



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

### Experiment-5

**Student Name:** Pulkit

**UID** 22BCS11634

**Branch:** BE-CSE

**Section/Group:** 22BCS\_IOT-638/B

**Semester:** 6th

**Date of Performance:** 21/01/2025

**Subject Name:** AP LAB-II

**Subject Code:** 22CSP-351

#### **1. Problem Statement : Sort Colors**

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]`

Constraints:

`n == nums.length`  $1 \leq n \leq 300$  `nums[i]` is either 0, 1, or 2.

Follow up: Could you come up with a one-pass algorithm using only constant extra space?

#### **Implementation/Code:**

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        unordered_map<int, int> count = {{0, 0}, {1, 0}, {2, 0}};

        for (int num : nums) {
            count[num]++;
        }

        int idx = 0;
        for (int color = 0; color < 3; color++) {
```



DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
int freq = count[color];
for (int j = 0; j < freq; j++) {
    nums[idx] = color;
    idx++;
}
}
}
};
```

**Output:**

The screenshot shows a code execution interface with a dark theme. At the top, there are tabs for 'Testcase' and 'Test Result', with 'Test Result' being the active tab. Below the tabs, the word 'Accepted' is displayed in green, followed by 'Runtime: 0 ms'. There are two tabs for test cases, 'Case 1' and 'Case 2', with 'Case 1' being the active one. Under the 'Input' section, the text 'nums =' is followed by the array '[2,0,2,1,1,0]'. Under the 'Output' section, the array '[0,0,1,1,2,2]' is shown. Under the 'Expected' section, the array '[0,0,1,1,2,2]' is shown, matching the output.

- **Time Complexity:**  $O(n)$
- **Space Complexity:**  $O(1)$

## 2. Problem Statement: Kth Largest Element in an Array

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

Example 1:

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



**DEPARTMENT OF**

# **COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

Example 2:

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

Constraints:

$1 \leq k \leq \text{nums.length} \leq 10^5$   $-10^4 \leq \text{nums}[i] \leq 10^4$

## **Implementation/Code:**

```
#include <vector>
#include <queue>

using namespace std;

class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
        priority_queue<int, vector<int>, greater<int>> minHeap;

        for (int num : nums) {
            minHeap.push(num);
            if (minHeap.size() > k) {
                minHeap.pop();
            }
        }

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



DEPARTMENT OF

# COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Output:

☒ Testcase | [Test Result](#)

**Accepted** Runtime: 0 ms

[Case 1](#) • [Case 2](#)

**Input**

nums =  
[3,2,1,5,6,4]

k =  
2

**Output**

5

**Expected**

5

- **Time Complexity:**  $O(n \log n)$
- **Space Complexity:**  $O(\text{sort})$

## Learning Outcomes:

### Problem 1: Sort Colors

1. **Understanding the Three-Way Partitioning** – Learn how to efficiently categorize elements into three groups using a single-pass algorithm.
2. **In-Place Sorting** – Gain experience in sorting an array without using extra space.
3. **Two-Pointer Technique** – Understand the use of multiple pointers to optimize sorting problems.
4. **Time Complexity Optimization** – Learn how to achieve an optimal  **$O(n)$  time complexity** instead of the naive  **$O(n \log n)$**  sorting approach.

### Problem 2: Kth Largest Element in an Array

1. **Selection Algorithms** – Learn about QuickSelect, a variant of QuickSort, to efficiently find the kth largest element.



**DEPARTMENT OF**

# **COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

2. **Partitioning Strategies** – Understand how to divide an array into smaller segments to reduce search space.
3. **Heap Data Structure (Alternative Approach)** – Learn how to use Min-Heap or Max-Heap to solve selection problems in  **$O(n \log k)$  time**.
4. **Average Case Optimization** – Gain insights into reducing time complexity to  **$O(n)$  average time** instead of sorting in  **$O(n \log n)$** .