DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Experiment - 4

Student Name: Masud Alom
UID: **22BCS16095**
Branch: **BE - CSE**
Section/Group: **NTPP 602 A**
Semester: **06**
Sub Code: **22CSP-351**
Subject Name: **Advanced Programming Lab - 2**
Date:13/02/2025

Problem - 1

Aim - The aim is to merge two sorted integer arrays, `nums1` and `nums2`, into a single sorted array in non-decreasing order. The solution should utilize the given integers `m` and `n`, representing the number of valid elements in `nums1` and `nums2` respectively, and store the merged result in `nums1`...

Objective - You are provided with two integer arrays, `nums1` and `nums2`, both sorted in nondecreasing order. The length of `nums1` is at least the sum of the lengths of `nums1` and `nums2`, as the unused space in `nums1` is meant to accommodate the elements of `nums2`. The task is to merge the two arrays efficiently, modifying `nums1` in-place, so that the final array remains sorted. The challenge involves using a two-pointer technique or similar algorithmic approach to achieve this in optimal time and space complexity.

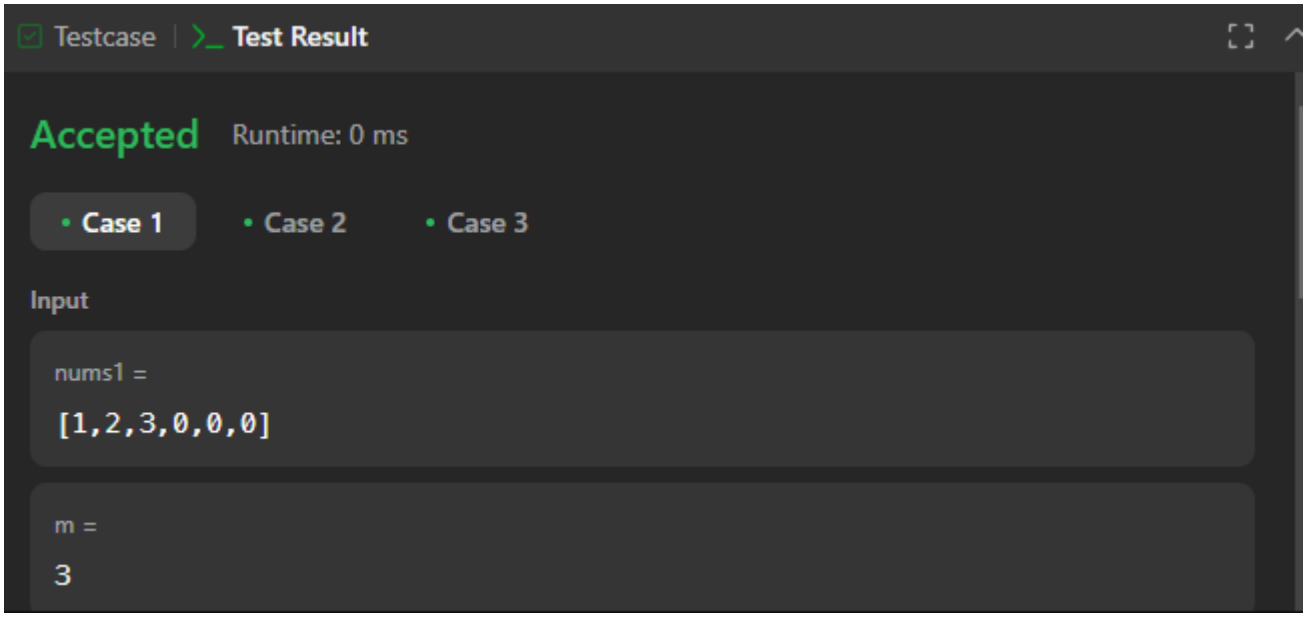
Code –

```
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];
        i--;
      } else {
        nums1[k] = nums2[j];
        j--;
      }
      k--;
    }
```

```
        // If there are remaining elements in nums2, copy them
        while (j >= 0) {
            nums1[k] = nums2[j];
            j--;
            k--;
        }
    }
}
```

Output -



Time Complexity : O(n)(Log(N))

Space Complexity - O(1)

Problem - 2

Aim - The aim is to sort an array of integers representing colors (red, white, and blue) in-place, where 0, 1, and 2 represent red, white, and blue, respectively, ensuring that the colors are arranged in the order red, white, and blue.

Objective - The objective is to efficiently sort the array without using built-in sorting functions, employing an optimal algorithm, such as the Dutch National Flag algorithm, to achieve the desired arrangement with minimal space and time complexity.

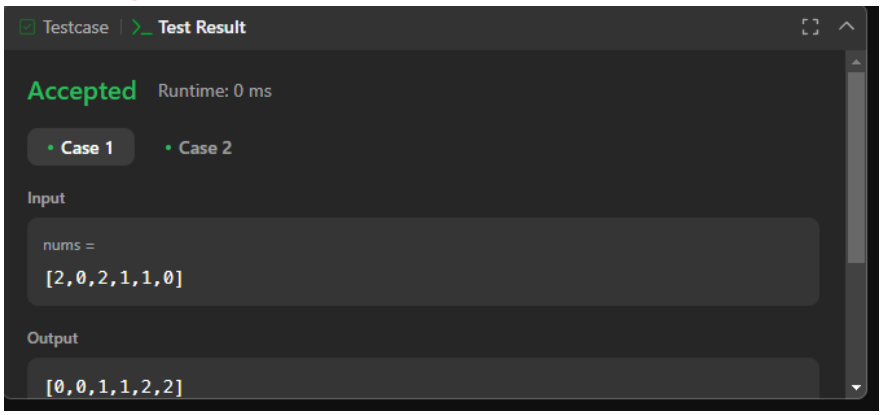
Code -

```
class Solution {
    public void sortColors(int[] nums) {
        int low = 0, mid = 0, high = nums.length - 1;

        while (mid <= high) {
            if (nums[mid] == 0) { // Swap 0 to the front
                swap(nums, low, mid);
                low++;
                mid++;
            } else if (nums[mid] == 1) { // Keep 1 in the middle
                mid++;
            } else { // Swap 2 to the end
                swap(nums, mid, high);
                high--;
            }
        }
    }

    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
}
```

Output -



Time Complexity:- O(N log N). **Space Complexity:-** O(1)

Learning Outcomes -

1. **Understanding Data Structures:**
   Learn to manipulate arrays efficiently by understanding how to work with pointers or indices for sorting and merging operations.
2. **Problem-Solving Skills:**
   Enhance problem-solving skills through the application of specific algorithms for merging arrays and sorting based on custom constraints.
3. **Algorithmic Thinking:**
   Develop algorithmic thinking by selecting and implementing efficient algorithms, such as bitwise operations or the Dutch National Flag algorithm for sorting.