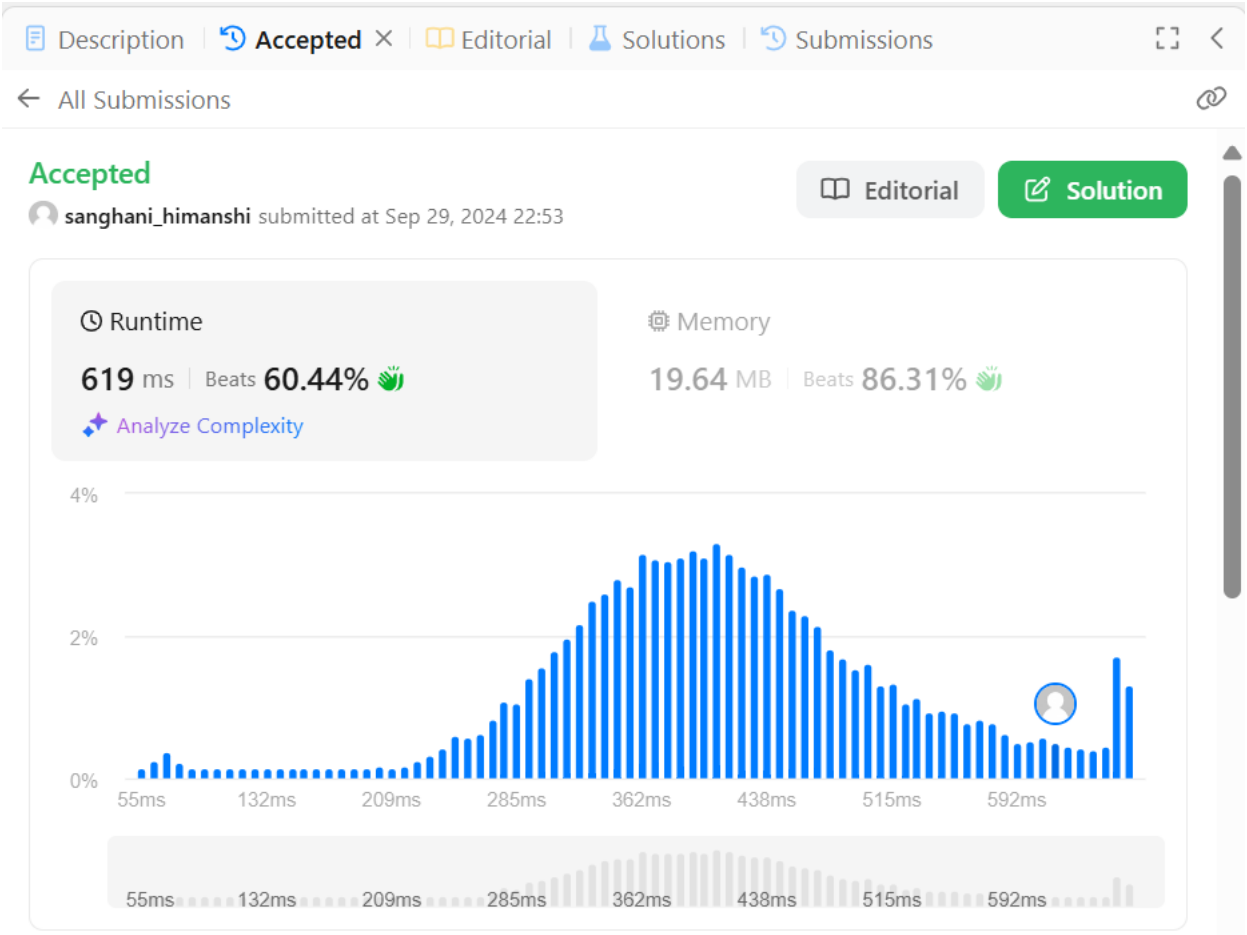
- **Code :**

```
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
```


Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

➢ **Output :**





Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

PRACTICAL : 8

AIM : Merge Intervals Leetcode problem number: 56

- **Code :**

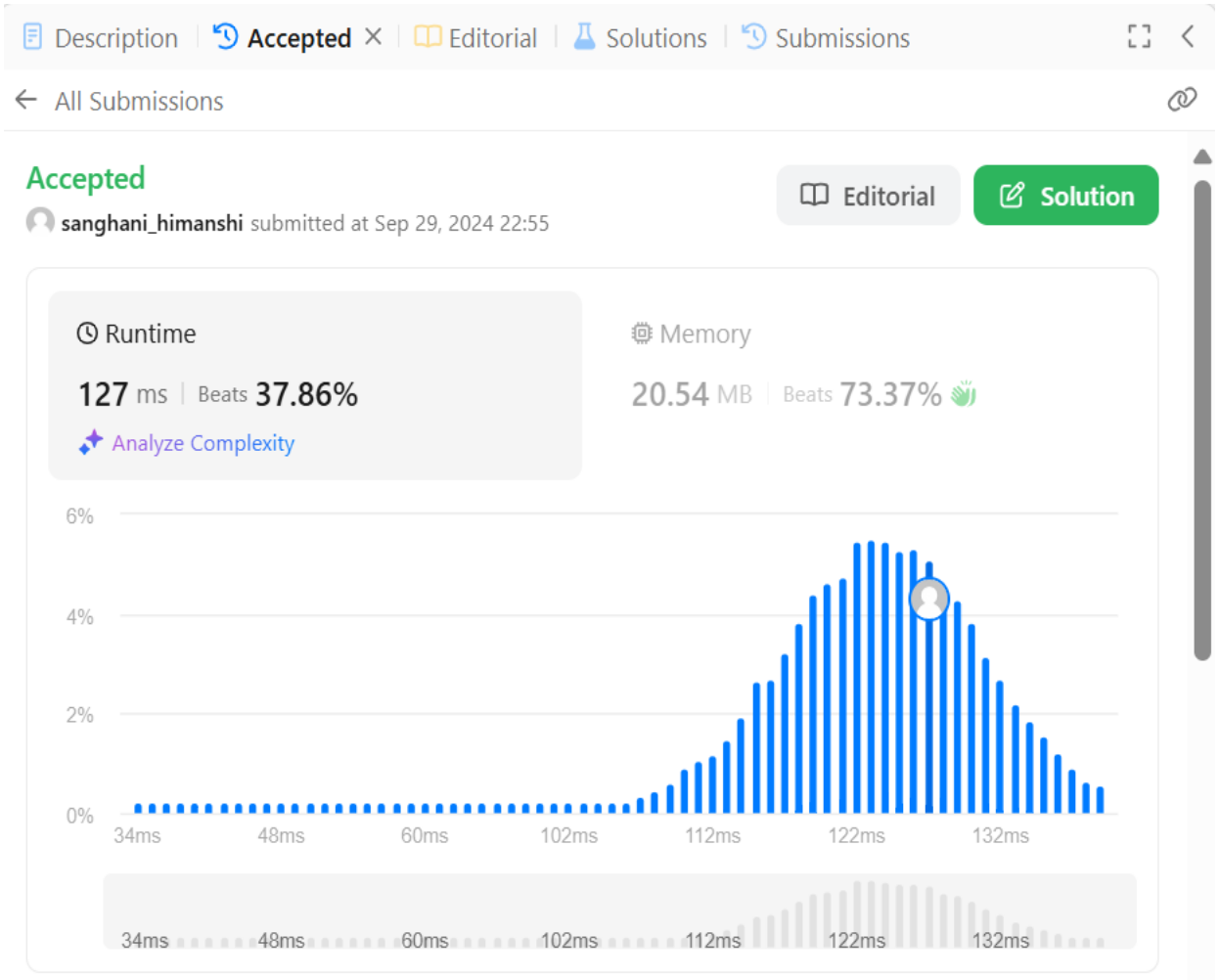
```
class Solution:
 def merge(self, intervals: List[List[int]]) -> List[List[int]]:
    intervals.sort(key=lambda x: x[0])
    merged = []

    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 :**

Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32



Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

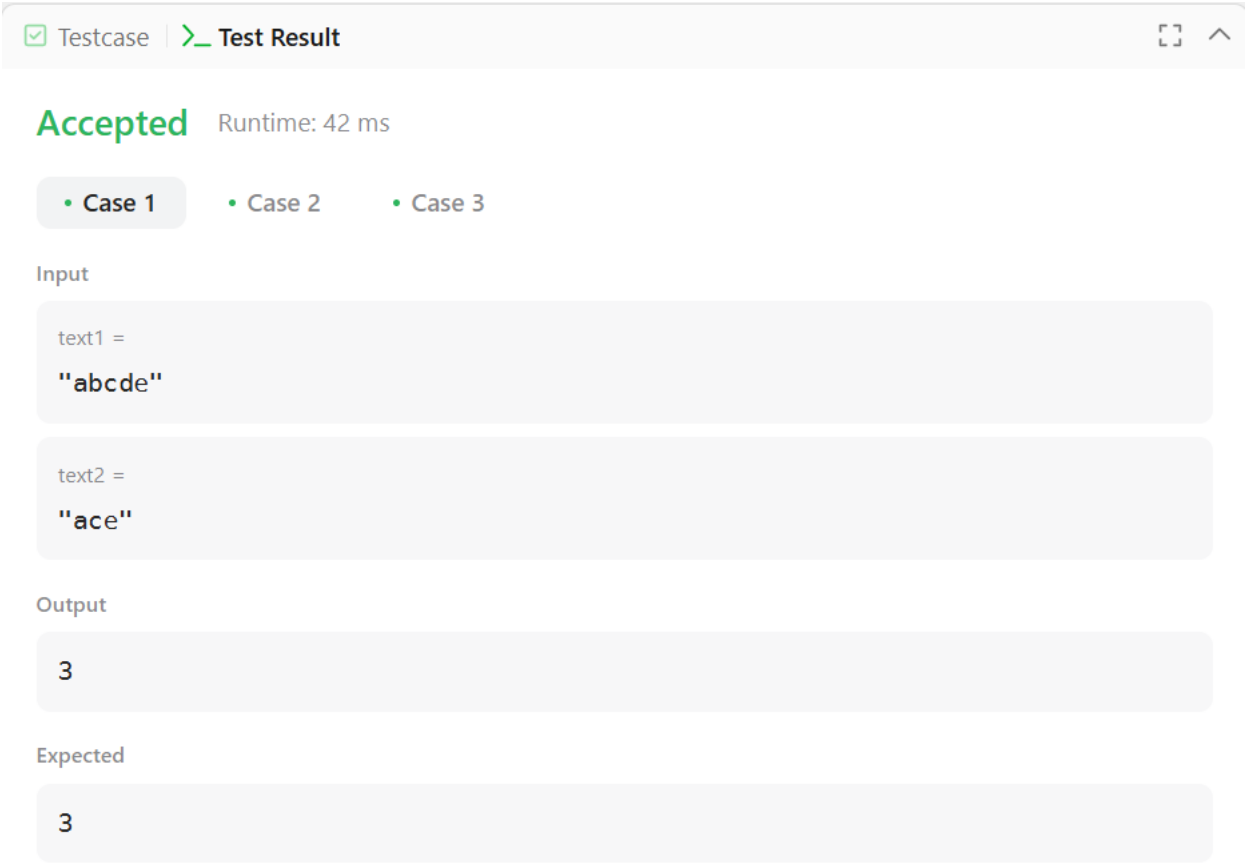
PRACTICAL : 9

AIM : LCS LeetCode problem number 1143

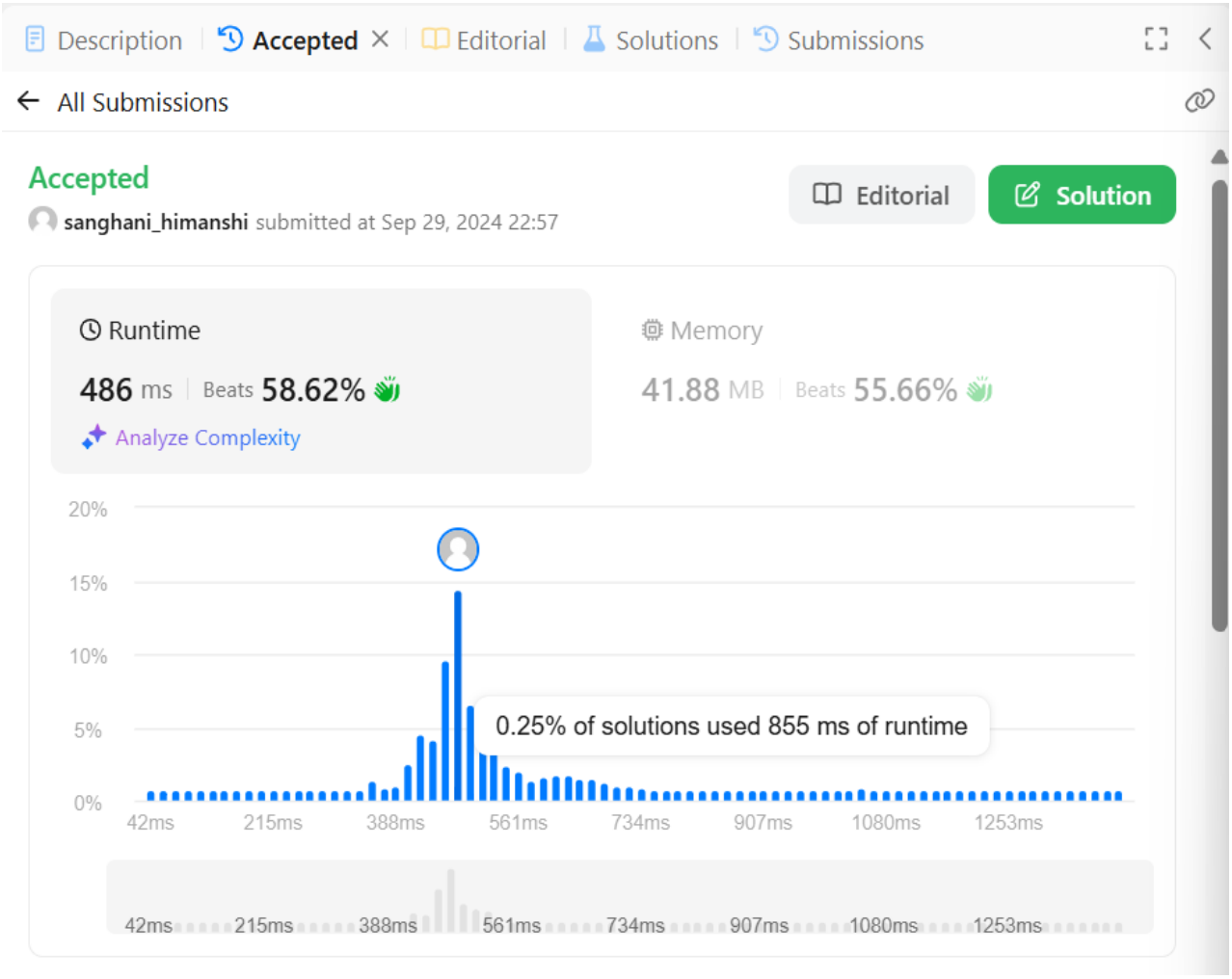
- **Code :**

```
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 :**



Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32



Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

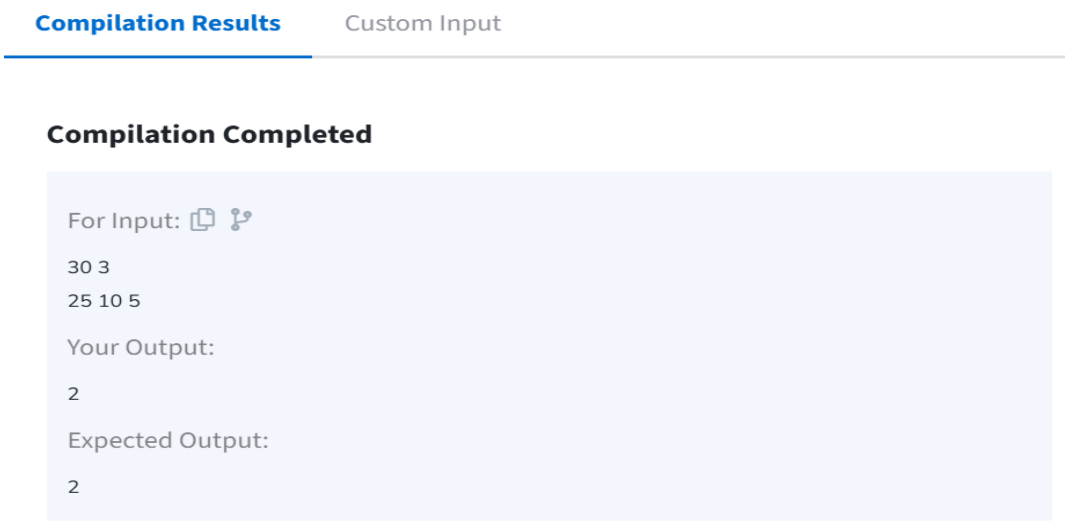
PRACTICAL : 10

AIM : Number of Coins GeeksForGeeks.

- **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]
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 :**



Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32

Name : Sanghani Himanshi
Enrollment No.: 2203031450021
Class : 5B32