

Experiment 4:

Student Name: Manish lalwani

Branch: BE-CSE

Semester: 6

Subject Name: Advanced Programming Lab-2

UID: 22BCS16288

Section/Group: KPIT_901/B

Date of Performance: 19/02/25

Subject Code: 22CSP-351

1. Aim(a):

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`.

2. Objective: The objective of this program is to merge two sorted integer arrays, `nums1` and `nums2`, into a single sorted array in non-decreasing order. The merged result should be stored in `nums1` without using extra space, utilizing its allocated size of `m + n`.

3. Algorithm:

- Initialize three pointers: `i = m - 1`, `j = n - 1`, and `k = m + n - 1`.
- Iterate while `j >= 0`
 - If `i >= 0` and `nums1[i] > nums2[j]`, place `nums1[i]` at `nums1[k]`, decrement `i`.
 - Else, place `nums2[j]` at `nums1[k]`, decrement `j`.
 - Decrement `k` after each placement.

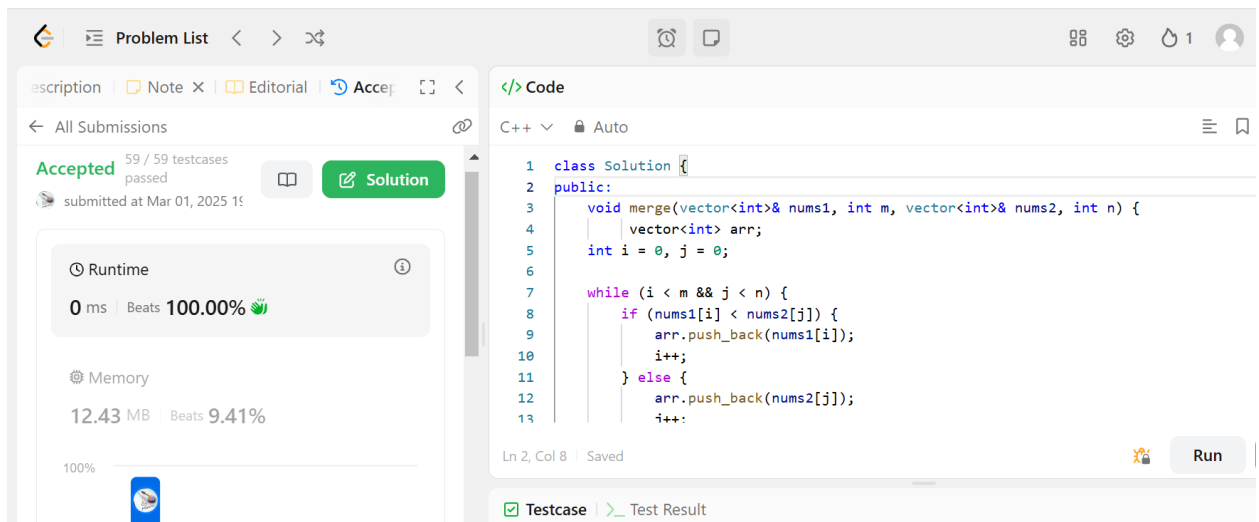
4. Code:

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i=m-1;
        int j=n-1;
        int k=m+n-1;
        while(j>=0)
            if(i>=0 && nums1[i]>nums2[j])
                nums1[k--]=nums1[i--];
            else
                nums1[k--]=nums2[j--];
    }
};
```

Leetcode link:

<https://leetcode.com/problems/merge-sorted-array/submissions/1559217600/>

5. Output:



The screenshot displays the LeetCode submission page for the 'Merge Sorted Array' problem. The solution is in C++ and has been accepted. The left sidebar shows the problem description, a list of submissions, and performance metrics: Runtime 0 ms (Beats 100.00%) and Memory 12.43 MB (Beats 9.41%). The main area shows the C++ code for the 'merge' function, which uses a two-pointer approach to merge two sorted arrays into a third array. The code is as follows:

```
1 class Solution {
2 public:
3     void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
4         vector<int> arr;
5         int i = 0, j = 0;
6
7         while (i < m && j < n) {
8             if (nums1[i] < nums2[j]) {
9                 arr.push_back(nums1[i]);
10                i++;
11            } else {
12                arr.push_back(nums2[j]);
13                j++;
14            }
15        }
16        while (i < m) arr.push_back(nums1[i]);
17        while (j < n) arr.push_back(nums2[j]);
18        nums1 = arr;
19    }
20 }
```

6. Time Complexity:

The time complexity is $O(m + n)$, where m and n are the lengths of the given arrays.

7. Learning Outcomes:

- Learnt how to merge two sorted arrays efficiently using a two-pointer approach.
- Learnt how to traverse arrays from the end to avoid unnecessary shifting.
- Learnt the time complexity analysis of merging sorted arrays in $O(m + n)$ time.

PROBLEM-2

- 1. Aim(b):** 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 $[1, 2, \dots, n]$ 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. Objective:** The objective of this program is to efficiently find the first bad version in a sequence of product versions using the `isBadVersion(version)` API, while minimizing the number of API calls by implementing a binary search approach.

3. Algorithm:

- Initialize Pointers: Set $left = 1$ and $right = n$ to define the search range.
- Perform Binary Search: While $left < right$:
 - Compute $mid = left + (right - left) / 2$.
 - If `isBadVersion(mid)` is true, update $right = mid$ (search in the left half).
 - Else, update $left = mid + 1$ (search in the right half).
- End Condition: When $left == right$, it points to the first bad version.
- Return the Result: Return $left$ as the first bad version.

4. Code:

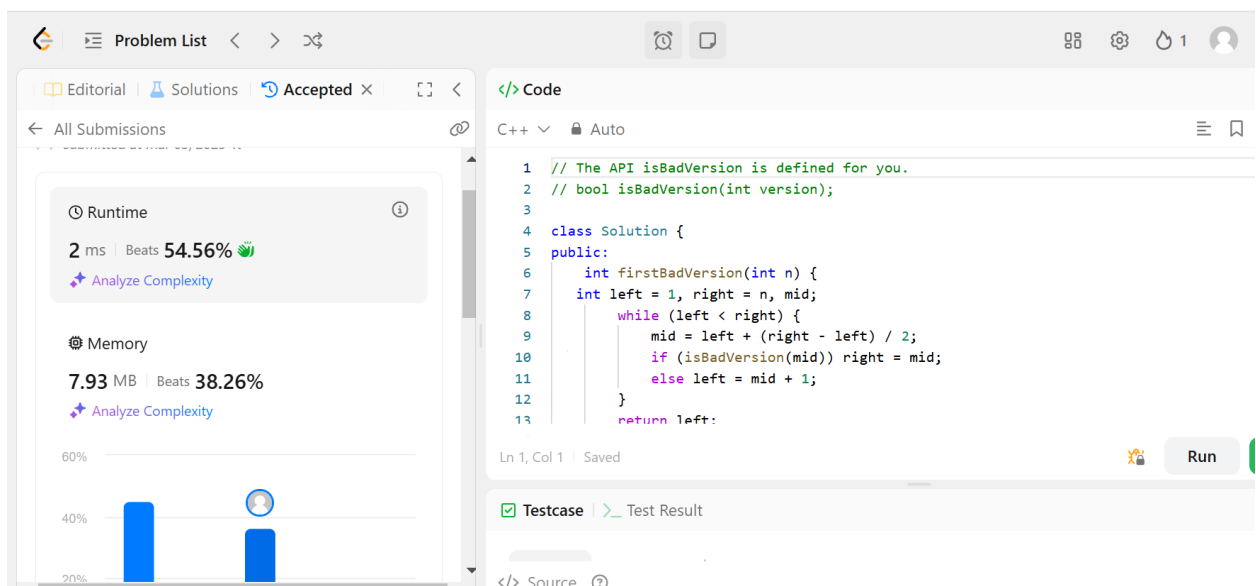
```
class Solution {  
public:  
    int firstBadVersion(int n)
```

```
{  
    int left = 1, right = n, mid;  
    while (left < right) {  
        mid = left + (right - left) / 2;  
        if (isBadVersion(mid))  
        {  
            right = mid;  
        } else  
        {  
            left = mid + 1;  
        }  
    }  
    return left;  
}  
};
```

LeetCode Link:

<https://leetcode.com/problems/first-bad-version/submissions/1559270518/>

5. Output:



6. Time Complexity:

The time complexity of this code is $O(\log n)$, where n represents the total number of product versions, numbered from 1 to n .

7. Learning outcomes:

- Learnt how to apply binary search to efficiently solve problems with ordered data.
- Learnt how to identify the first occurrence of a condition in a sorted sequence.
- Learnt how to identify the first occurrence of a condition in a sorted sequence.
- Learnt how to handle real-world scenarios like version control and quality testing efficiently.