

## ASSIGNMENT - 5

Name: Chahat Sharma

UID: 22BCS15005

Section: 608- B

Subject: AP

Solution 1: class Solution {

public:

void merge(vector<int>C nums1, int m, vector<int>C nums2, int n) {

int i = m - 1, j = n - 1, k = m + n - 1;

while (i >= 0 CC j >= 0) {

if (nums1[i] > nums2[j]) {

nums1[k--] = nums1[i--];

} else {

nums1[k--] = nums2[j--];

}

}

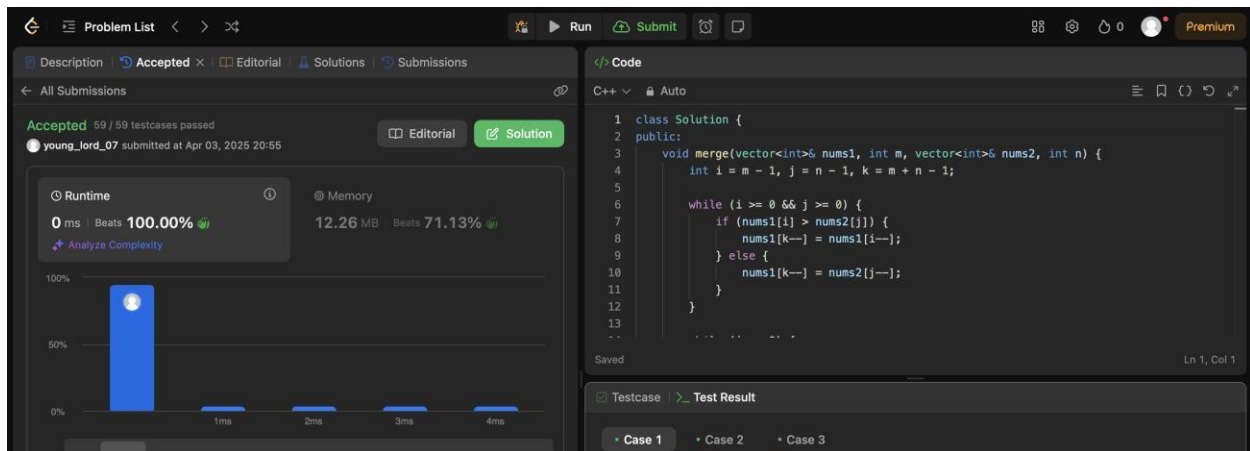
while (j >= 0) {

nums1[k--] = nums2[j--];

}

}

};



Solution 2: 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; // Move left to find the first bad version

} else {

left = mid + 1; // Move right

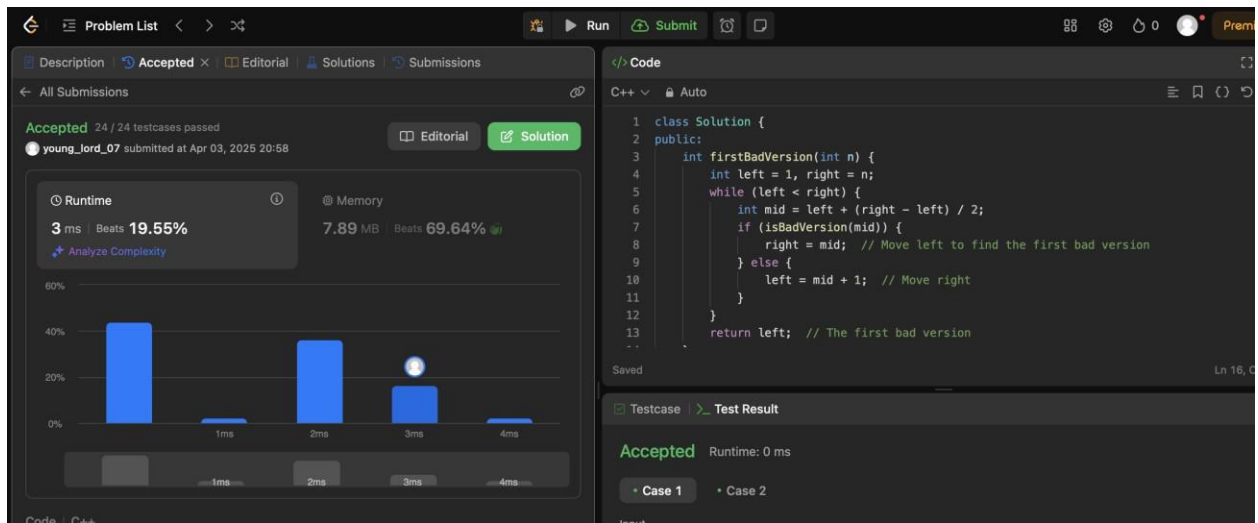
}

}

return left; // The first bad version

}

};



Solution 3: class Solution {

public:

void sortColors(vector<int>C 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--]);

}

}

}

};

Description
Accepted
Editorial
Solutions
Submissions

Accepted 89 / 89 testcases passed

young\_lord\_07 submitted at Apr 03, 2025 20:59

Runtime
0 ms | Beats 100.00%

Memory
11.64 MB | Beats 43.91%

Analyze Complexity

150%
100%
50%
0%

1ms
2ms
3ms
4ms

Code
C++

```

1 class Solution {
2 public:
3     void sortColors(vector<int>& nums) {
4         int low = 0, mid = 0, high = nums.size() - 1;
5
6         while (mid <= high) {
7             if (nums[mid] == 0) {
8                 swap(nums[low++], nums[mid++]);
9             } else if (nums[mid] == 1) {
10                 mid++;
11             } else { // nums[mid] == 2
12                 swap(nums[mid], nums[high--]);
13             }
14         }
15     }
16 }

```

Saved
Ln 17, Col 1

Testcase
Test Result

Accepted Runtime: 0 ms

Case 1
Case 2

Solution 4: 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 left since peak is in the left half

} else {

left = mid + 1; // Move right since peak is in the right half

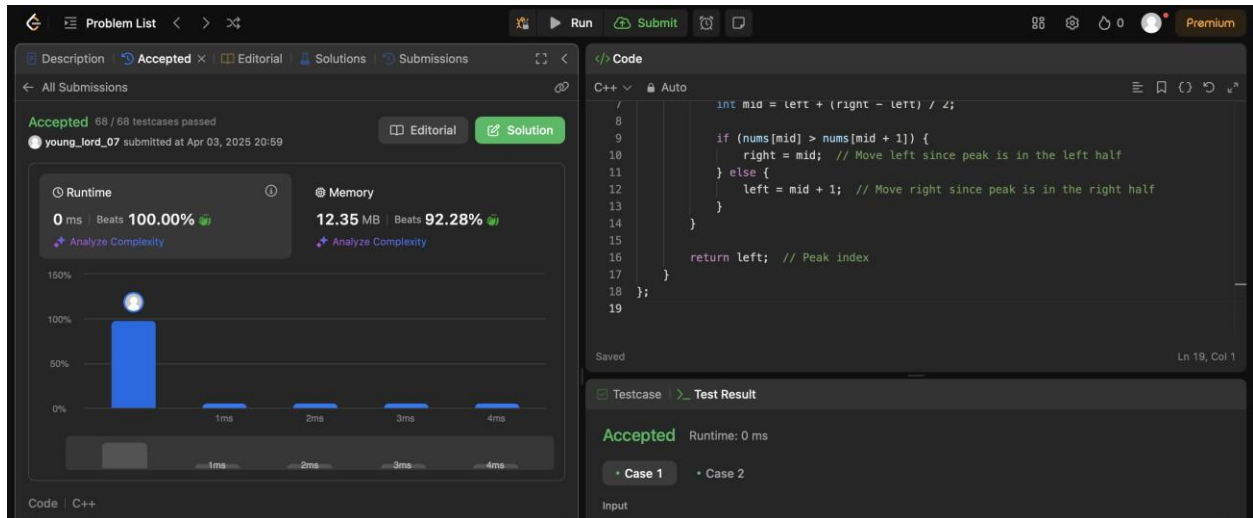
}

}

return left; // Peak index

}

```
};
```



Solution 5:

```
class Solution {
```

```
public:
```

```
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
```

```
        if (nums1.size() > nums2.size())
```

```
            return findMedianSortedArrays(nums2, nums1); // Ensuring nums1 is the smaller
        array
```

```
        int m = nums1.size(), n = nums2.size();
```

```
        int left = 0, right = m, halfLen = (m + n + 1) / 2;
```

```
        while (left <= right) {
```

```
            int mid1 = left + (right - left) / 2;
```

```
            int mid2 = halfLen - 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;
} else {
    left = mid1 + 1;
}
}

return 0.0; // This should never be reached
}
};

```

