Experiment-4

Student Name: Anupreet Kaur **UID: 22BCS50071**

Branch: BE-CSE **Section/Group: _NTPP_IOT-602-A**

Semester: 6th **Date of Performance: 10/02/2025**

Subject Name: AP Lab **Subject Code: 22CSP-351**

1. Aim: Sorting and Searching.

- Problem 1.2.1: Merge Sort
- Problem 1.2.2: First Bad Version
- Problem 1.2.2: Sort Color

2. Objective:

To understand and implement efficient sorting and searching algorithms for problem-solving.

3. Theory:

Sorting and searching are fundamental operations in computer science.
Merge Sort is a divide-and-conquer sorting algorithm with a time complexity of O(n log n).
The First Bad Version problem is solved using binary search to efficiently locate a faulty version in O(log n) time.
Sort Colors (Dutch National Flag problem) sorts an array of 0s, 1s, and 2s using a two-pointer approach for optimal O(n) performance.

4. Code:

Merge Sort

```
class Solution {
 public void merge(int[] nums1, int m, int[] nums2, int n) {
    int i = m - 1;
    int j = n - 1;
    int k = m + n - 1;

    while (i >= 0 && j >= 0) {
       if (nums1[i] > nums2[j]) {
          nums1[k--] = nums1[i--];
       } else {
          nums1[k--] = nums2[j--];
       }
    }

    while (j >= 0) {
```

```
        nums1[k--] = nums2[j--];
    }
  }
}
```

First Bad Version

```
public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        int left = 1, right = n;

        while (left < right) {
            int mid = left + (right - left) / 2;

            if (isBadVersion(mid)) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }

        return left;
    }
   }
```

Sort Color

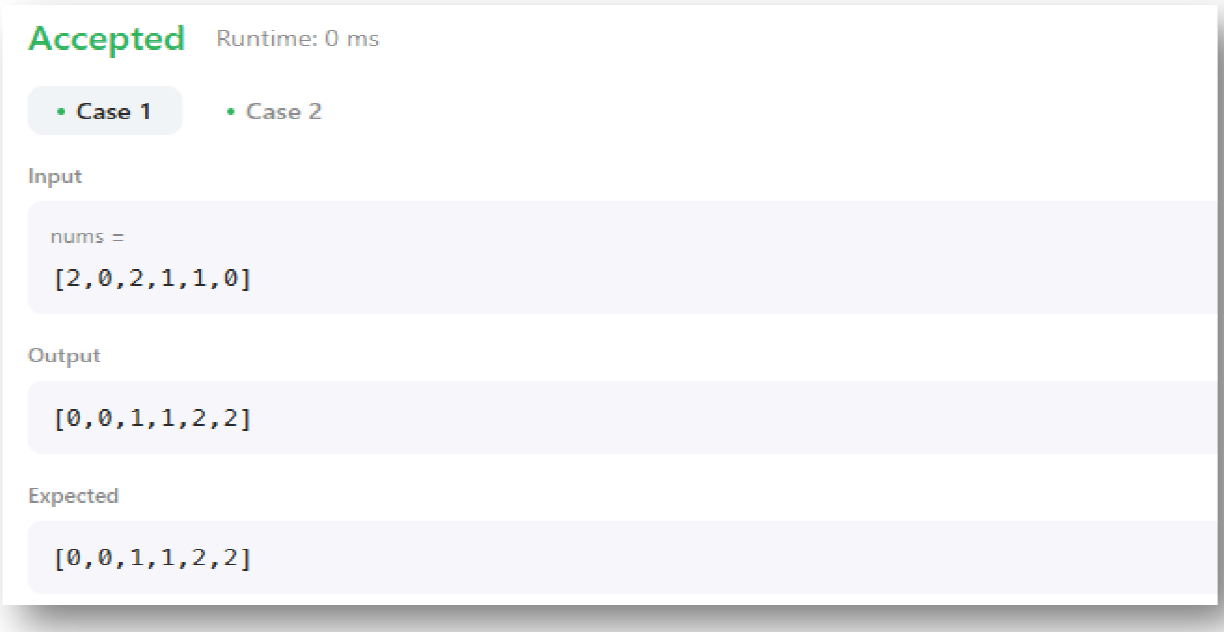
```
class Solution {
    public void sortColors(int[] nums) {
        int left = 0, right = nums.length - 1, i = 0;

        while (i <= right) {
            if (nums[i] == 0) {
                swap(nums, i, left);
                left++;
                i++;
            } else if (nums[i] == 2) {
                swap(nums, i, right);
                right--;
            } else {
                i++; // nums[i] == 1, just move forward
            }
        }
    }
    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
}
```

6. Output:


Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
nums1 =
[1,2,3,0,0,0]
m =
3
nums2 =
[2,5,6]
n =
3
Output
[1,2,2,3,5,6]
Expected
[1,2,2,3,5,6]


Accepted
Runtime: 1 ms
Case 1
Case 2
Input
n =
5
bad =
4
Output
4



7. Learning Outcomes:

- Understand and implement the Merge Sort algorithm.
- Apply binary search to solve decision problems efficiently.
- Optimize sorting problems using two-pointer techniques.