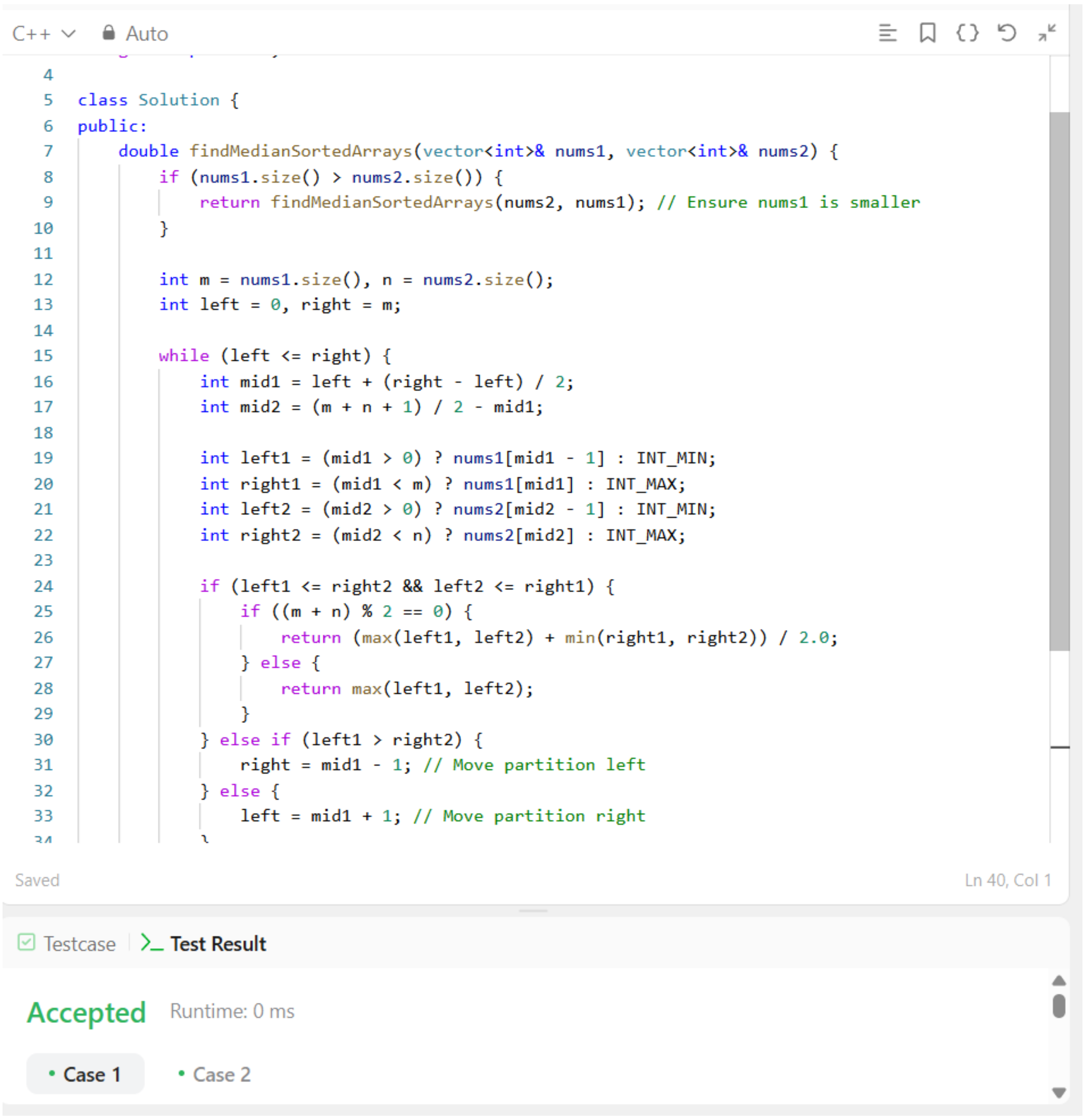
ASSIGNMENT

Student Name: Nikhil yadav **UID: 22BCS11565**

Branch: BE-CSE **Section/Group: 608/B**

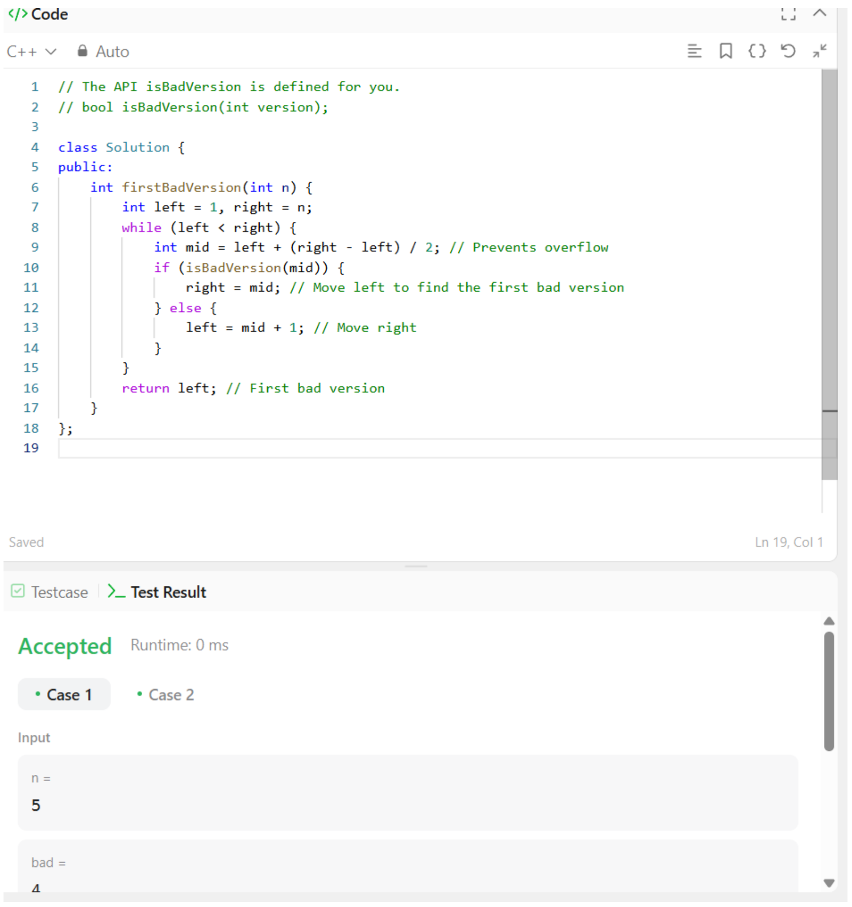
Semester: 6th **Subject Name: AP LAB**

1. Merge Sorted Array:



```cpp

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

        int m = nums1.size(), n = nums2.size();
        int left = 0, right = m;

        while (left <= right) {
            int mid1 = left + (right - left) / 2;
            int mid2 = (m + n + 1) / 2 - mid1;

            int left1 = (mid1 > 0) ? nums1[mid1 - 1] : INT_MIN;
            int right1 = (mid1 < m) ? nums1[mid1] : INT_MAX;
            int left2 = (mid2 > 0) ? nums2[mid2 - 1] : INT_MIN;
            int right2 = (mid2 < n) ? nums2[mid2] : INT_MAX;

            if (left1 <= right2 && left2 <= right1) {
                if ((m + n) % 2 == 0) {
                    return (max(left1, left2) + min(right1, right2)) / 2.0;
                } else {
                    return max(left1, left2);
                }
            } else if (left1 > right2) {
                right = mid1 - 1; // Move partition left
            } else {
                left = mid1 + 1; // Move partition right
```

2. First Bad Version:



```cpp
// The API isBadVersion is defined for you.
// bool isBadVersion(int version);

class Solution {
public:
    int firstBadVersion(int n) {
        int left = 1, right = n;
        while (left < right) {
            int mid = left + (right - left) / 2; // Prevents overflow
            if (isBadVersion(mid)) {
                right = mid; // Move left to find the first bad version
            } else {
                left = mid + 1; // Move right
            }
        }
        return left; // First bad version
    }
};

```

Saved

Ln 19, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

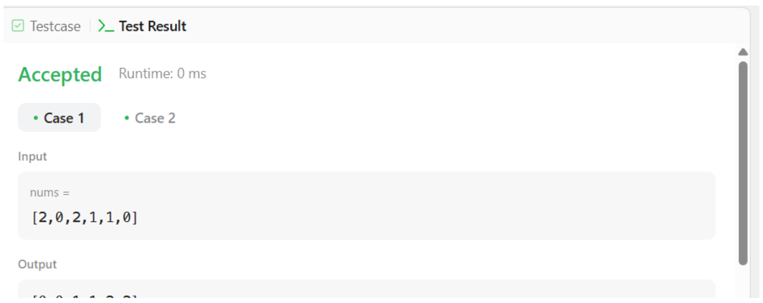
n =

5

bad =

4

3. Sort Colors:



Testcase | Test Result

Accepted Runtime: 0 ms

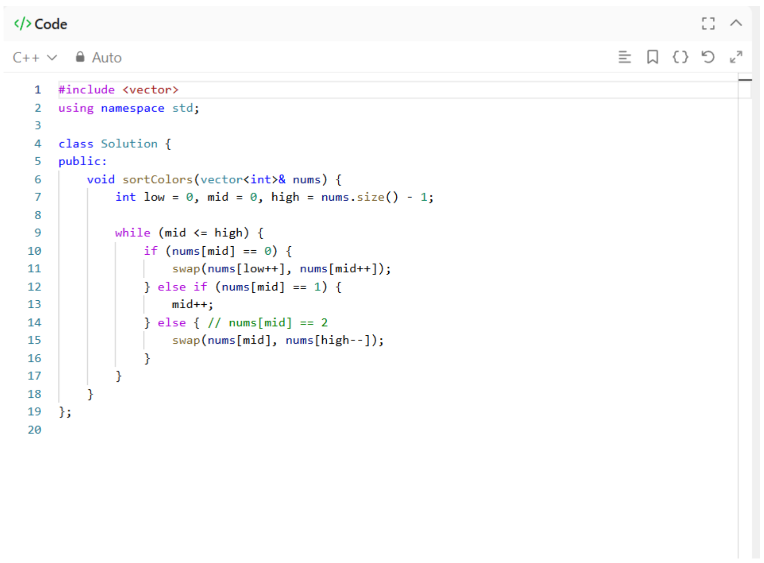
• Case 1 • Case 2

Input

nums =

[2,0,2,1,1,0]

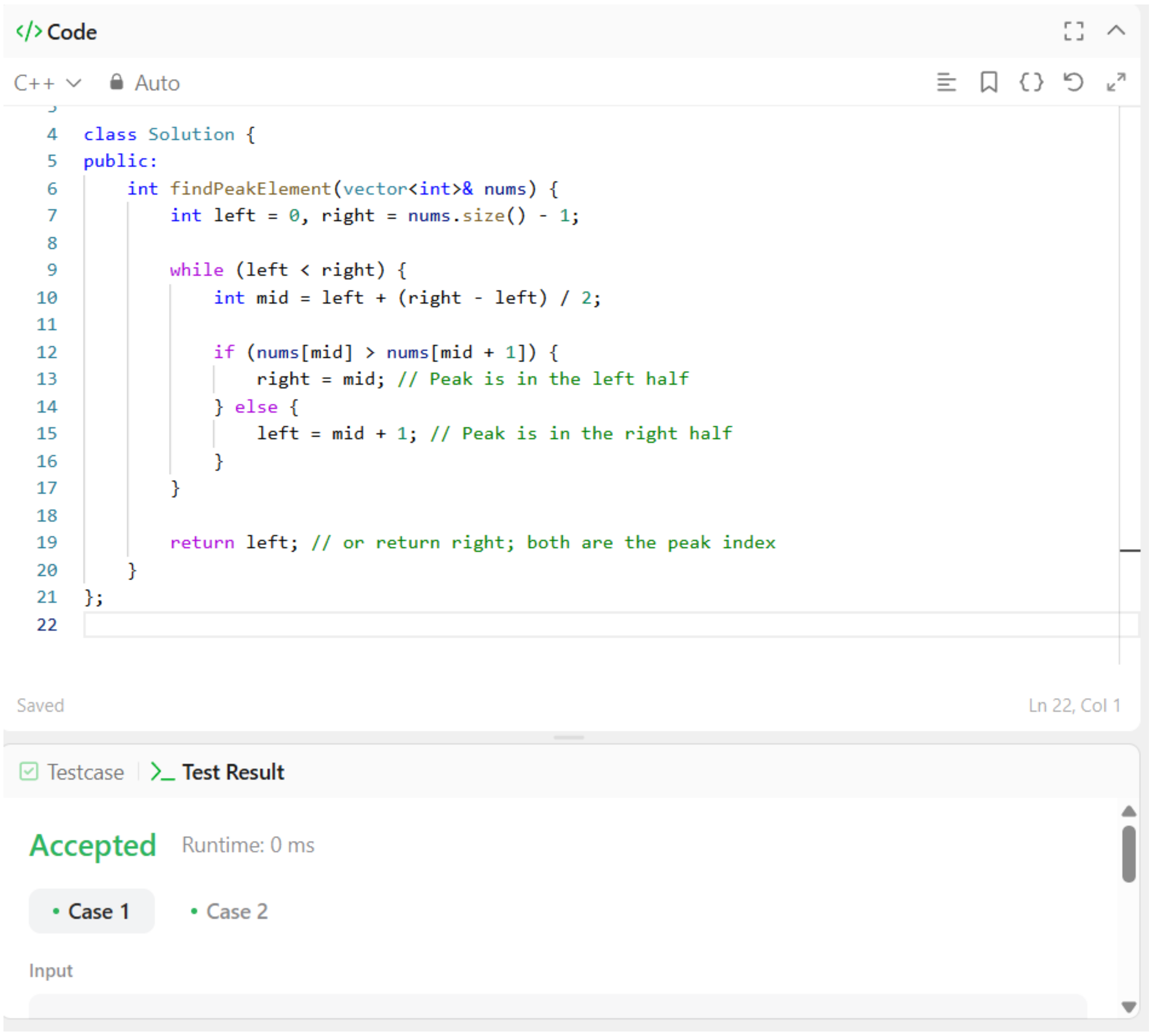
Output


</> Code

C++ Auto

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

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

```

4. Find Peak Element:


</> Code

C++ Auto

```cpp
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; // Peak is in the left half
            } else {
                left = mid + 1; // Peak is in the right half
            }
        }

        return left; // or return right; both are the peak index
    }
};

```

Saved Ln 22, Col 1

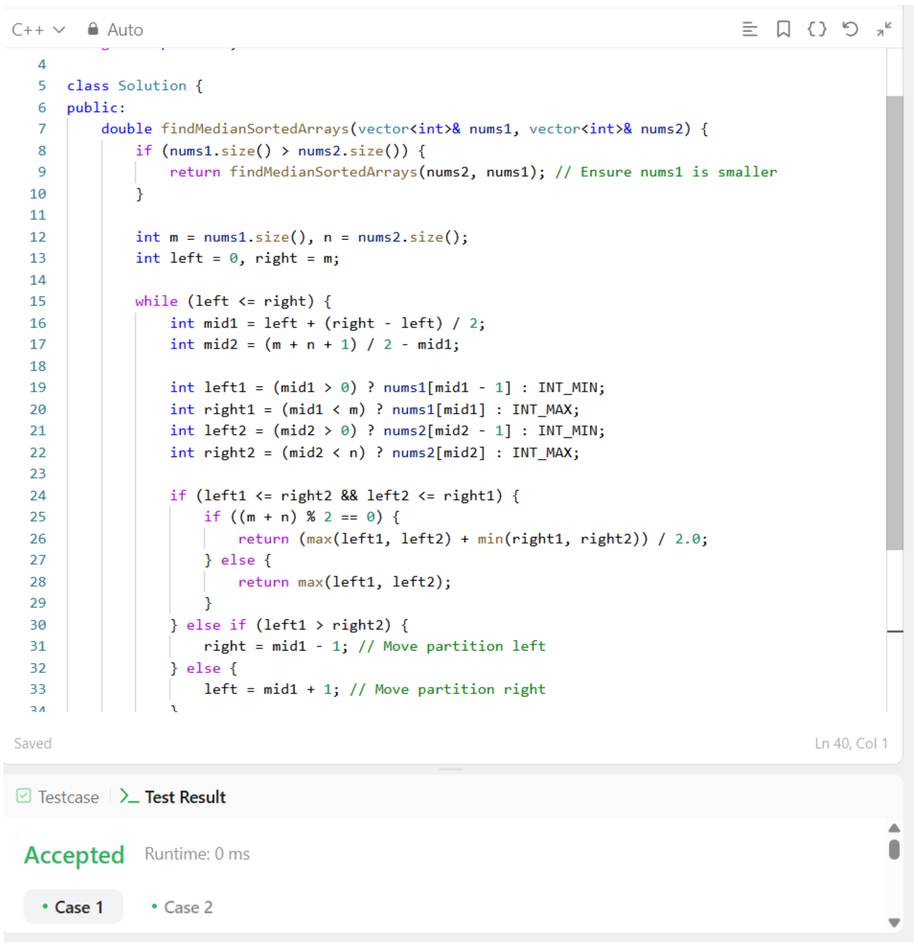
Testcase >_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

5. Median of Two Sorted Arrays:



```cpp

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

        int m = nums1.size(), n = nums2.size();
        int left = 0, right = m;

        while (left <= right) {
            int mid1 = left + (right - left) / 2;
            int mid2 = (m + n + 1) / 2 - mid1;

            int left1 = (mid1 > 0) ? nums1[mid1 - 1] : INT_MIN;
            int right1 = (mid1 < m) ? nums1[mid1] : INT_MAX;
            int left2 = (mid2 > 0) ? nums2[mid2 - 1] : INT_MIN;
            int right2 = (mid2 < n) ? nums2[mid2] : INT_MAX;

            if (left1 <= right2 && left2 <= right1) {
                if ((m + n) % 2 == 0) {
                    return (max(left1, left2) + min(right1, right2)) / 2.0;
                } else {
                    return max(left1, left2);
                }
            } else if (left1 > right2) {
                right = mid1 - 1; // Move partition left
            } else {
                left = mid1 + 1; // Move partition right
```

Accepted Runtime: 0 ms

• Case 1 • Case 2