Assignment-5

Student Name: Lakhan Singh
UID: 22BCS12194
Branch: BE-CSE
Section/Group: 608-B
Semester: 6th
Date of Performance: 03/04/25
Subject Name: Advance Programming Lab-2
Subject Code: 22CSP-351

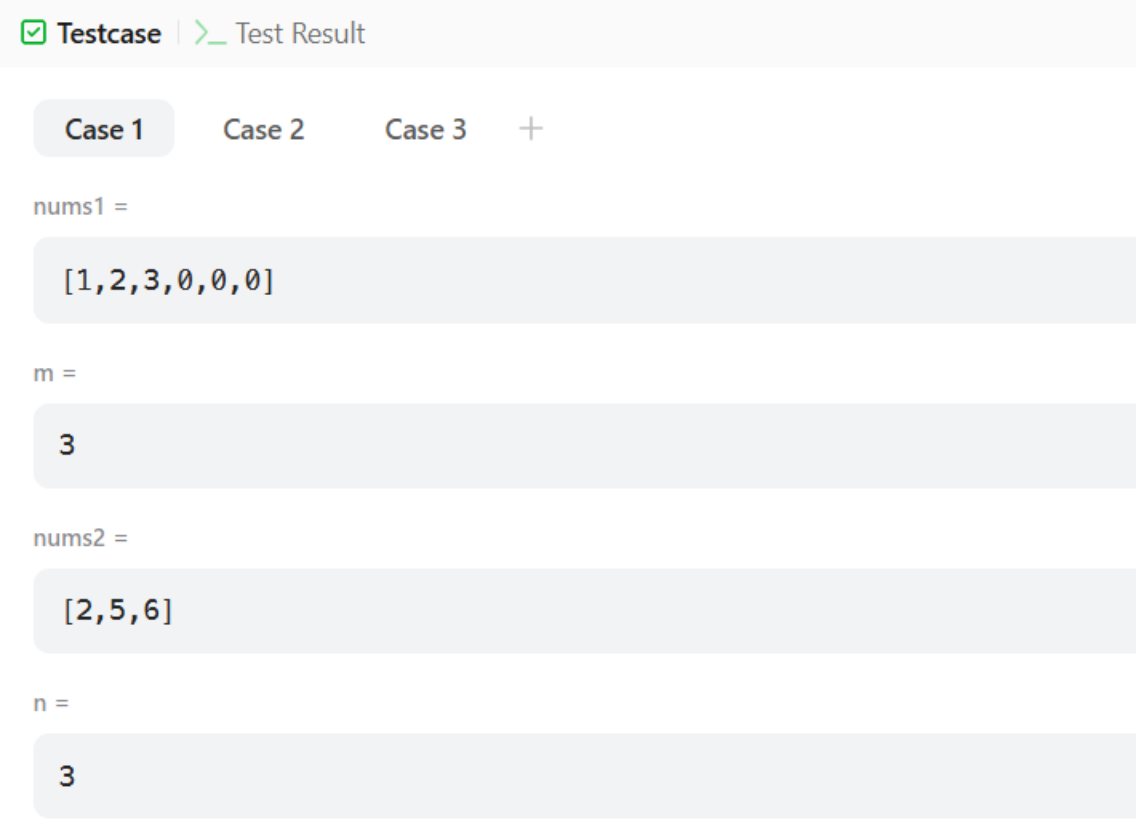
1. Aim: Merge Sorted Array.

Code:

```
class Solution {
public:
    void merge(std::vector<int>& nums1, int m, std::vector<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--];
        }
    }
```

};

Output:



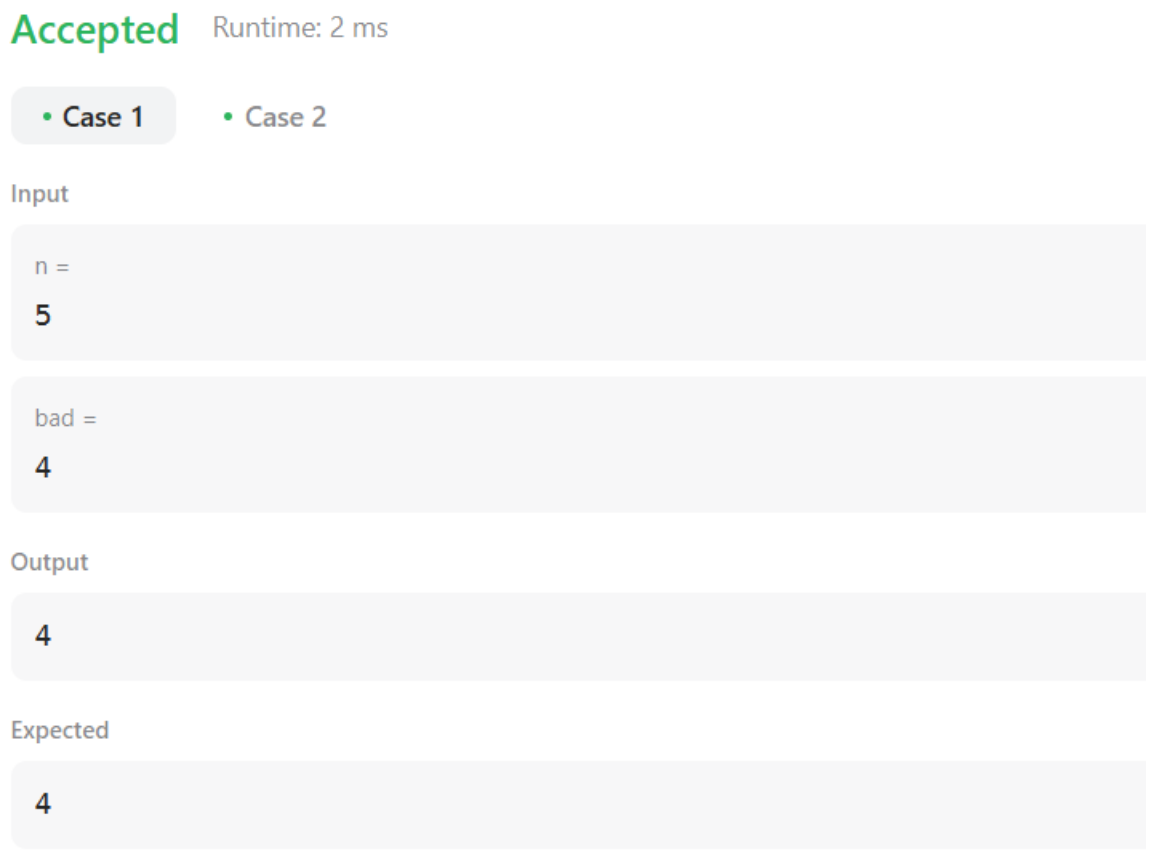
2. Aim: First Bad Version.

Code:

```
class Solution {
public:
    int firstBadVersion(int n) {
        int left = 1, right = n;
        while (left < right) {
            int mid = left + (right - left) / 2; // Prevents integer overflow
```

```
            if (isBadVersion(mid)) {
                right = mid; // Move left to find the first bad version
            } else {
                left = mid + 1; // Ignore the good versions
            }
        }
        return left; // The first bad version
    }
};
```

Output:



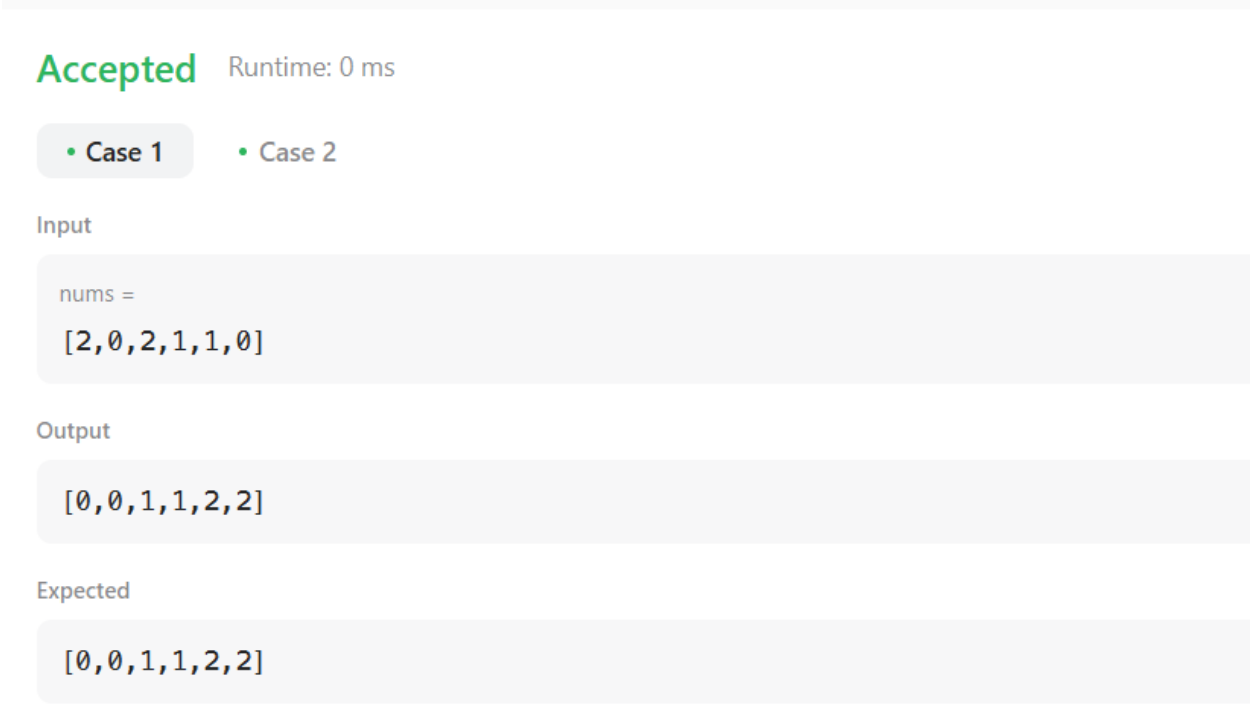
3. Aim: Sort Colors.

Code:

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int low = 0, mid = 0, high = nums.size() - 1;


        while (mid <= high) {
            if (nums[mid] == 0) {
                swap(nums[mid], nums[low]);
                low++;
                mid++;
            } else if (nums[mid] == 1) {
                mid++;
            } else {  // nums[mid] == 2
                swap(nums[mid], nums[high]);
                high--;
            }
        }
    }
};
```

Output:



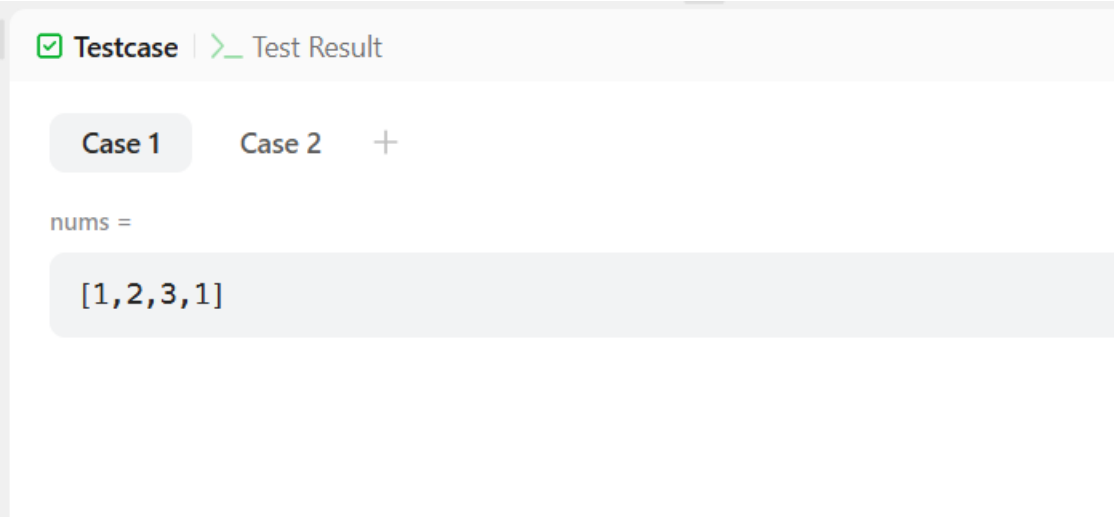
4. Aim: Find Peak Element.

Code:

```
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int left = 0, right = nums.size() - 1;
        while (left < right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] > nums[mid + 1]) {
                right = mid; // Move towards the peak
            } else {
```

```
            left = mid + 1; // Move right
        }
    }
    return left; // Peak index
  }
};
```

Output:



5. Aim: Median of Two Sorted Arrays.

Code:

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        // Ensure nums1 is the smaller array for efficient binary search
```

```
        if (nums1.size() > nums2.size()) {
            return findMedianSortedArrays(nums2, nums1);
        }
        int m = nums1.size(), n = nums2.size();
        int left = 0, right = m, halfLen = (m + n + 1) / 2;
        while (left <= right) {
            int i = (left + right) / 2;
            int j = halfLen - i;
            int nums1Left = (i == 0) ? INT_MIN : nums1[i - 1];
            int nums1Right = (i == m) ? INT_MAX : nums1[i];
            int nums2Left = (j == 0) ? INT_MIN : nums2[j - 1];
            int nums2Right = (j == n) ? INT_MAX : nums2[j];
            if (nums1Left <= nums2Right && nums2Left <= nums1Right) {
                // Odd total length
                if ((m + n) % 2 == 1) {
                    return max(nums1Left, nums2Left);
                }
                return (max(nums1Left, nums2Left) + min(nums1Right, nums2Right)) /
2.0;
            }
            else if (nums1Left > nums2Right) {
                right = i - 1; // Move left
            }
            else {
```

```
            left = i + 1; // Move right
        }
    }
    return 0.0; // Should never reach here
  }
};
```

Output: