Experiment 4

Student Name: Ankit Vashisth **UID: 22BCS13378**

Branch: UIE CSE 3rd Year **Section/Group: 22BCS_KPIT-901-'B'**

Semester: 6th **Date of Performance: 18th Feb 2025**

Subject Name: Advanced Programming – II **Subject Code: 22CSP-351**

1. Aim:

You are given two integer arrays nums1 and nums2, sorted **non-decreasing order**, and two integers m and n, representing the number of elements in nun and nums2 respecti

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

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

2. Implementation/Code:

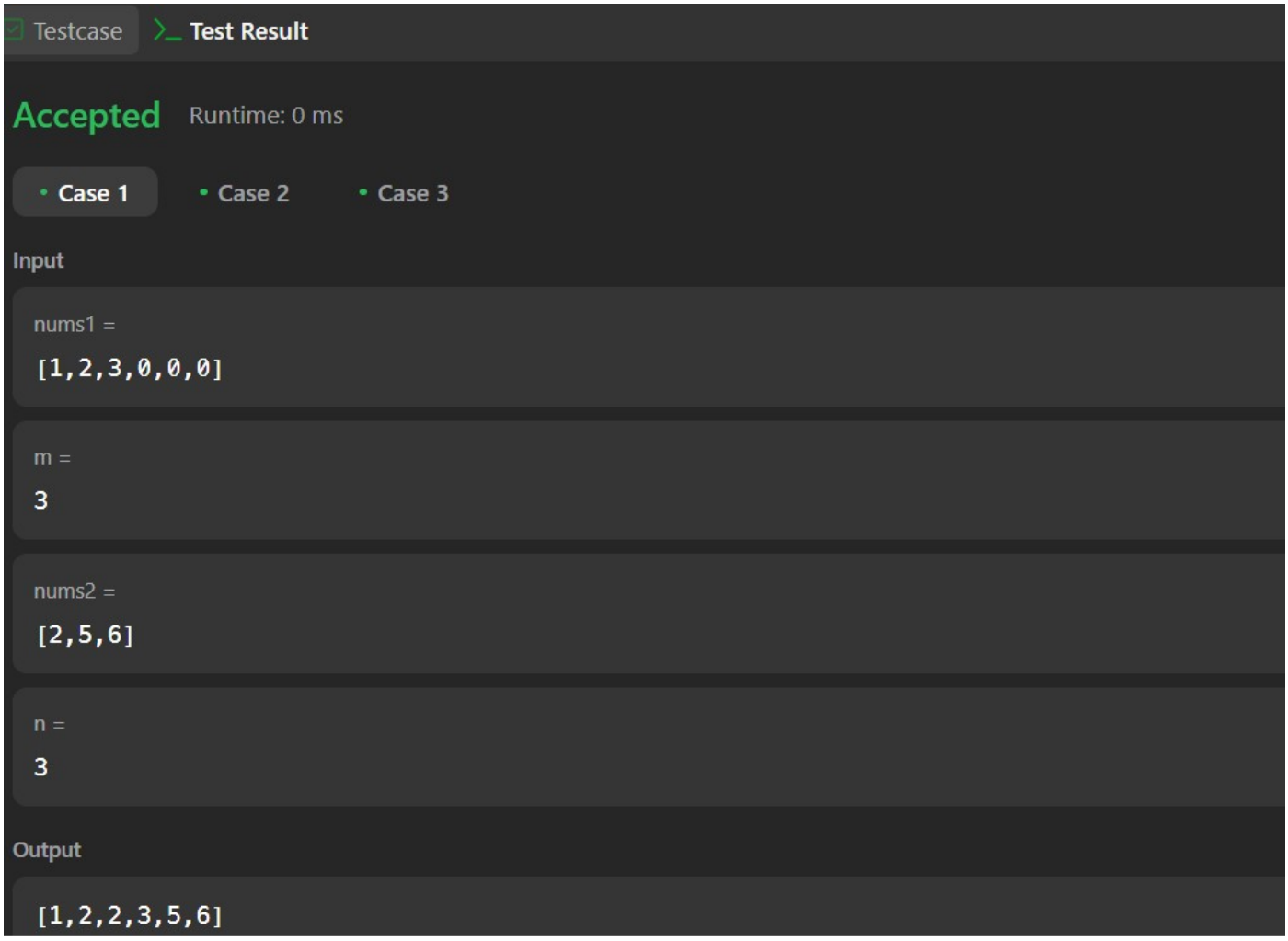
```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n)
{         vector<int> v;         int i = 0, j = 0;         while (i <
m && j < n) {             if (nums1[i] >= nums2[j])
                v.push_back(nums2[j++]);
else

v.push_back(nums1[i++]);         }
while (i < m)
            v.push_back(nums1[i++]);
while (j < n)
            v.push_back(nums2[j++]);


        nums1 = v;
```

```
    }
};
```

3. **Output:**



QUES:2

1. **Aim:**

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.

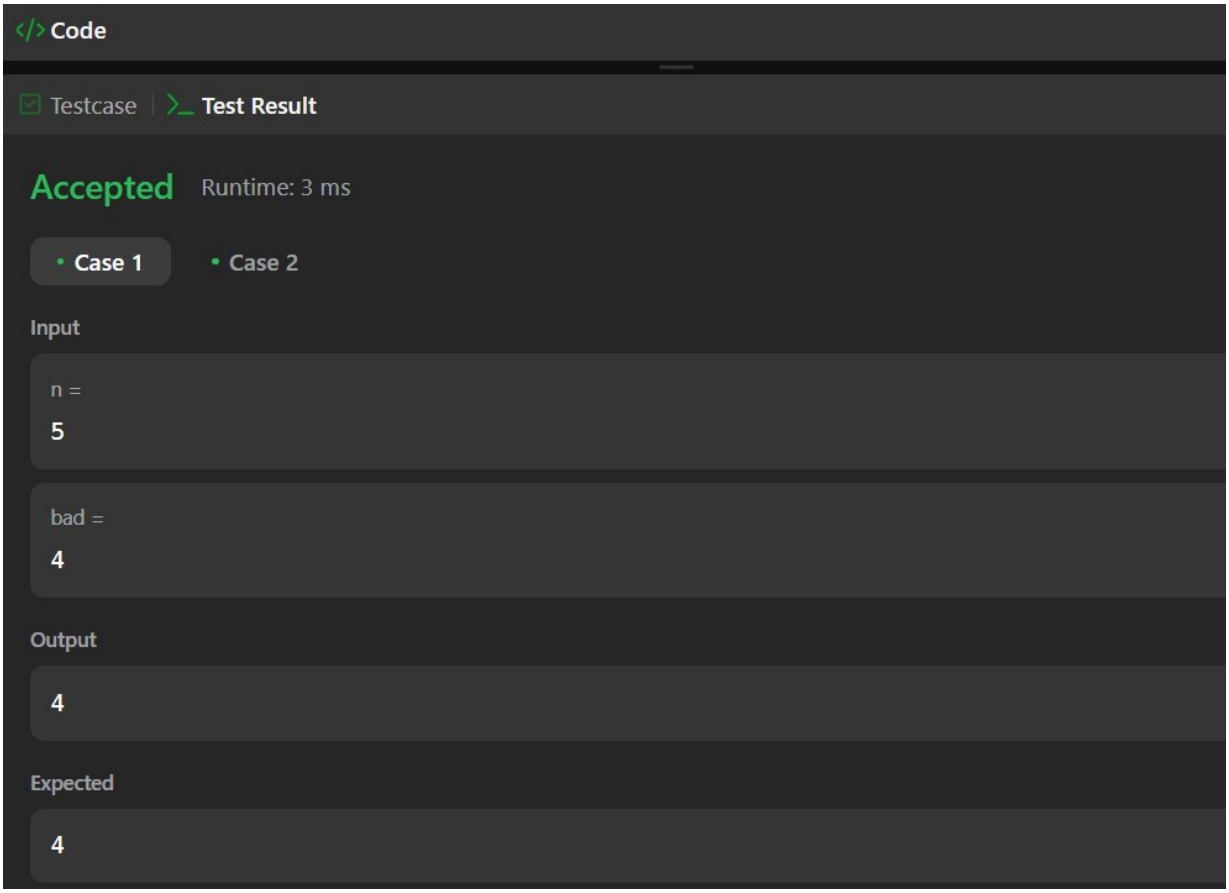
You are given an API bool isBadVersion(version) which returns whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

2. **Implementation/Code:**

```
class Solution {
public:
    int firstBadVersion(int n) {
int first = 1;          int last
= n;
         while (first < last) {             int
mid = first + (last - first) / 2;

            if    (isBadVersion(mid))    {
last = mid;
            } else {
first = mid + 1;
            }
        }
         return
first;
    }
};
```

3. **Output:**



QUESTION:3 Sort Colors

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
int n=nums.size();          for(int i=0;i<n-
1;i++){             for(int j=0;j<n-i-
1;j++){                 if(nums[j] >
nums[j+1]){
swap(nums[j],nums[j+1]);
                }
            }
        }

    }
};
```

QUESTION:4 Top K Frequent Elements

```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k)
{         vector<int> arr;
unordered_map<int,int>freq;         for(int f:
nums)freq[f]++;
priority_queue<pair<int,int>>maxHeap;         for(auto
&i:freq){           maxHeap.push({i.second,i.first});
        }
        while(k-- && !maxHeap.empty()){
arr.push_back(maxHeap.top().second);             maxHeap.pop();
        }
        return arr;
    }
};
```

QUESTION:5 Kth Largest Element in an Array

```
class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
sort(nums.begin(),nums.end());              return
nums[nums.size()-k];
    }
};
```

QUESTION:6 Merge Intervals

```
class   Solution   {   public:              vector<vector<int>>
merge(vector<vector<int>>& intervals) {
               if(intervals.size()==1)
return intervals;
vector<pair<int,int>> p;       for(int
i=0;i<intervals.size();i++)       {



          p.push_back({intervals[i][0],intervals[i][1]});
}
      sort(p.begin(),p.end());

      vector<vector<int>>  ans;          int
f=p[0].first,s=p[0].second;         for(int
i=0;i<p.size()-1;i++)
      {
          vector<int> a(2);
if(s>=p[i+1].first)
          {
s=max(s,p[i+1].second);
          }           else
{               a[0]=f;
a[1]=s;                f=p[i+1].first;
s=p[i+1].second;
ans.push_back(a);
          }       }
      int n=intervals.size();
ans.push_back({f,s});       return ans;
    }
};
```

QUESTION:7 Search in Rotated Sorted Array

```
class Solution { public:
    int search(std::vector<int>& nums, int target) {          int low = 0, high =
nums.size() - 1;
         while (low <= high) {
            int mid = (low + high) / 2;

            if (nums[mid] == target) {



                return mid;
            }
             if (nums[low] <= nums[mid]) {                  if
(nums[low] <= target && target < nums[mid]) {
high = mid - 1;
                } else {
                    low = mid + 1;
                }
} else {
                if (nums[mid] < target && target <= nums[high]) {
low = mid + 1;
                } else {
high = mid - 1;
                }
            }
        }

        return -1;
    }
};
```

QUESTION:8 Search a 2D Matrix II

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target)
{         int row = matrix.size()-1;         int col = matrix[
0 ].size()-1;         int i = 0, j = col;         while( i <=
row && j >= 0 ){             if( matrix[ i ][ j ] == target )
return true;             else if( matrix[ i ][ j ] > target )
j--;             else i++;
        }
return false;
    }
};
```

QUESTION:9 Median of Two Sorted Arrays

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
int s1=nums1.size(),s2=nums2.size();         int i=0,j=0;
vector<int>v;         while(i<s1 && j<s2){
if(nums1[i]<nums2[j])v.push_back(nums1[i++]);
            v.push_back(nums2[j++]);
}

while(i<s1)v.push_back(nums1[i++]);
while(j<s2)v.push_back(nums2[j++]);
double median;         int size =
v.size();         if(size%2==0){
int mid1=size/2;             int
mid2=(size/2)-1;
median=(v[mid1]+v[mid2])/2.0;
return median;
        }
        return v[size/2];

    }
};
```

QUESTION 10: Kth Smallest Element in a Sorted Matrix

```
class Solution {
public:
    int kthSmallest(vector<vector<int>>& matrix, int k) {
priority_queue<int> maxHeap;
         for (const auto& row : matrix)
{             for (int num : row) {
maxHeap.push(num);                 if
(maxHeap.size() > k) {
maxHeap.pop();
```

```
                }
            }
        }
        return maxHeap.top();
    }
};
```