Experiment- 4A

Student Name: Deepanshu **UID: 22BCS16692**

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

Semester:6TH **Date of Performance:10/02/25**

Subject Name: AP Lab-2 **Subject Code: 22CSH-352**

1. TITLE:

Merge Sorted Array

2. AIM:

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.

3. Algorithm

- Start **merging from the end** of `nums1`, since it has extra space at the end.
- Use **two pointers**.
- Compare elements from `nums1` and `nums2` **from the end**.
- If elements remain in `nums2`, copy them to `nums1`.

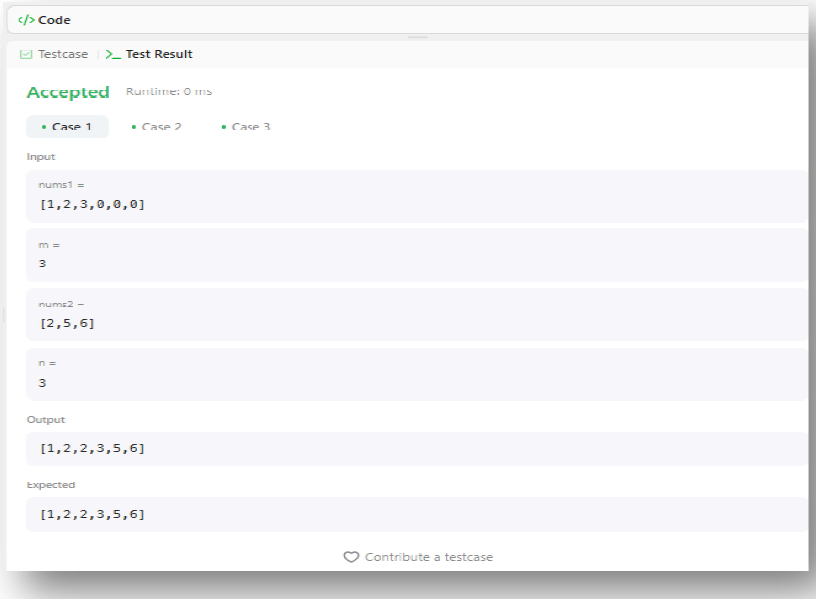
Implemetation/Code

```
class Solution:
def merge(self, nums1, m, nums2, n):
p1, p2, p = m - 1, n - 1, m + n - 1
while p1 >= 0 and p2 >= 0:
if nums1[p1] > nums2[p2]:
nums1[p] = nums1[p1]
p1 -= 1
else:
nums1[p] = nums2[p2]
p2 -= 1
p -= 1

while p2 >= 0:
nums1[p] = nums2[p2]
p2 -= 1
```

```
p -= 1
```

Output:



Time Complexity : O(m + n)

Space Complexity : O(1)

Learning Outcomes:-

- Instead of merging in a new array, we efficiently **merge from the back**.
- This avoids unnecessary shifting of elements.

Experiment - 4B

Student Name: Deepanshu **UID: 22BCS16443**

Branch:BE-CSE **Section/Group: NTPP- 602(A)**

Semester:6TH **Date of Performance:10/02/25**

Subject Name: AP Lab-2 **Subject Code: 22CSH-352**

1. TITLE:

To Sort colors.

2. AIM:

Given an array nums with n objects colored red, white, or blue, sort them **in-place** so that objects of the same color are adjacent, with the colors in the order red, white, and blue

3. Algorithm

- Create a **queue** and enqueue the root node..
- Create an empty **result list** to store the final level order traversal.
- Append the `level` list to `result`.
- This list contains all levels of the binary tree.

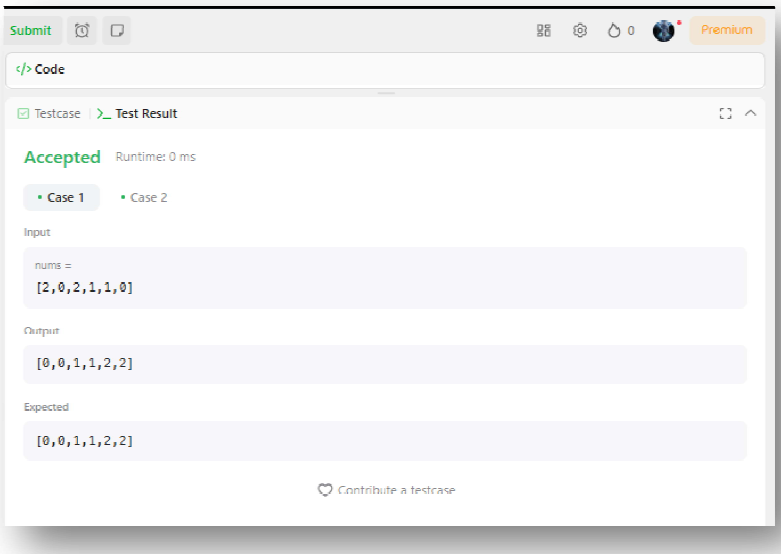
Implemetation/Code:

```
class Solution {
public:
void sortColors(vector<int>& nums) {
int zero = -1;
int one = -1;
int two = -1;

for (const int num : nums)
 if (num == 0) {
 nums[++two] = 2;
 nums[++one] = 1;
 nums[++zero] = 0;
 } else if (num == 1) {
 nums[++two] = 2;
 nums[++one] = 1;
 } else {
 nums[++two] = 2;
```

```
        }
    }
};
```

Output:



Time Complexity : O(N)

Space Complexity : O(1)

Learning Outcomes:-

- The **Dutch National Flag Algorithm** sorts three categories in **one pass** using a **three-pointer approach**.
- This technique is useful when memory constraints prevent the use of extra arrays.