Experiment 5

Student Name: Ankit Kumar Yadav **UID: 22BCS11046**
Branch: BE-CSE **Section/Group: 638_A**
Semester: 6th **Date of Performance:18/02/25**
Subject Name: Advance Programming Lab -2 **Subject Code: 22CSP-351**

1. Aim:

Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue. We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively. You must solve this problem without using the library's sort function.

Example 1:

Input: nums = [2,0,2,1,1,0] Output: [0,0,1,1,2,2]

Example 2:

Input: nums = [2,0,1] Output: [0,1,2]

Link:- Sort Colors - LeetCode

2. Objective:

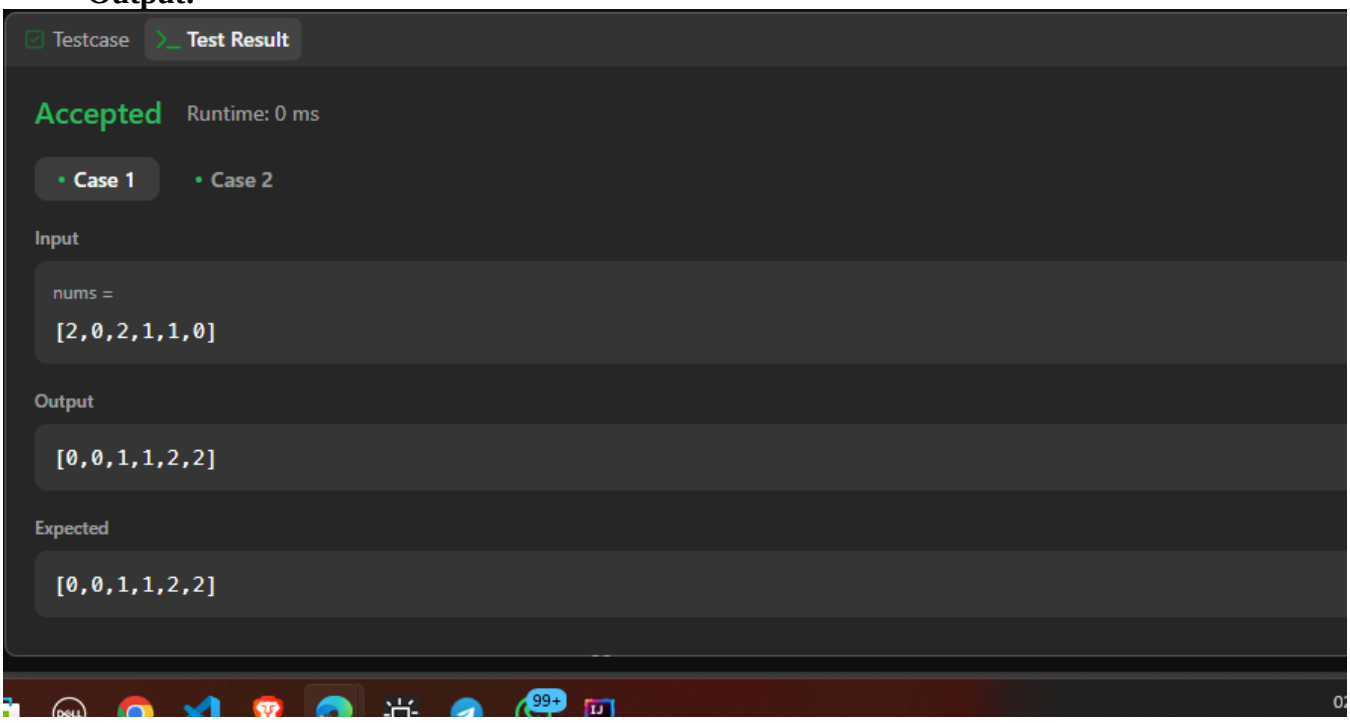
The task is to sort an array representing colored objects (red, white, and blue) in-place. These colors are represented by integers 0, 1, and 2, respectively. The goal is to arrange the array such that elements of the same color are grouped together and the colors appear in the order red, white, then blue. A key constraint is to achieve this sorting without using any built-in library sorting functions, focusing on a manual sorting approach. The in-place requirement means modifying the original array directly, without creating a new one. The solution should be efficient and adhere to the specified color order

3. Implementation/Code:

```
class Solution {
    public void sortColors(int[] nums) {
        int red = 0;
        int white = 0;
        int blue = 0;

        for (int num : nums) {
            if (num == 0) {
                red++;
            } else if (num == 1) {
                white++;
            } else {
                blue++;
```

```
            }
        }
        int index = 0;
        for (int i = 0; i < red; i++) {
            nums[index++] = 0;
        }
        for (int i = 0; i < white; i++) {
            nums[index++] = 1;
        }
        for (int i = 0; i < blue; i++) {
            nums[index++] = 2;
        }
    }
}
```

Output:-



4. Learning outcomes:

- Understanding in-place algorithms.
- Proficiency in array manipulation.
- Ability to implement sorting logic without library functions.
- Knowledge of the Dutch National Flag algorithm (or similar partitioning techniques).
- Analyzing space and time complexity for sorting algorithms.

1. Aim:

Given an integer array nums and an integer k, return the kth largest element in the array. Note that it is the kth largest element in the sorted order, not the kth distinct element. Can you solve it without sorting?

Example 1:

Input: nums = [3,2,1,5,6,4], k = 2 Output: 5

Example 2:

Input: nums = [3,2,3,1,2,4,5,5,6], k = 4 Output: 4

Link:- Kth Largest Element in an Array - LeetCode

2. Objective:

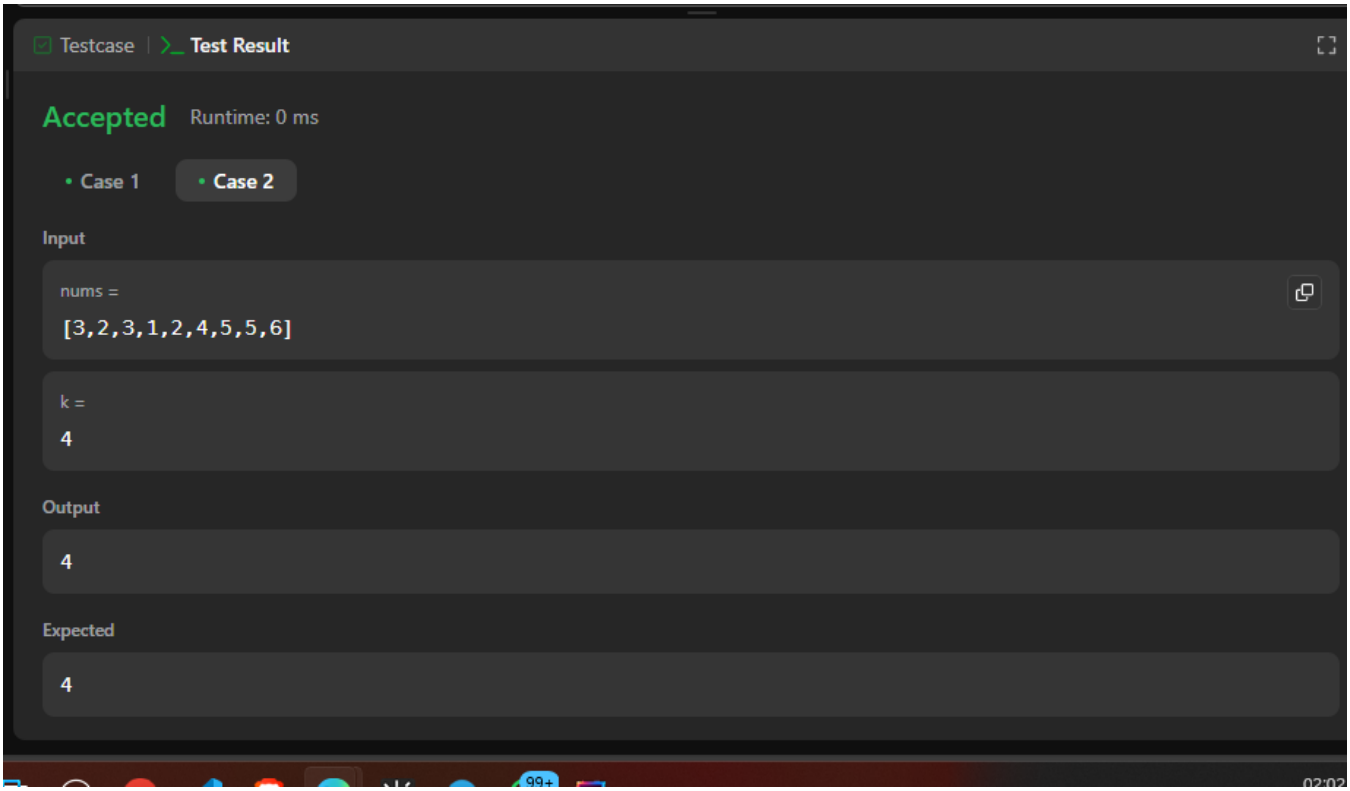
The objective is to efficiently identify the kth largest element within a given integer array. The target element is determined by its position in the sorted order of the array, not by its uniqueness. The challenge lies in achieving this without resorting to a full sorting of the array, implying the need for a more optimized approach. Efficiency is key, especially considering the potential size of the input array. The goal is to return the value of the kth largest element.

3. Implementation/Code:

```
class Solution {
    public int findKthLargest(int[] nums, int k) {
        return quickSelect(nums, 0, nums.length - 1, k);
    }

    private int quickSelect(int[] nums, int left, int right, int k) {
        Random random = new Random();
        int pivotIndex = left + random.nextInt(right - left + 1);
        pivotIndex = partition(nums, left, right, pivotIndex);

        if (k == nums.length - pivotIndex) {
            return nums[pivotIndex];
        } else if (k < nums.length - pivotIndex) {
            return quickSelect(nums, pivotIndex + 1, right, k);
        } else {
            return quickSelect(nums, left, pivotIndex - 1, k);
        }
    }
    private int partition(int[] nums, int left, int right, int pivotIndex) {
        int pivotValue = nums[pivotIndex];
        swap(nums, pivotIndex, right);
        int storeIndex = left;

        for (int i = left; i < right; i++) {
            if (nums[i] < pivotValue) {
                swap(nums, i, storeIndex);
                storeIndex++;
```

```
            }
        }
        swap(nums, storeIndex, right);
        return storeIndex;
    }

    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;

    }
     }
```

Output:-



5. Learning outcomes:

- Understanding selection algorithms (e.g., Quickselect).
- Knowledge of partitioning techniques.
- Ability to analyze the average and worst-case time complexity of selection algorithms.
- Understanding the importance of pivot selection in Quickselect.