Experiment 1.2

Student Name: Nikhil Sharma **UID:**22BCS15209
Branch:CSE **Section/Group:**640/B
Semester: 6 **Date of Performance:** 16/01/25
Subject Name:Advance Programming -2 **Subject Code:** 22CSH-351

Aim 1: : 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

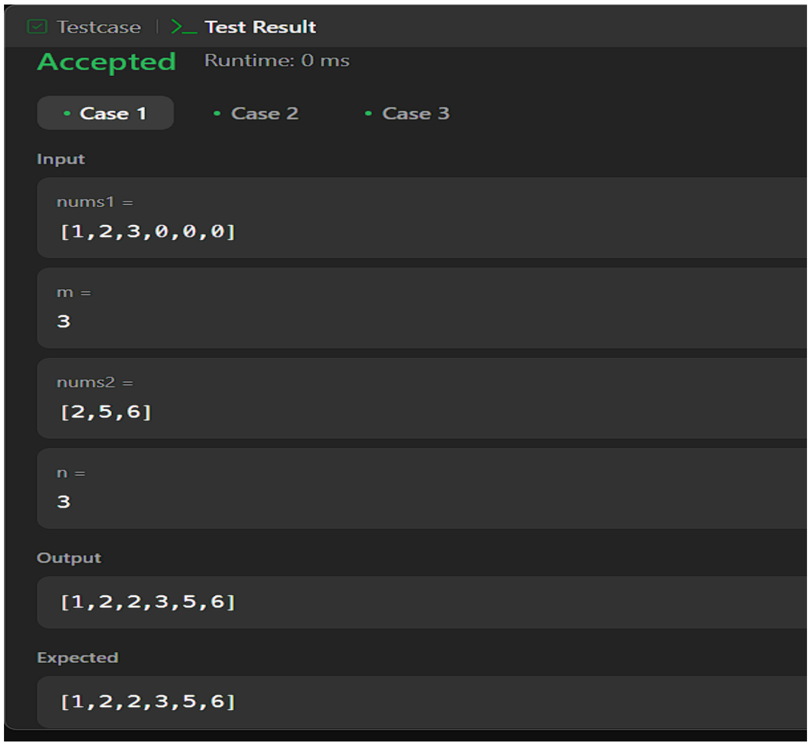
Objective : Return indices of the two numbers such that they add up to target.

Code:

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

    }
};
```

OUTPUT:



```
Testcase | >_ Test Result
Accepted    Runtime: 0 ms

• Case 1    • Case 2    • Case 3

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

m =
3

nums2 =
[2,5,6]

n =
3

Output
[1,2,2,3,5,6]

Expected
[1,2,2,3,5,6]
```

Aim 2: 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.

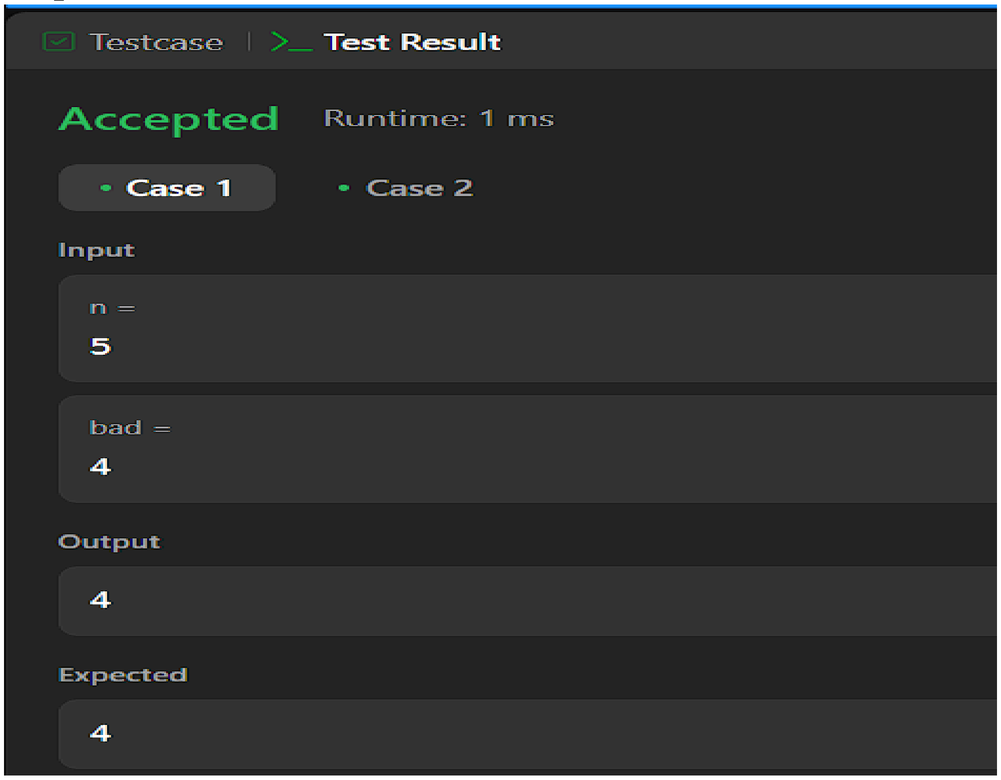
Objective: You are given an API bool isBadVersion(version) which returns whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Code:

```
public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        // The API isBadVersion is defined for you.
        int l=1;
        int r =n;
        while(l< r){
            int mid=l+(r-l)/2;
            if(isBadVersion(mid)){
                r=mid;
            }
            else{
                l=mid+1;
            }
        }
        return l;

    }
}
```

Output:



```
Testcase | >_ Test Result

Accepted   Runtime: 1 ms

• Case 1   • Case 2

Input
n =
5
bad =
4

Output
4

Expected
4
```

Learning Outcomes:

1. Understanding Algorithms – Learn the principles and mechanics behind various searching and sorting algorithms.
2. Time and Space Complexity – Analyze and compare the efficiency of different algorithms.
3. Implementation Skills – Gain hands-on experience in coding and optimizing search and sort functions.
4. Real-World Applications – Apply searching and sorting techniques to practical problems in computing.