ASSIGNMENT 5

Name – Suman Kumar

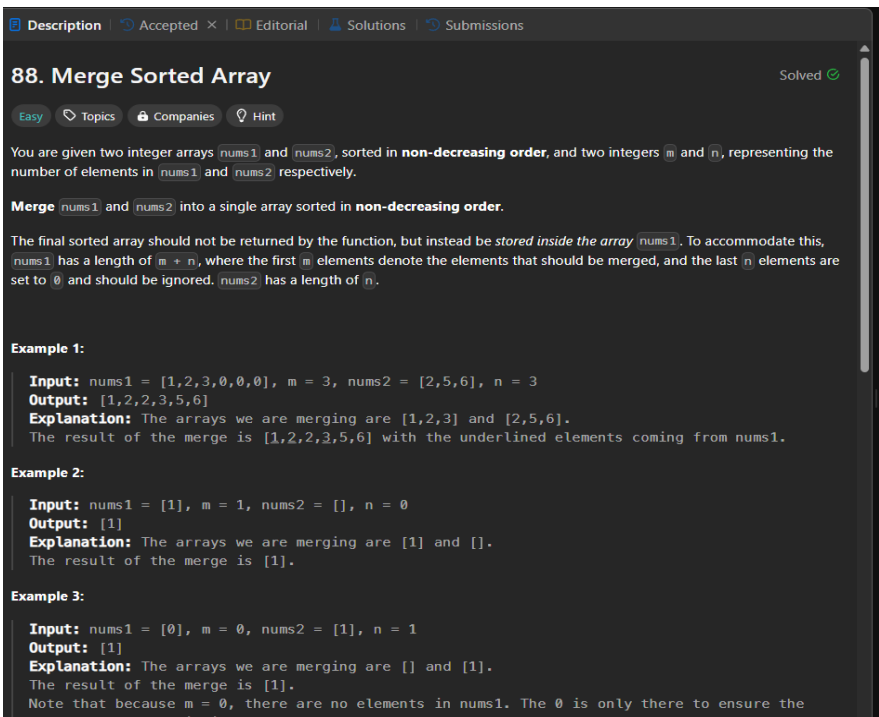
UID – 22BCS15488

Section – IOT-608/B

---

1. Merge Sorted Array

```
class Solution {
public:
    void merge(vector<int>& nums1, int m,
vector<int>& nums2, int n) {
        for (int j = 0, i = m; j<n; j++){
            nums1[i] = nums2[j];
            i++;
        }
        sort(nums1.begin(),nums1.end());
    }
};
```



Description | Accepted | Editorial | Solutions | Submissions

88. Merge Sorted Array — Solved

Easy | Topics | Companies | Hint

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`.

Example 1:

Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]
Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

Example 2:

Input: nums1 = [1], m = 1, nums2 = [], n = 0
Output: [1]
Explanation: The arrays we are merging are [1] and [].
The result of the merge is [1].

Example 3:

Input: nums1 = [0], m = 0, nums2 = [1], n = 1
Output: [1]
Explanation: The arrays we are merging are [] and [1].
The result of the merge is [1].
Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the

Code

C++ | Auto

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        for (int j = 0, i = m; j<n; j++){
            nums1[i] = nums2[j];
            i++;
        }
        sort(nums1.begin(),nums1.end());
    }
};
```

Saved

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

nums1 =
[1,2,3,0,0,0]

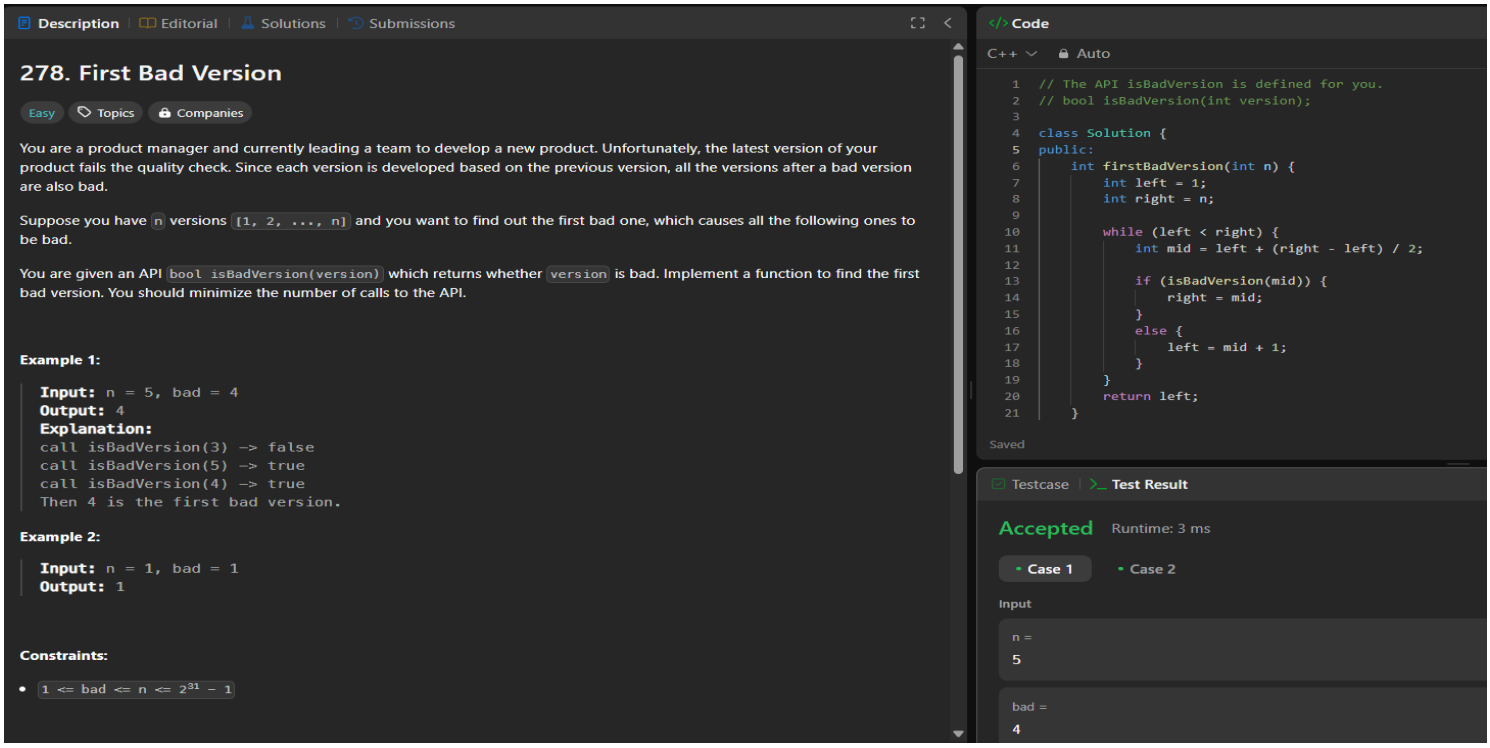
m =
3

nums2 =

```cpp
class Solution {
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;
    }
};
```



Description | Editorial | Solutions | Submissions

278. First Bad Version

Easy | Topics | Companies

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.

Suppose you have `n` versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which returns whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example 1:

```
Input: n = 5, bad = 4
Output: 4
Explanation:
call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true
Then 4 is the first bad version.
```

Example 2:

```
Input: n = 1, bad = 1
Output: 1
```

Constraints:

- `1 <= bad <= n <= 2^31 - 1`

Code

C++ | Auto

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

class Solution {
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;
    }
```

Saved

Testcase | Test Result

Accepted Runtime: 3 ms

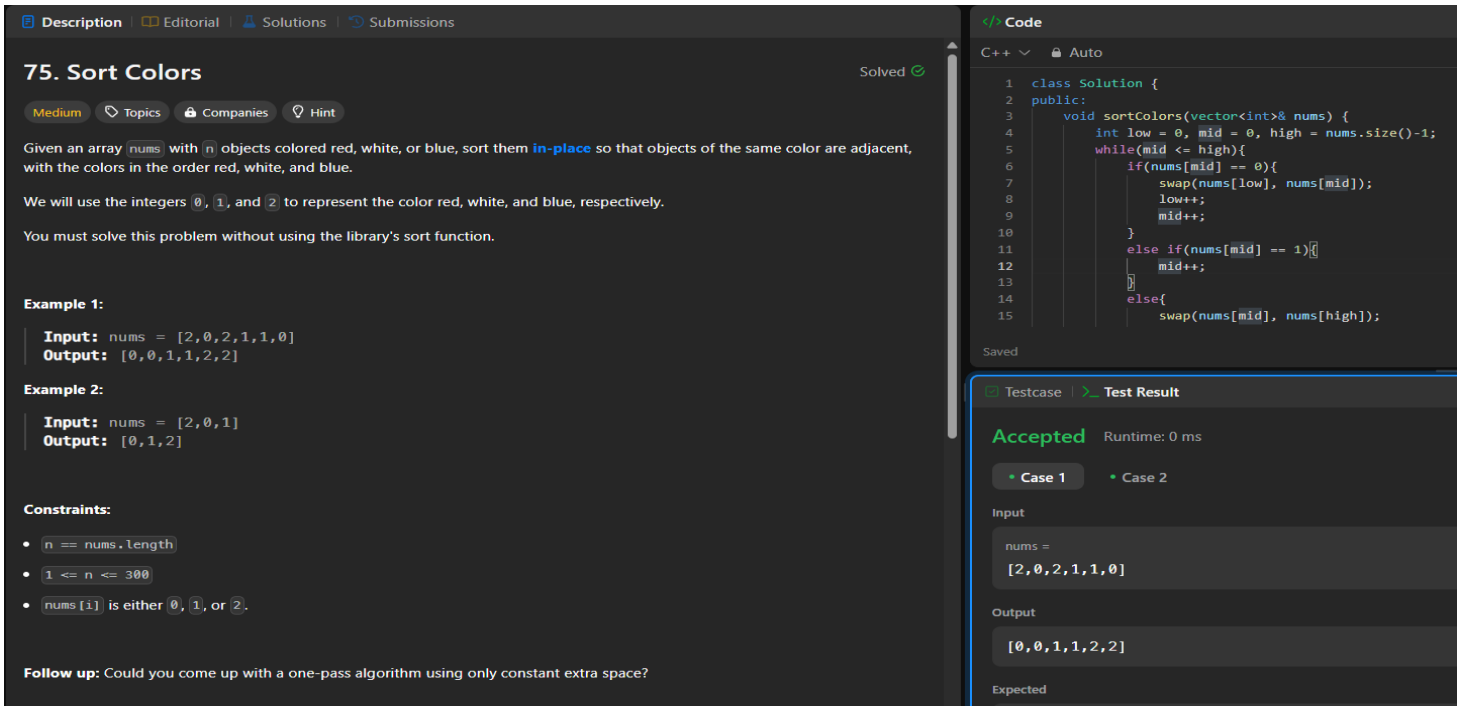
Case 1 | Case 2

Input

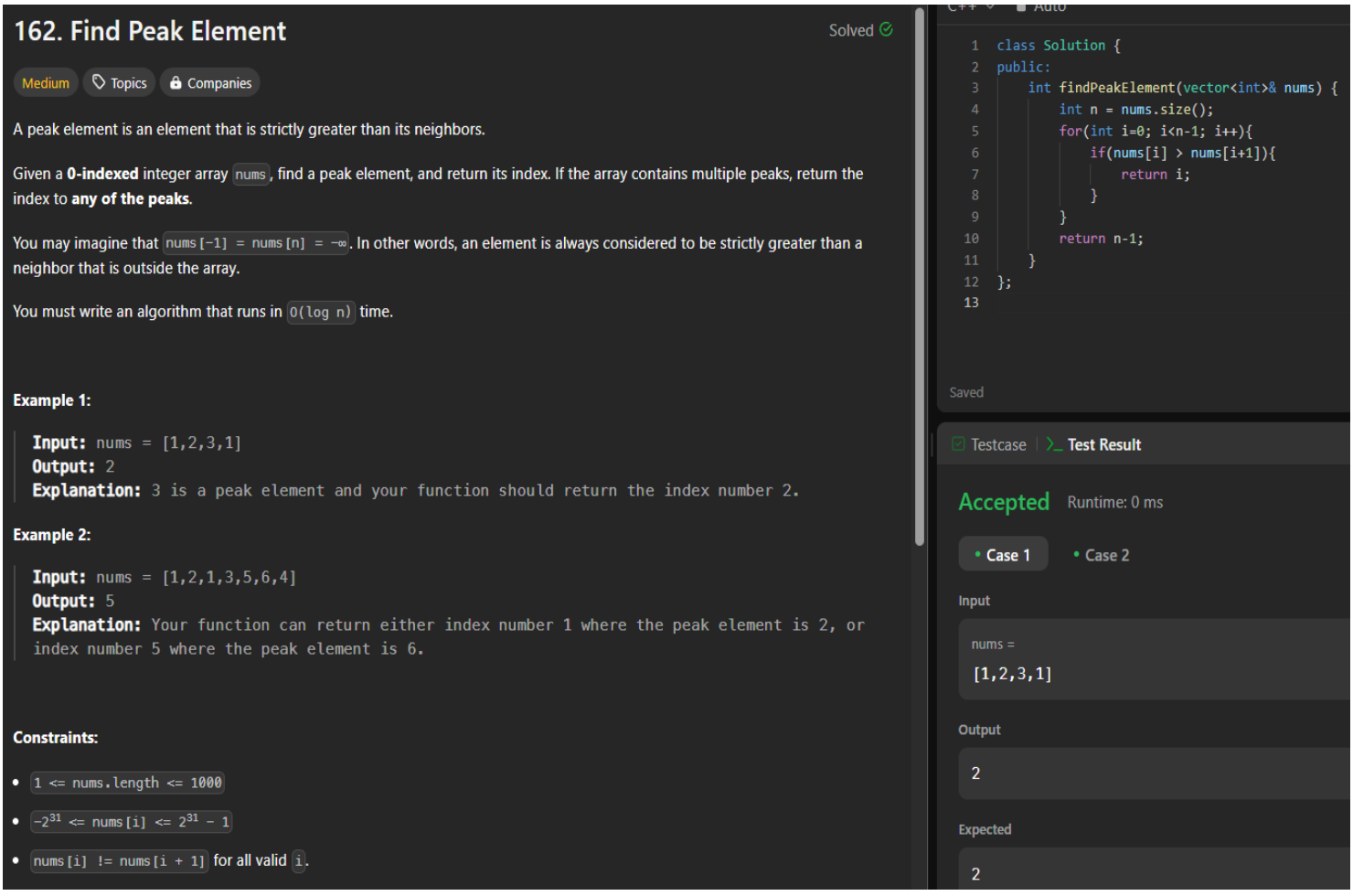
n =
5

bad =
4

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



```cpp
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int n = nums.size();
        for(int i=0; i<n-1; i++){
            if(nums[i] > nums[i+1]){
                return i;
            }
        }
        return n-1;
    }
};
```



162. Find Peak Element

Solved

Medium | Topics | Companies

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.

Example 1:

> **Input:** nums = [1,2,3,1]
> **Output:** 2
> **Explanation:** 3 is a peak element and your function should return the index number 2.

Example 2:

> **Input:** nums = [1,2,1,3,5,6,4]
> **Output:** 5
> **Explanation:** Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

Constraints:

- `1 <= nums.length <= 1000`
- `-2^31 <= nums[i] <= 2^31 - 1`
- `nums[i] != nums[i + 1]` for all valid `i`.

C++ Auto

```cpp
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int n = nums.size();
        for(int i=0; i<n-1; i++){
            if(nums[i] > nums[i+1]){
                return i;
            }
        }
        return n-1;
    }
};
```

Saved

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =
[1,2,3,1]

Output

2

Expected

2

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>&
nums1, vector<int>& nums2) {
        vector<int>v;

        for(auto num:nums1)
            v.push_back(num);

        for(auto num:nums2)
            v.push_back(num);

        sort(v.begin(),v.end());
        int n=v.size();

        return n%2?v[n/2]:(v[n/2-1]+v[n/2])/2.0;
    }
};
```