Experiment- 4

Student Name: Sahil **UID: 22BCS10928**

Branch: BE-CSE **Section/Group: KPIT_901(B)**

Semester: 6th **Date of Performance: 21/02/2025**

Subject Name: AP **Subject Code: 22CSH-359**

Problem-1

1. **Aim:** To merge two sorted arrays into a single sorted array efficiently.

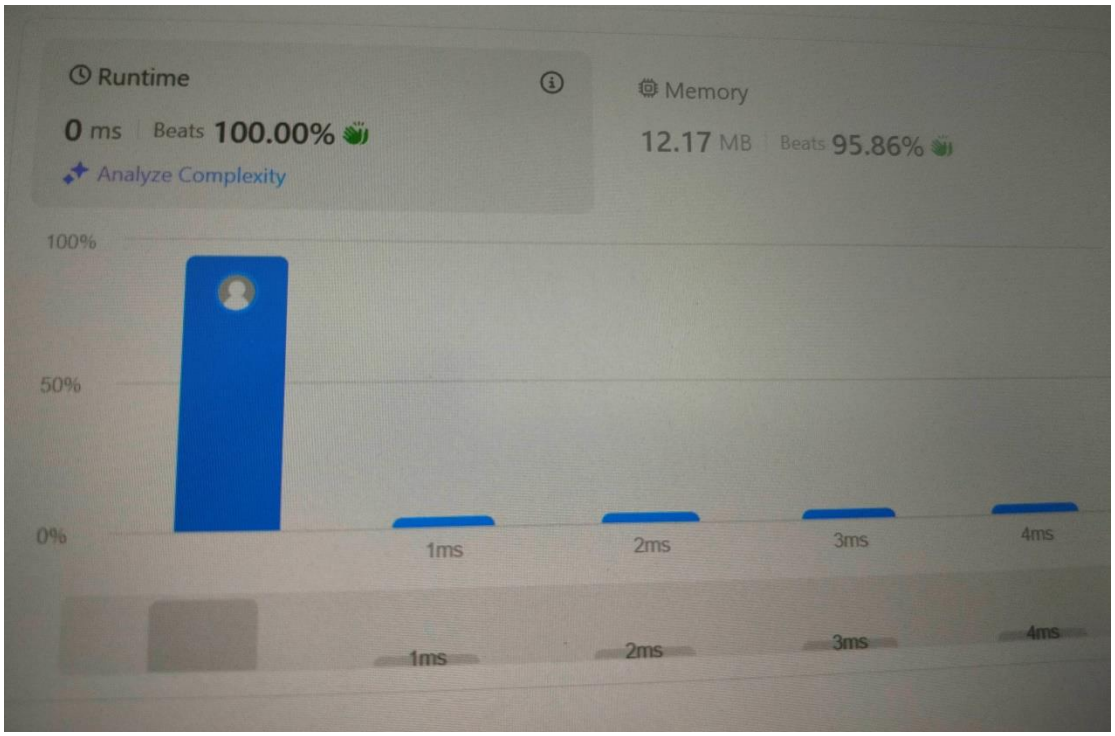
2. Objective:

- Append elements of nums2 to nums1 while maintaining order.
- Use sorting to ensure the merged array remains sorted.

3. Implementation/Code:

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        for (int j = 0, i = m; j<n; j++){
            nums1[i] = nums2[j];
            i++;
        }
        sort(nums1.begin(),nums1.end());
    }
};
```

1. Output



2. Learning Outcomes:

- Understanding how to merge two sorted arrays
- Using in-place merging without extra space
- Applying sorting techniques on merged arrays
- Handling edge cases like empty arrays or different sizes

Problem-2

1. **Aim:** To find the first bad version efficiently using binary search, minimizing the number of API calls.

2. **Objective:**
   Implement a binary search algorithm to locate the first bad version.
   Optimize the search process by reducing the time complexity to **O(log n)**.

3. **Implementation/Code:**

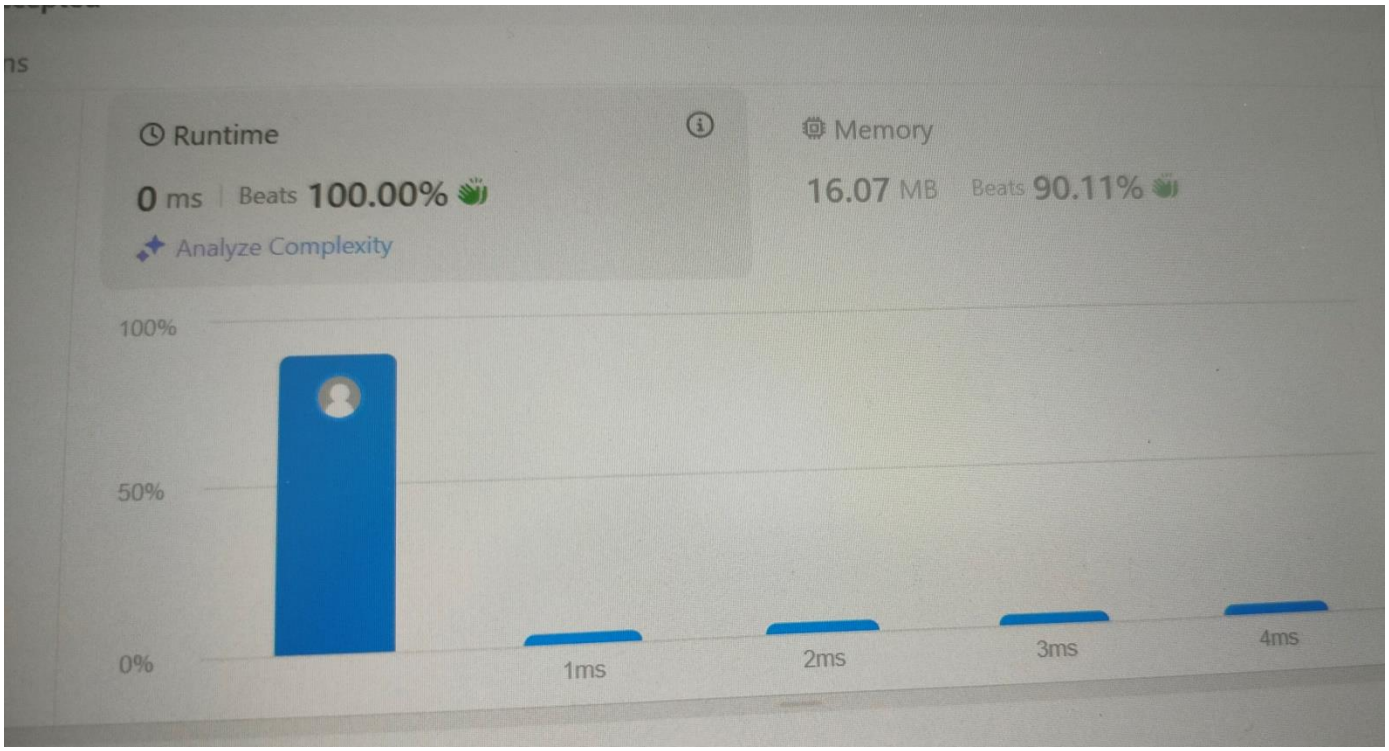
```
class Solution {
public:
    int firstBadVersion(int n) {
        int first = 1;
        int last = n;

        while (first < last) {
            int mid = first + (last - first) / 2;

            if (isBadVersion(mid)) {
                last = mid; // Mid could be the first bad version, so narrow the
                            // range to the left half.
            } else {
                first = mid + 1; // If mid is not bad, the first bad version
                                 // must be after mid.
            }
        }

        return first; // At the end, first will be the first bad version.
    }
};
```

4.Output



5. Learning Outcomes:

- Understanding binary search for efficient searching
- Applying a divide-and-conquer approach to minimize search space
- Optimizing search in a sorted sequence with a condition check
- Handling edge cases like all versions being good or bad