Experiment 5

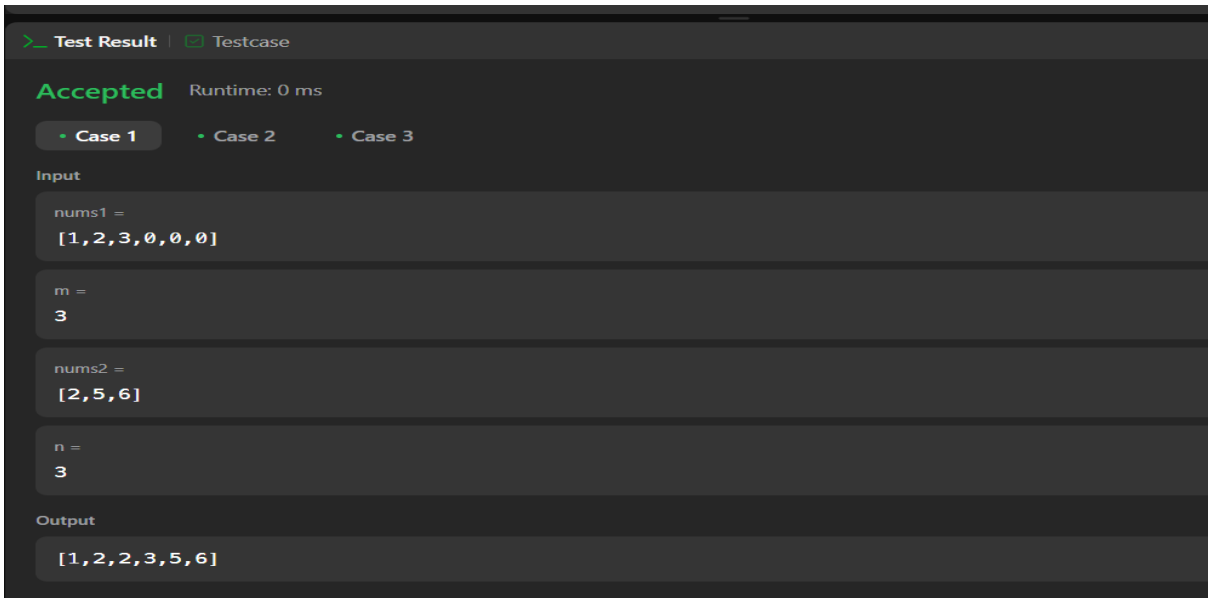
Ques1) Merge Sorted Array -- https://leetcode.com/problems/merge-sorted-array/

Code Submit Ctrl Enter

Java Auto

```java
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int i = m - 1; // Last element of nums1's actual values
        int j = n - 1; // Last element of nums2
        int k = m + n - 1; // Last position of nums1

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

        // Copy remaining elements of nums2 if any
        while (j >= 0) {
            nums1[k--] = nums2[j--];
        }
    }
    }
```

Output:



Ques2) First Bad Version -- https://leetcode.com/problems/first-bad-version/

Ans:

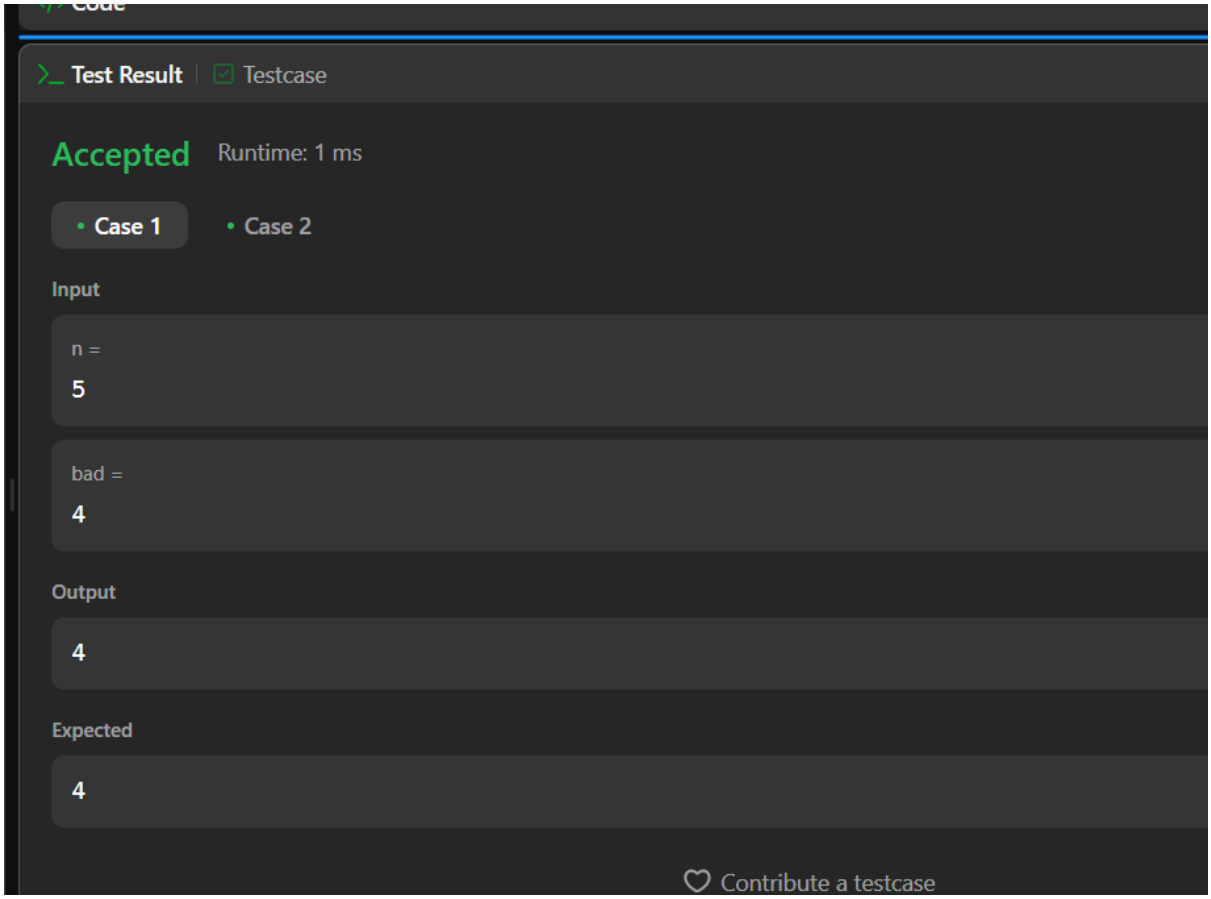
Code

Java Auto

```java
/* The isBadVersion API is defined in the parent class VersionControl.
      boolean isBadVersion(int 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; // Avoids overflow
            if (isBadVersion(mid)) {
                right = mid; // Narrow down to the left half
            } else {
                left = mid + 1; // Narrow down to the right half
            }
        }
        return left; // First bad version
    }
```

Saved

Output:



Ques3) Sort Colors -- https://leetcode.com/problems/sort-colors/

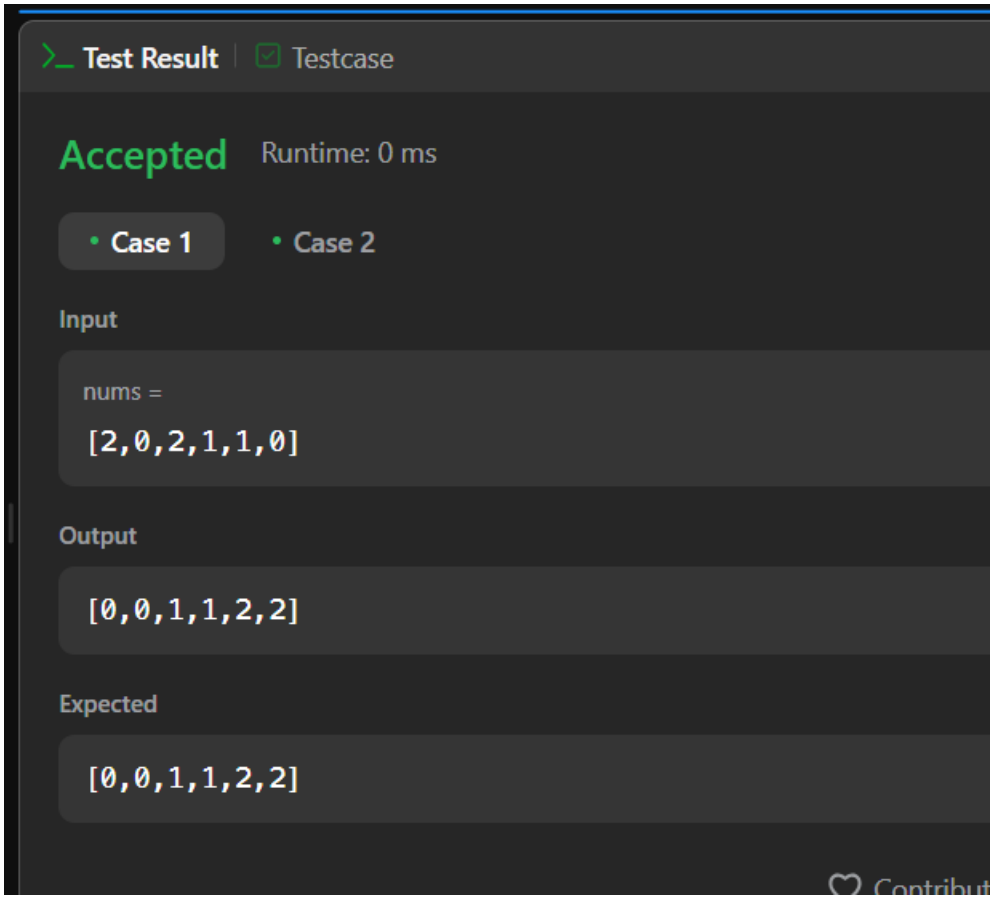
Ans:

Code

Java Auto

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

        while (mid <= high) {
            if (nums[mid] == 0) { // Swap nums[mid] and nums[low]
                int temp = nums[low];
                nums[low] = nums[mid];
                nums[mid] = temp;
                low++;
                mid++;
            } else if (nums[mid] == 1) { // Keep mid in place for white (1)
                mid++;
            } else { // Swap nums[mid] and nums[high]
                int temp = nums[mid];
                nums[mid] = nums[high];
                nums[high] = temp;
                high--;
            }
        }
    }
}

```

Output:



Ques4) Median of Two Sorted Arrays -- https://leetcode.com/problems/median-of-two-sorted-arrays/

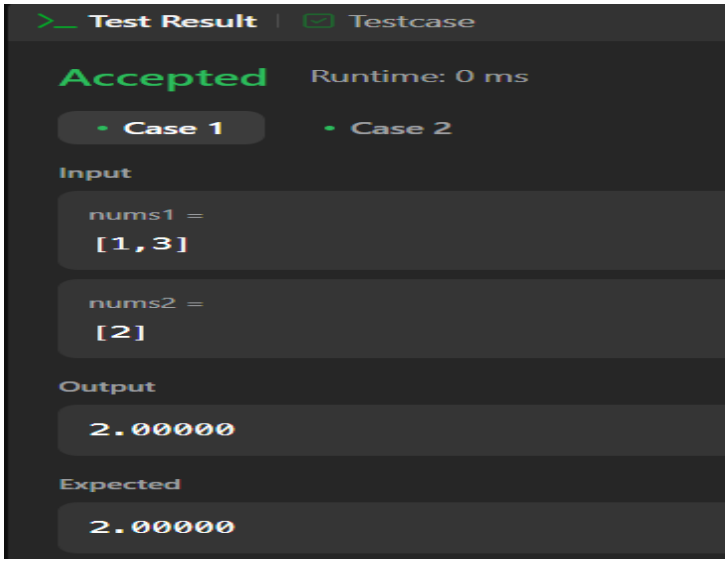
Ans:

Code

Java Auto

```java
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        // Ensure nums1 is the smaller array for optimized binary search
        if (nums1.length > nums2.length) {
            return findMedianSortedArrays(nums2, nums1);
        }

        int m = nums1.length, n = nums2.length;
        int low = 0, high = m;

        while (low <= high) {
            int partitionX = (low + high) / 2;
            int partitionY = (m + n + 1) / 2 - partitionX;

            int maxLeftX = (partitionX == 0) ? Integer.MIN_VALUE : nums1[partitionX - 1];
            int minRightX = (partitionX == m) ? Integer.MAX_VALUE : nums1[partitionX];

            int maxLeftY = (partitionY == 0) ? Integer.MIN_VALUE : nums2[partitionY - 1];
            int minRightY = (partitionY == n) ? Integer.MAX_VALUE : nums2[partitionY];

            if (maxLeftX <= minRightY && maxLeftY <= minRightX) {
                // Even length case
                if ((m + n) % 2 == 0) {
                    return (Math.max(maxLeftX, maxLeftY) + Math.min(minRightX, minRightY)) / 2.0;
                }
                // Odd length case
                else {
                    return Math.max(maxLeftX, maxLeftY);
                }
            } else if (maxLeftX > minRightY) {
                high = partitionX - 1; // Move left
            } else {
                low = partitionX + 1; // Move right
            }
        }

        throw new IllegalArgumentException("Input arrays are not sorted correctly");
    }
}
```

Output:



Ques5) Top K frequent elements -- https://leetcode.com/problems/top-k-frequent-elements/

Ans:

Code

Java Auto

```java
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> freqMap = new HashMap<>();
        for (int num : nums) {
            freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
        }
        PriorityQueue<Integer> minHeap = new PriorityQueue<>(Comparator.comparingInt(freqMap::get));

        for (int num : freqMap.keySet()) {
            minHeap.offer(num);
            if (minHeap.size() > k) {
                minHeap.poll(); // Remove the least frequent element
            }
        }
        int[] result = new int[k];
        for (int i = 0; i < k; i++) {
            result[i] = minHeap.poll();
        }
        return result;
    }
}
```

Output: