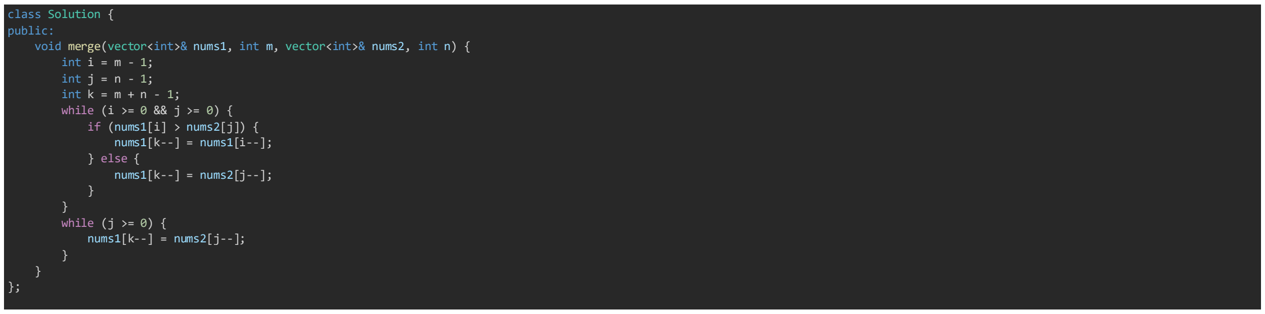
Name : Kumad Mahajan

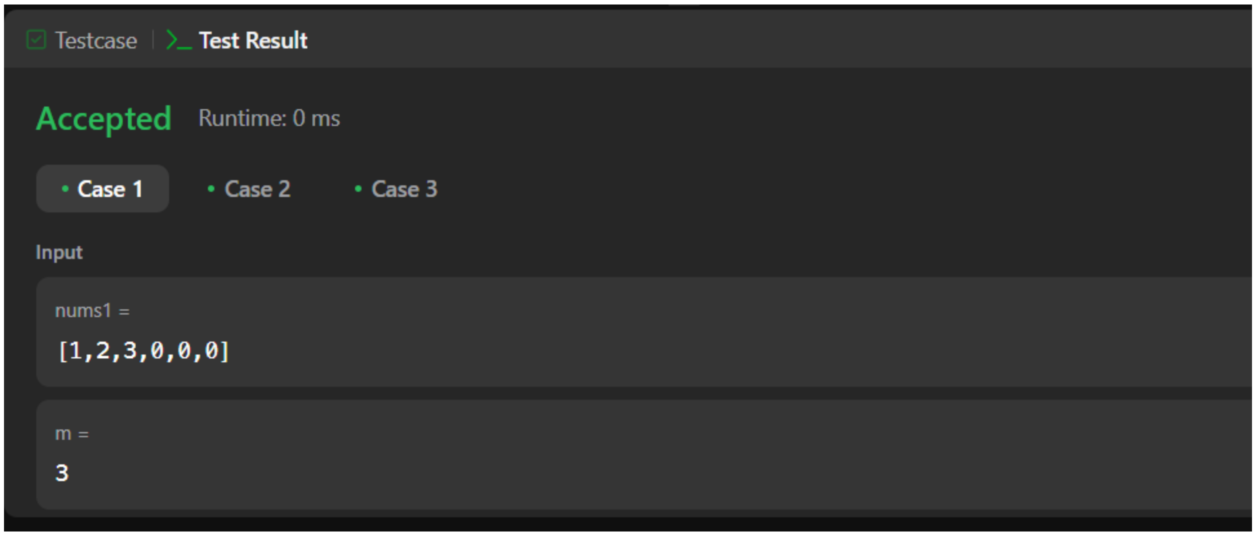
UID: 22BCS13821

1. Merge Sorted array



```cpp
class Solution {
public:
    void merge(vector<int>& nums1, int m, 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:



Accepted Runtime: 0 ms

Input

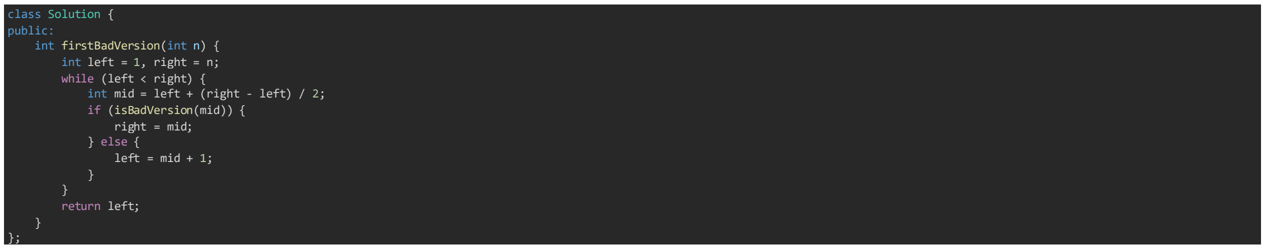
nums1 =

```
[1,2,3,0,0,0]
```

m =

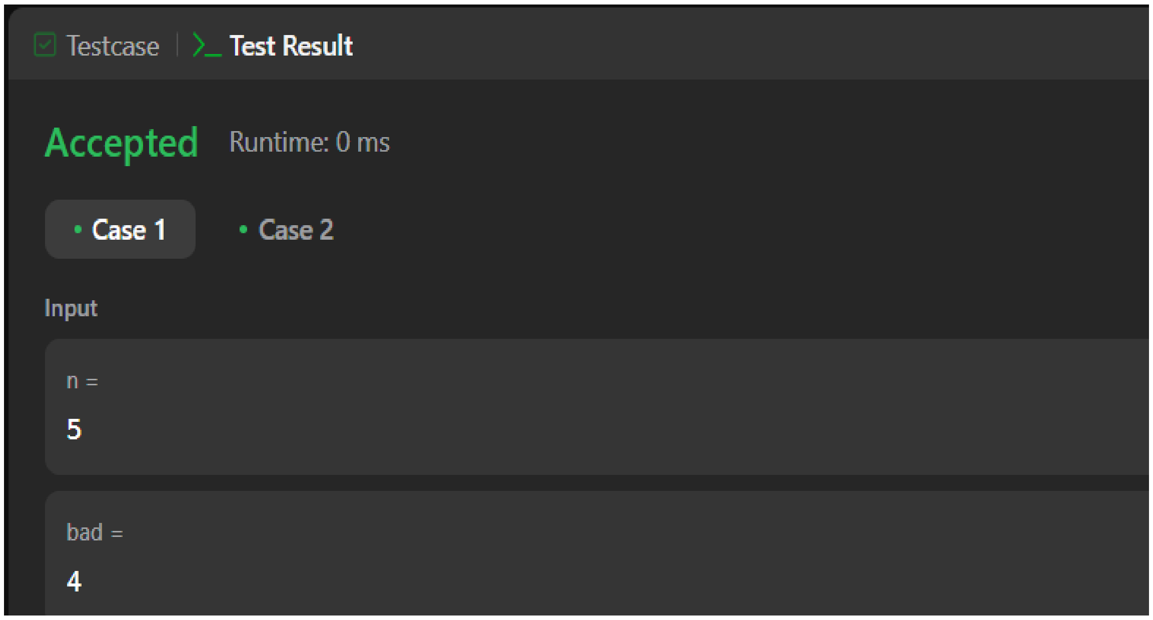
```
3
```

2. First bad Version



```cpp
class Solution {
public:
    int firstBadVersion(int n) {
        int left = 1, right = n;
        while (left < right) {
            int mid = left + (right - left) / 2;
            if (isBadVersion(mid)) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }
        return left;
    }
};
```

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

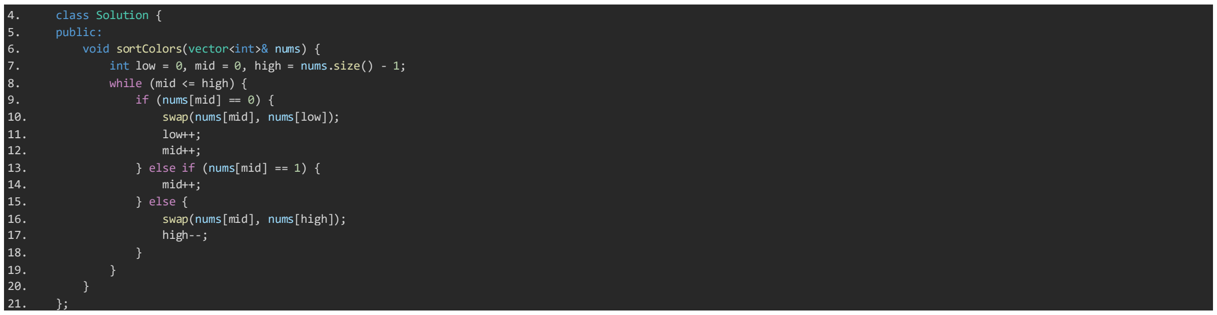
n =

5

bad =

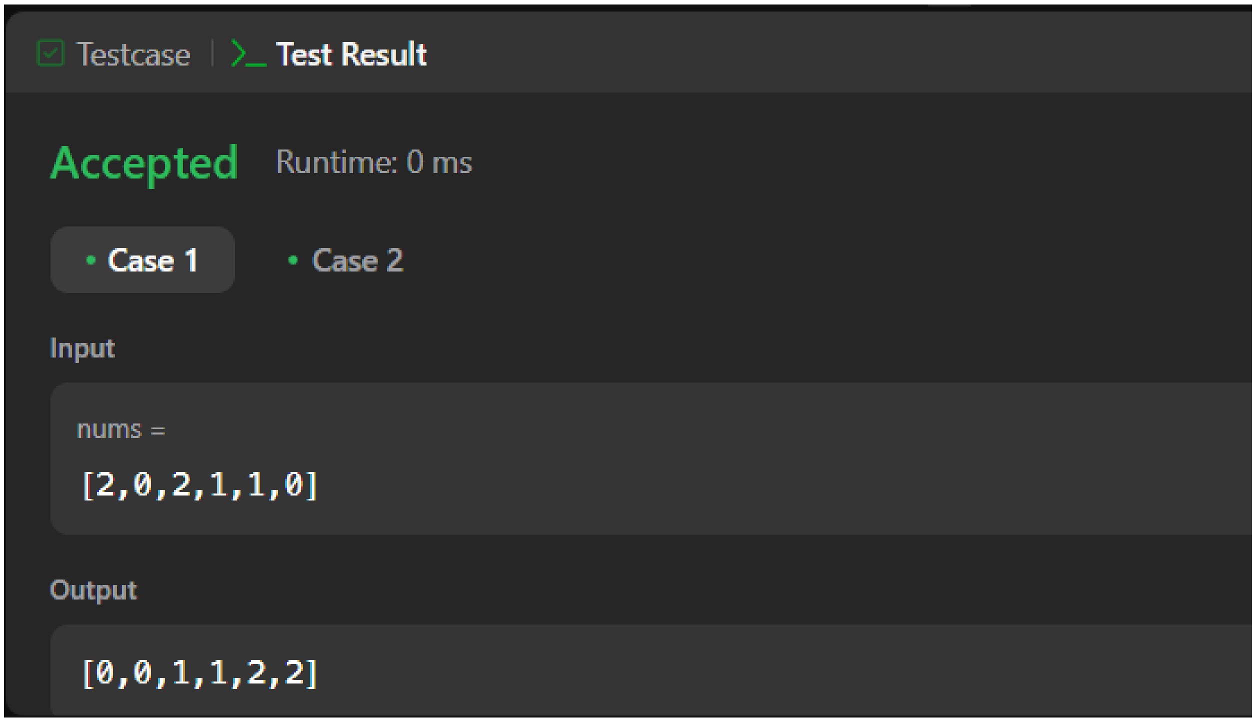
4

3. Sort Colors



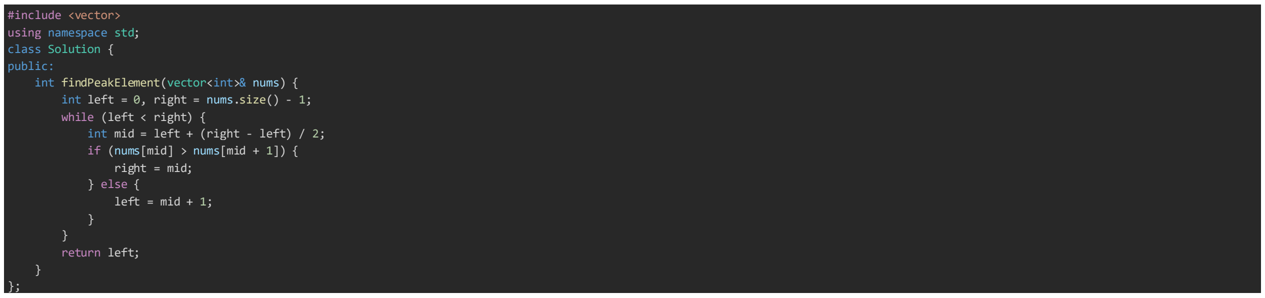
```cpp
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int low = 0, mid = 0, high = nums.size() - 1;
        while (mid <= high) {
            if (nums[mid] == 0) {
                swap(nums[mid], nums[low]);
                low++;
                mid++;
            } else if (nums[mid] == 1) {
                mid++;
            } else {
                swap(nums[mid], nums[high]);
                high--;
            }
        }
    }
};
```

Output



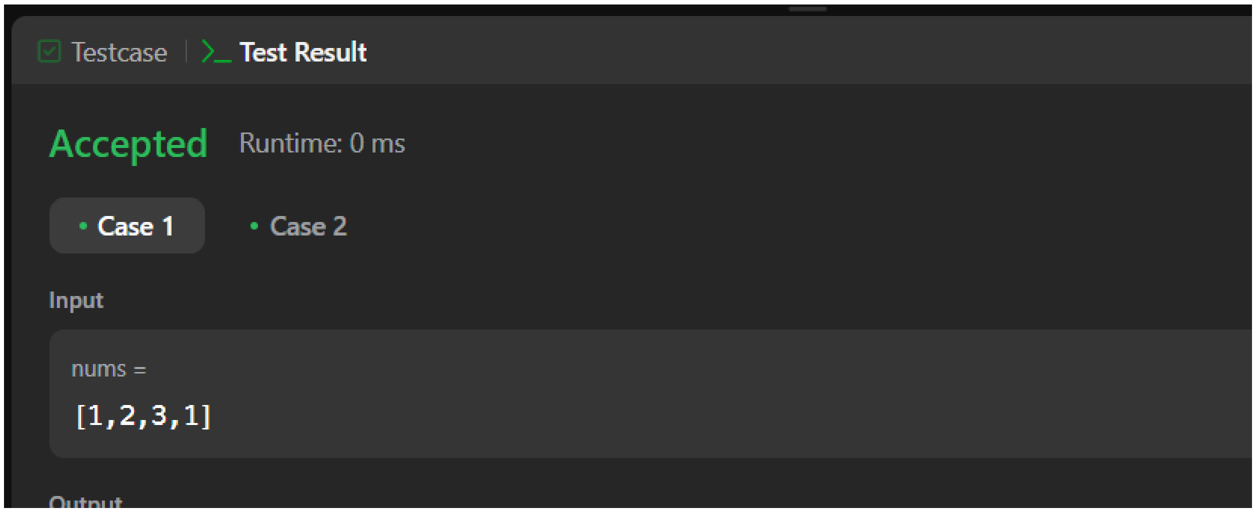
```
Accepted Runtime: 0 ms
Case 1 Case 2
Input
nums =
[2,0,2,1,1,0]
Output
[0,0,1,1,2,2]
```

4.Find Peak Element



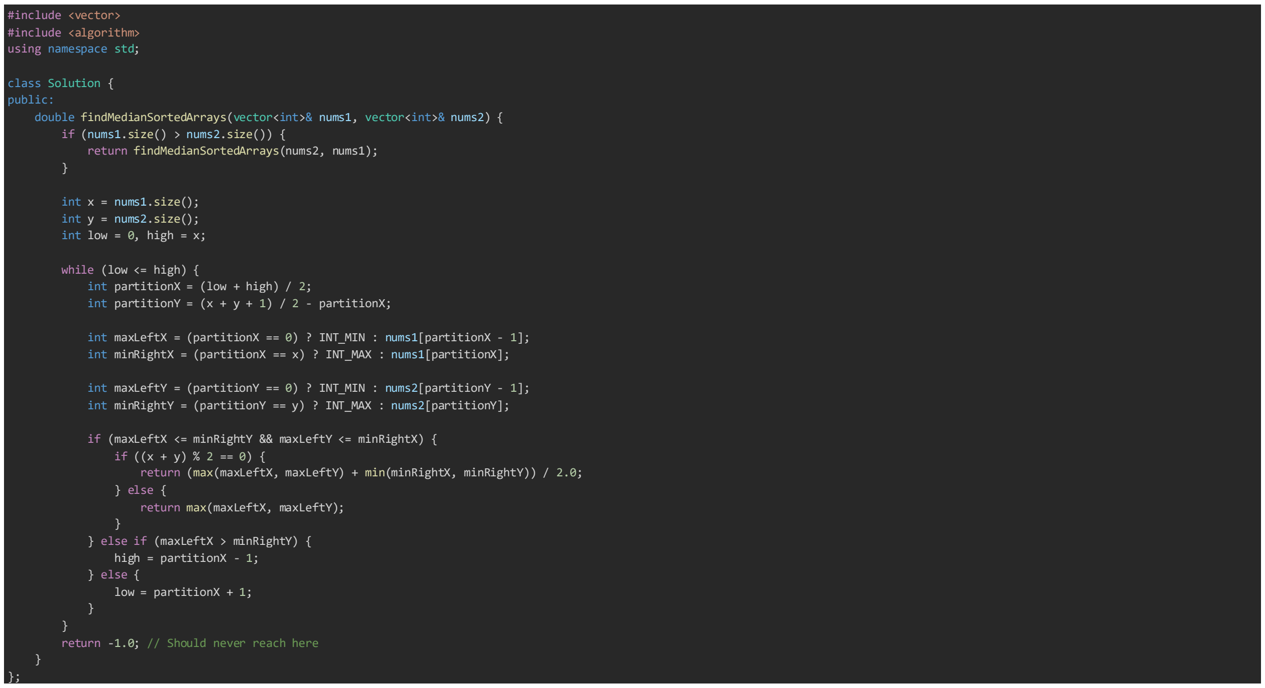
```
#include <vector>
using namespace std;
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int left = 0, right = nums.size() - 1;
        while (left < right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] > nums[mid + 1]) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }
        return left;
    }
};
```

Output:



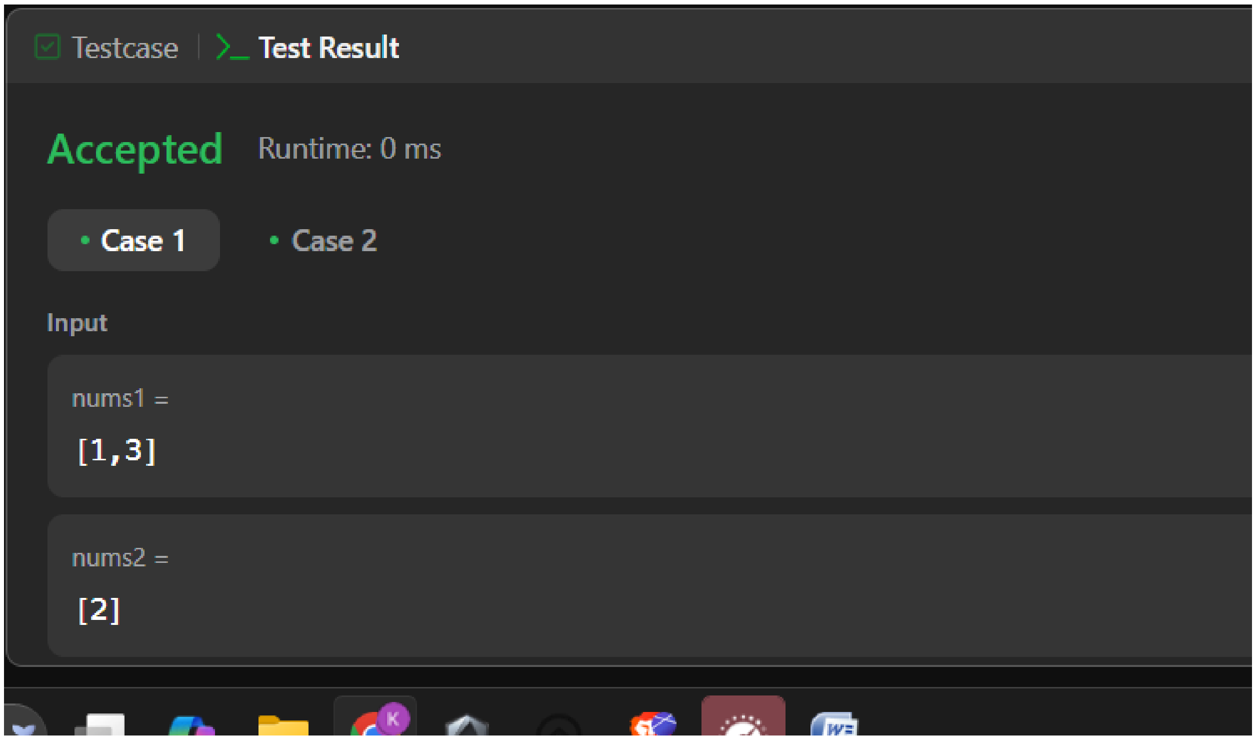
```
Accepted Runtime: 0 ms
Case 1 Case 2
Input
nums =
[1,2,3,1]
Output
```

5. Median of two sorted array



```cpp
#include <vector>
#include <algorithm>
using namespace std;

class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        if (nums1.size() > nums2.size()) {
            return findMedianSortedArrays(nums2, nums1);
        }

        int x = nums1.size();
        int y = nums2.size();
        int low = 0, high = x;

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

            int maxLeftX = (partitionX == 0) ? INT_MIN : nums1[partitionX - 1];
            int minRightX = (partitionX == x) ? INT_MAX : nums1[partitionX];

            int maxLeftY = (partitionY == 0) ? INT_MIN : nums2[partitionY - 1];
            int minRightY = (partitionY == y) ? INT_MAX : nums2[partitionY];

            if (maxLeftX <= minRightY && maxLeftY <= minRightX) {
                if ((x + y) % 2 == 0) {
                    return (max(maxLeftX, maxLeftY) + min(minRightX, minRightY)) / 2.0;
                } else {
                    return max(maxLeftX, maxLeftY);
                }
            } else if (maxLeftX > minRightY) {
                high = partitionX - 1;
            } else {
                low = partitionX + 1;
            }
        }
        return -1.0; // Should never reach here
    }
};
```

Output:



Accepted Runtime: 0 ms

Case 1 Case 2

Input

nums1 =

[1,3]

nums2 =

[2]