Experiment 1

Student Name: Karan Goyal **UID:-22BCS15864**

Branch: BE-CSE **Section/Group: Ntpp-602-A**

Semester: 5TH **Dateof Performance:8-1-25**

Subject Name:-Advance Programming lab-2 **Subject Code:-22CSP-351**

Aim:-Given an integer array nums sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**. Then return *the number of unique elements in* nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

- Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.
- Return k

Objective:-The Ojective of this problem is to **remove duplicates** from a given **sorted integer array** in-place while maintaining the relative order of unique elements. The function should return the number of unique elements, denoted as kk, while modifying the array such that the first kk elements contain only the unique values. The remaining elements beyond kk are not considered important. The solution should achieve this **without using extra space**, modifying the array in-place with O(1)O(1) additional space.

Apparatus Used:

1. **Software**: -Leetcode
2. **Hardware**: Computer with 4 GB RAM and keyboard.

Algorithm for the Two Sum Problem:

1. Check if the array is empty:
    - If nums is empty, return 0 because there are no unique elements.
2. Initialize two pointers:
    - Set i = 1 (to track the position for unique elements).
    - Start iterating from j = 1 to compare the current element with the previous unique element.
3. Iterate through the array:
    - For each j, compare nums[j] with nums[i - 1] (the last unique element).
4. Identify unique elements:
    - If nums[j] is different from nums[i - 1], it means nums[j] is a unique element.
    - Set nums[i] = nums[j] and increment i.
5. Continue the iteration:
    - Continue iterating through the array, repeating step 4 for all elements in the array.
    - Return the result:Return i, which represents the number of unique elements in the array.

This algorithm effectively moves the unique elements to the front, ensuring in-place modification.

Code:

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if (nums.empty()) return 0;

        int i = 1;

        for (int j = 1; j < nums.size(); j++) {
            if (nums[j] != nums[i - 1]) {
                nums[i] = nums[j];
                i++;
            }
        }

        return i;
    }
};
```

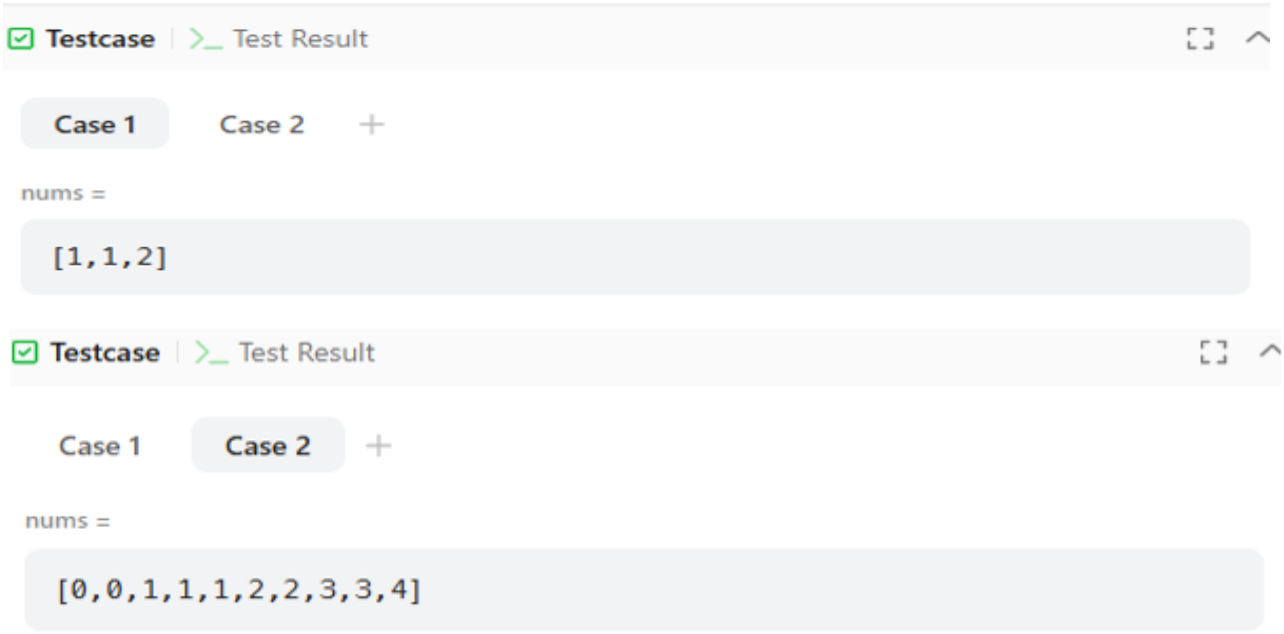
Time Complexity:

- **Time Complexity:** O(n)O(n), where nn is the number of elements in the input array nums.
    - The algorithm uses a **single loop** to traverse the array once, comparing and shifting elements. Each element is visited exactly once.

Space Complexity:

- **Space Complexity:** O(1)O(1), because the algorithm uses **constant extra space**.
    - It only uses a few integer variables (i and j) for indexing, and the modification is done **in-place** without requiring additional data structures.

Output- All the test cases passed



Problem Statement: Given an integer array nums and an integer val, remove all occurrences of val in nums **in-place**. The order of the elements may be changed. Then return *the number of elements in* nums *which are not equal to* val.

Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the following things:

- Change the array nums such that the first k elements of nums contain the elements which are not equal to val. The remaining elements of nums are not important as well as the size of nums.
- Return k.

Objective:- The objective of this problem is to remove all occurrences of a given integer val from an integer array nums in-place. The order of elements may change. After removing the occurrences, the array should be modified such that the first k elements are not equal to val. Return k.

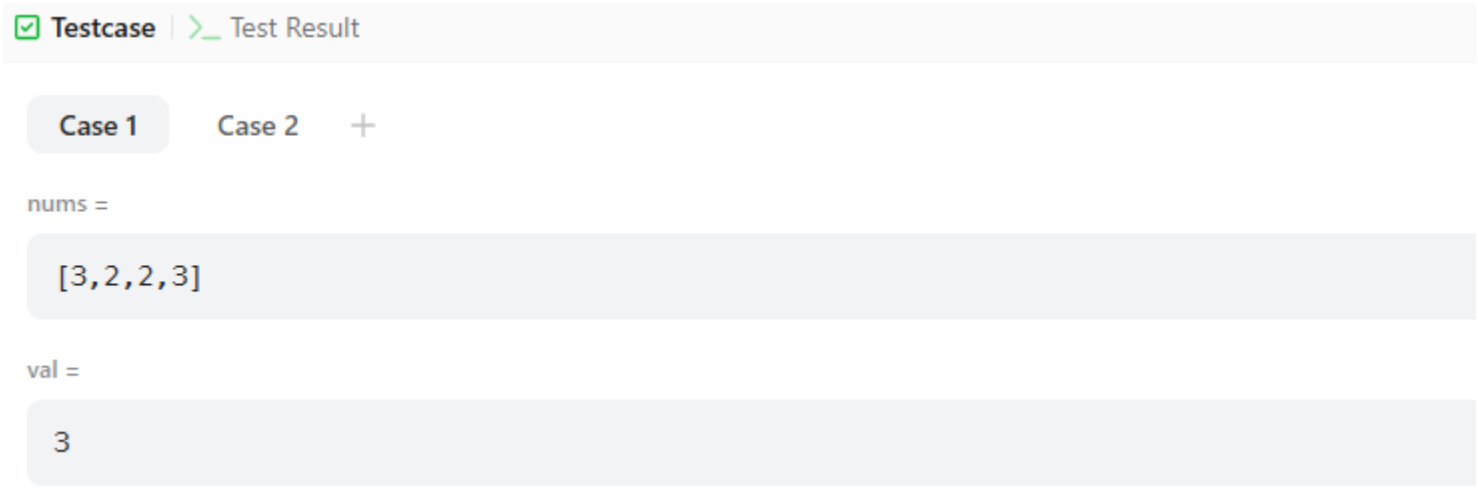
Algorithm

1. Initialize an index variable index to 0. This will track the position of the next element that is not equal to val.
2. Loop through the array nums from index i = 0 to i < nums.size().
3. Inside the loop, check if the current element nums[i] is not equal to val.
4. If nums[i] is not equal to val, assign nums[index] = nums[i], and increment index.
5. Continue the loop until all elements in the array are checked.
6. Return the value of index, which represents the number of elements not equal to val in the array.

Code:-

```
class Solution {

public:

   int removeElement(vector<int>& nums, int val) {

      int index = 0;

      for(int i = 0; i< nums.size(); i++){

         if(nums[i] != val){

            nums[index] = nums[i];

            index++;

         } }

      return index;

   }};
```

Result-All test cases passes



Learning Outcomes:

1. Understand how to modify arrays in-place without using extra space.
2. Learn how to remove specific elements from a sorted array while maintaining the order of remaining elements.
3. Develop techniques for eliminating duplicates in a sorted array while keeping unique elements in their original order.
4. Practice updating an array in-place and returning the count of unique or non-matching elements.
5. Enhance problem-solving skills by optimizing array manipulation tasks with linear time complexity.