AP Assignment – 05

Name – Akshat Dimri

UID – 22BCS14302

SECTION - 608 - B

Q1- 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.

Merge nums1 and nums2 into a single array sorted in **non-decreasing order**.

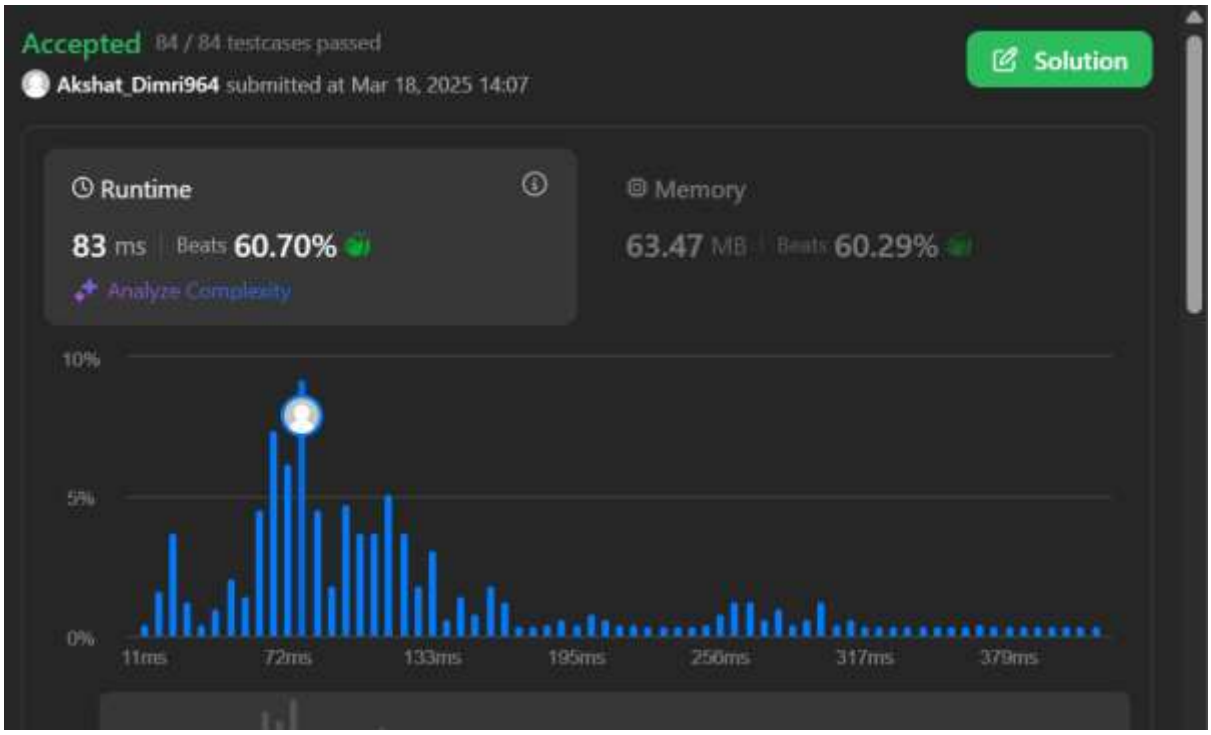
The final sorted array should not be returned by the function, but instead be *stored inside the array* nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

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

```
        while (j >= 0) {
            nums1[k--] = nums2[j--];
        }
    }
}
```



Q2 - You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

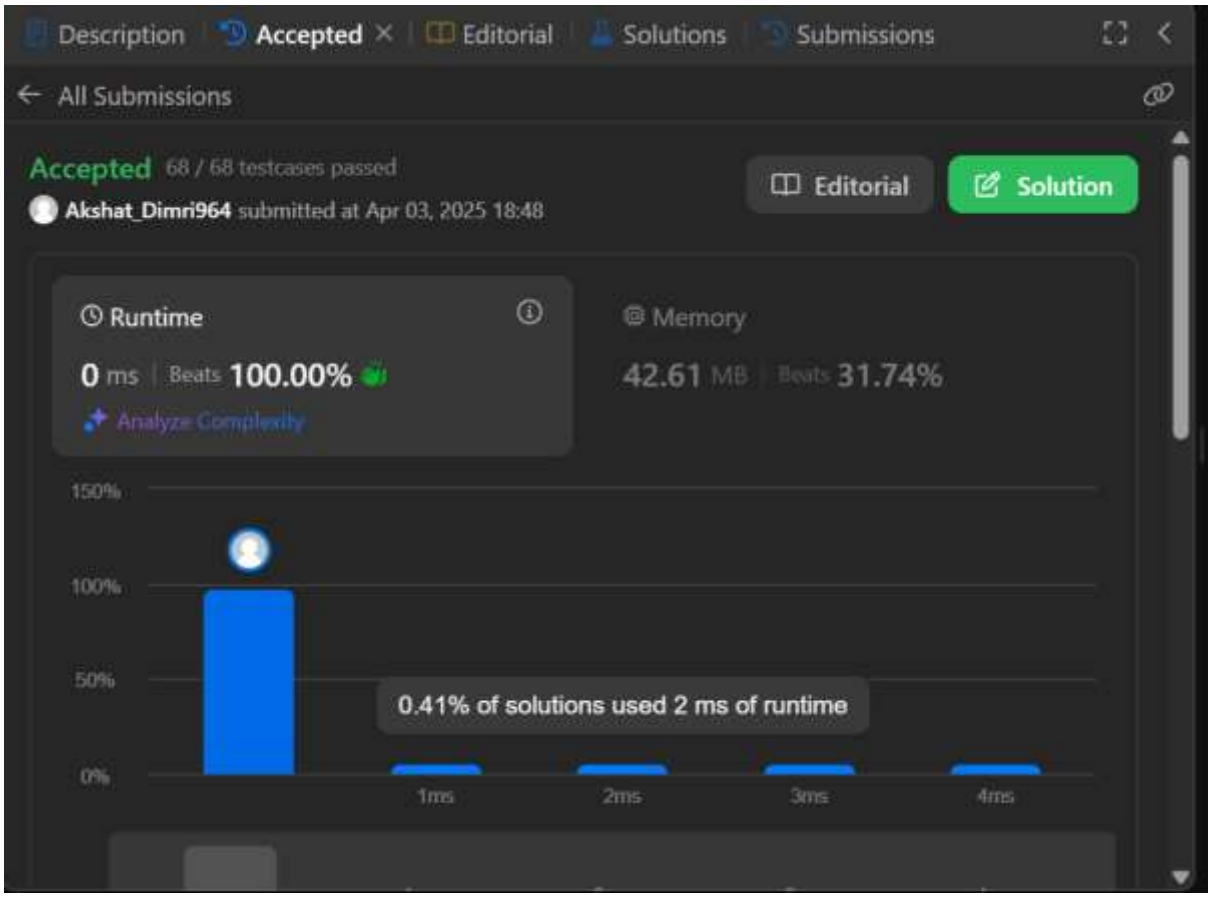
Code

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

        while (left < right) {
```

```
            int mid = left + (right - left) / 2;
            if (isBadVersion(mid)) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }

        return left;
    }
}
```



Q3 - 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.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

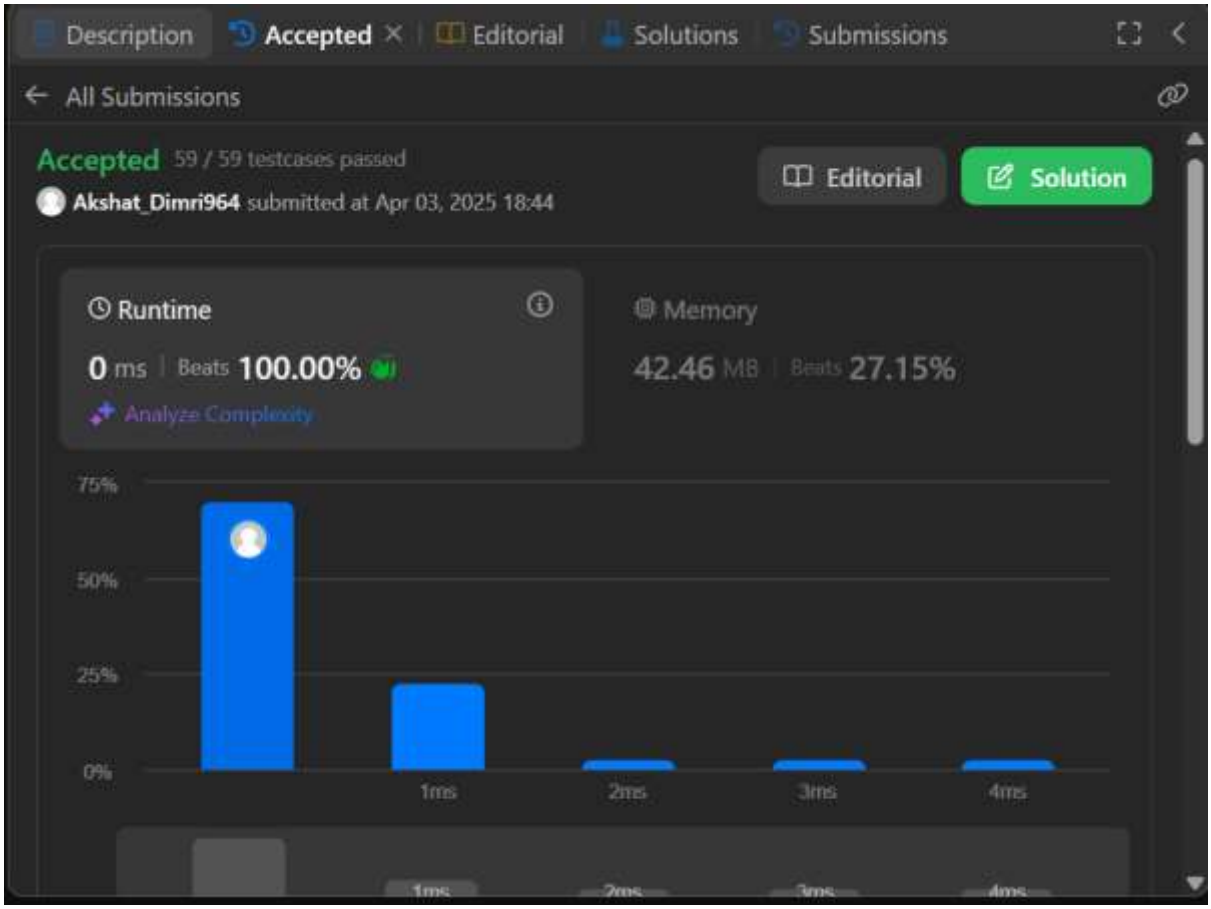
You must solve this problem without using the library's sort function.

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(nums, low, mid);
                low++;
                mid++;
            } else if (nums[mid] == 1) {
                mid++;
            } else { // nums[mid] == 2
                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;
    }
}
```



Q4 - A peak element is an element that is strictly greater than its neighbors.

Given a **0-indexed** integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in O(log n) time.

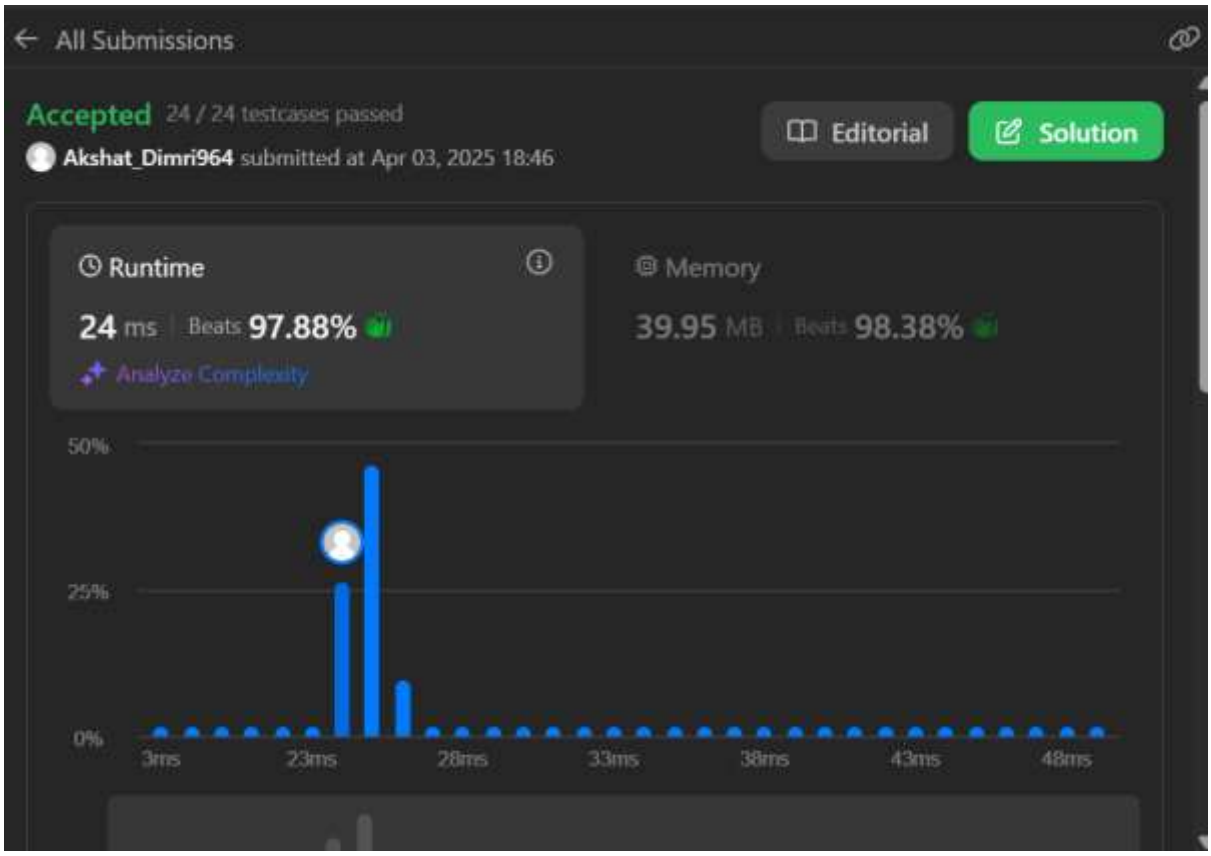
Code

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

        while (left < right) {
```

```
            int mid = left + (right - left) / 2;
            if (nums[mid] > nums[mid + 1]) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }

        return left;
    }
}
```



Q5 - Given two sorted arrays nums1 and nums2 of size m and n respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

Code

```
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        if (nums1.length > nums2.length) {
            return findMedianSortedArrays(nums2, nums1);
        }

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

        while (low <= high) {
            int partition1 = (low + high) / 2;
            int partition2 = (m + n + 1) / 2 - partition1;

            int maxLeft1 = (partition1 == 0) ? Integer.MIN_VALUE : nums1[partition1 - 1];
            int minRight1 = (partition1 == m) ? Integer.MAX_VALUE : nums1[partition1];

            int maxLeft2 = (partition2 == 0) ? Integer.MIN_VALUE : nums2[partition2 - 1];
            int minRight2 = (partition2 == n) ? Integer.MAX_VALUE : nums2[partition2];

            if (maxLeft1 <= minRight2 && maxLeft2 <= minRight1) {
                if ((m + n) % 2 == 0) {
                    return (Math.max(maxLeft1, maxLeft2) + Math.min(minRight1, minRight2)) / 2.0;
                } else {
                    return Math.max(maxLeft1, maxLeft2);
                }
            } else if (maxLeft1 > minRight2) {
                high = partition1 - 1;
            } else {
```

```
            low = partition1 + 1;
        }
    }

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