WORKSHEET 4

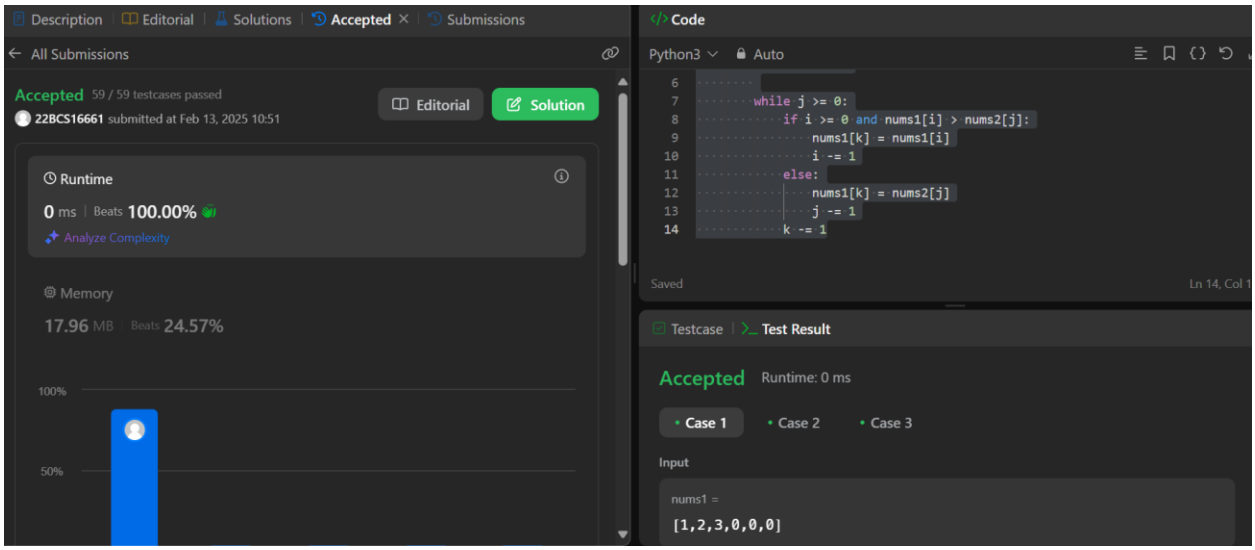
Student Name: Avreet Singh
UID: 22BCS16488
Branch: BE-CSE
Section/Group: 22BCS_NTPP-602-A
Semester: 6th
Date of Performance: 13/01/2025
Subject Name: AP LAB - II
Subject Code: 22CSP-351

1. **Aim:** Reverse You are given two integer arrays nums1 and nums2, sorted in **non-decreasing order**, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.
Merge nums1 and nums2 into a single array sorted in **non-decreasing order**.
The final sorted array should not be returned by the function, but instead be *stored inside the array* nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

2. **Source Code:**

```
class Solution(object):
    def merge(self, nums1, m, nums2, n):
        i = m - 1
        j = n - 1
        k = m + n - 1

        while j >= 0:
            if i >= 0 and nums1[i] > nums2[j]:
                nums1[k] = nums1[i]
                i -= 1
            else:
                nums1[k] = nums2[j]
                j -= 1
            k -= 1
```

3. Screenshots of outputs:



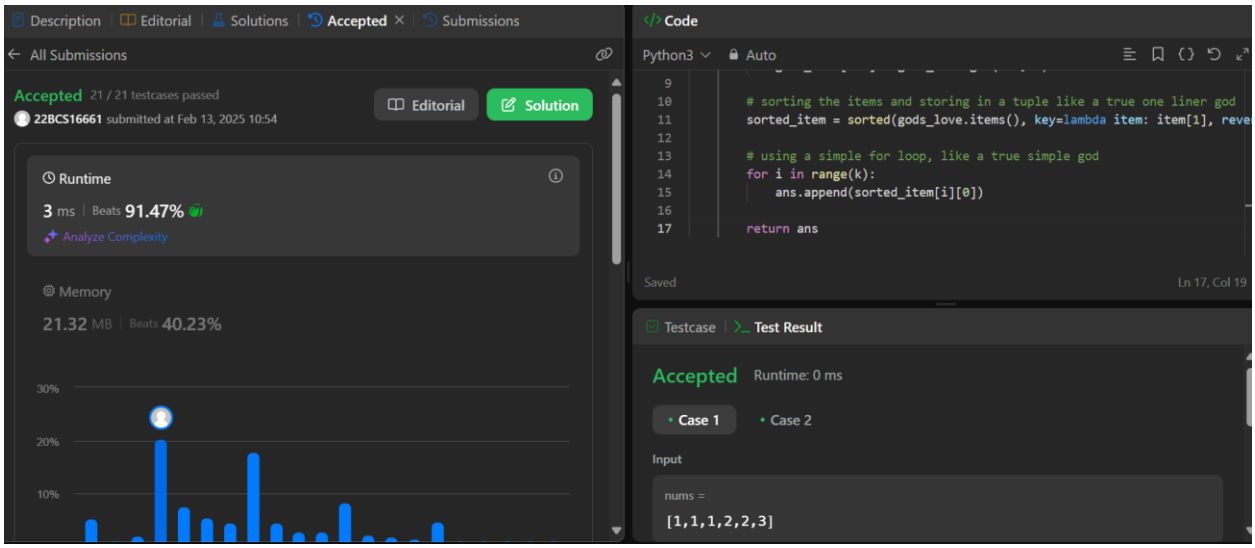
2.

Aim: Given an integer array nums and an integer k, return *the* k *most frequent elements*. You may return the answer in **any order**.

Source Code:

```
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        gods_love = dict()
        ans = list()

        for num in nums:
            # adding freq of elem like a true one liner god
            gods_love[num] = gods_love.get(num, 0) + 1

        # sorting the items and storing in a tuple like a true one liner god
        sorted_item = sorted(gods_love.items(), key=lambda item: item[1], reverse=True)

        # using a simple for loop, like a true simple god
        for i in range(k):
            ans.append(sorted_item[i][0])

        return ans
```

Screenshots of outputs:



3.

Aim: Given an array of intervals where intervals[i] = [$start_i$, end_i], merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

Source Code:

```
class Solution:
def merge(self, intervals: List[List[int]]) -> List[List[int]]:
    intervals.sort(key=lambda x: x[0]) # Sort intervals by start time
    k = 0 # Index for merged intervals

    for i in range(1, len(intervals)):
        if intervals[k][1] >= intervals[i][0]: # Overlap detected
            intervals[k][1] = max(intervals[k][1], intervals[i][1]) # Merge
        else:
            k += 1 # Move to the next position
            intervals[k] = intervals[i] # Replace in-place

    return intervals[:k + 1] # Return only merged intervals
            result.append([x, curr_max_height])
```

4. Screenshots of outputs: