



Experiment 5

Student Name: Akash Kumar Singh

UID: 22BCS15769

Branch: B.E-C.S.E

Section/Group: IOT_638-B

Semester: 6th

Date of Performance: 21/2/25

Subject Name: Advanced Programming Lab

Subject Code: 22CSP-351

1. Aim:

Problem-1: Sort Colors

2. Objective:

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.

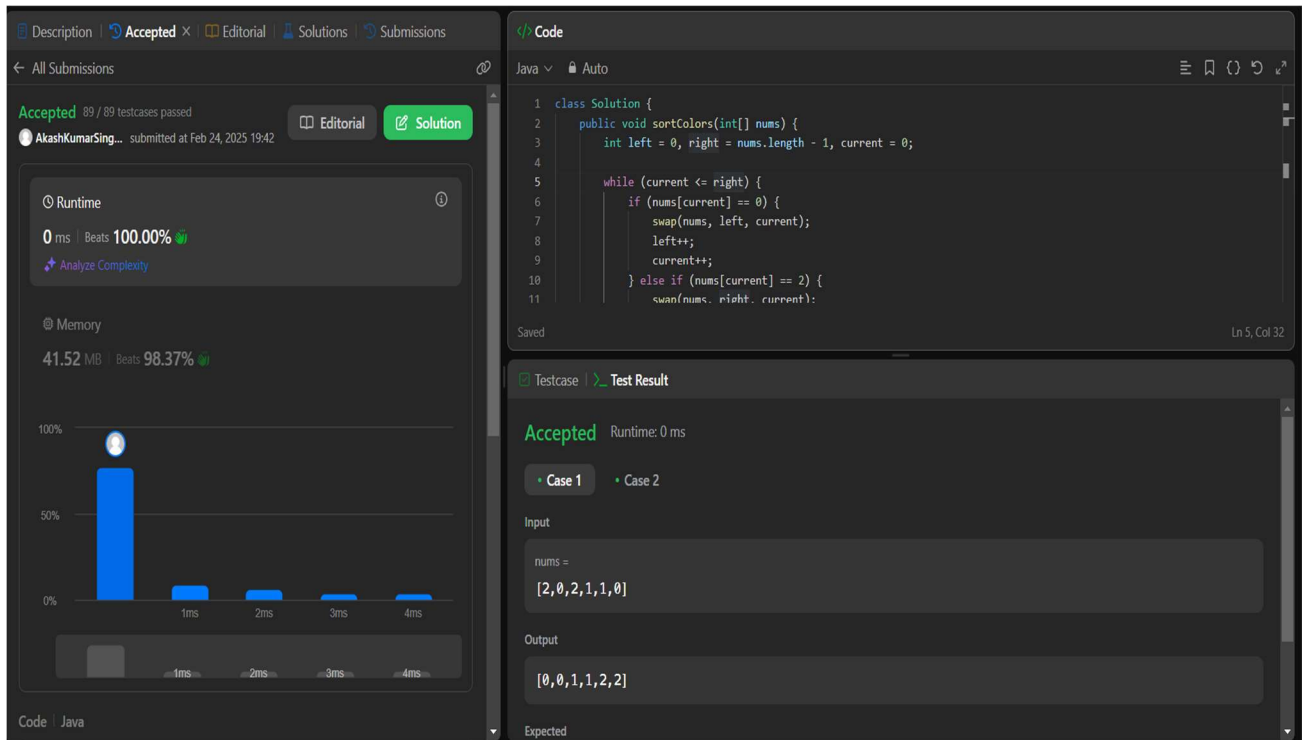
3. Implementation/Code:

```
class Solution {  
    public void sortColors(int[] nums) {  
        int left = 0, right = nums.length - 1, current = 0;  
  
        while (current <= right) {  
            if (nums[current] == 0) {  
                swap(nums, left, current);  
                left++;  
                current++;  
            } else if (nums[current] == 2) {  
                swap(nums, right, current);  
                right--;  
            } else {  
                current++;  
            }  
        }  
    }  
}
```

```
    }
  }
}
```

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

4. Output



The screenshot displays a coding platform interface for a Java solution. The top navigation bar includes tabs for Description, Accepted (selected), Editorial, Solutions, and Submissions. The main content area shows the submission details for a user named AkashKumarSingh, submitted on Feb 24, 2025, at 19:42. The solution is marked as Accepted, with 89 / 89 testcases passed. The runtime is 0 ms, and the memory usage is 41.52 MB. A bar chart shows the runtime performance across different test cases, with the first case being the most time-consuming. The code editor on the right shows the Java code for the solution, which implements a sorting algorithm. The test case section shows the input array [2,0,2,1,1,0] and the expected output [0,0,1,1,2,2].

Accepted 89 / 89 testcases passed

Submitted by: AkashKumarSingh... submitted at Feb 24, 2025 19:42

Runtime
0 ms | Beats 100.00%

Memory
41.52 MB | Beats 98.37%

Code

```
1 class Solution {
2     public void sortColors(int[] nums) {
3         int left = 0, right = nums.length - 1, current = 0;
4
5         while (current <= right) {
6             if (nums[current] == 0) {
7                 swap(nums, left, current);
8                 left++;
9                 current++;
10            } else if (nums[current] == 2) {
11                swap(nums, right, current);
12            }
13        }
14    }
15}
```

Testcase | **Test Result**

Accepted Runtime: 0 ms

Case 1 | **Case 2**

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

Output
[0,0,1,1,2,2]

Expected



Problem-2: Kth Largest Element in an Array

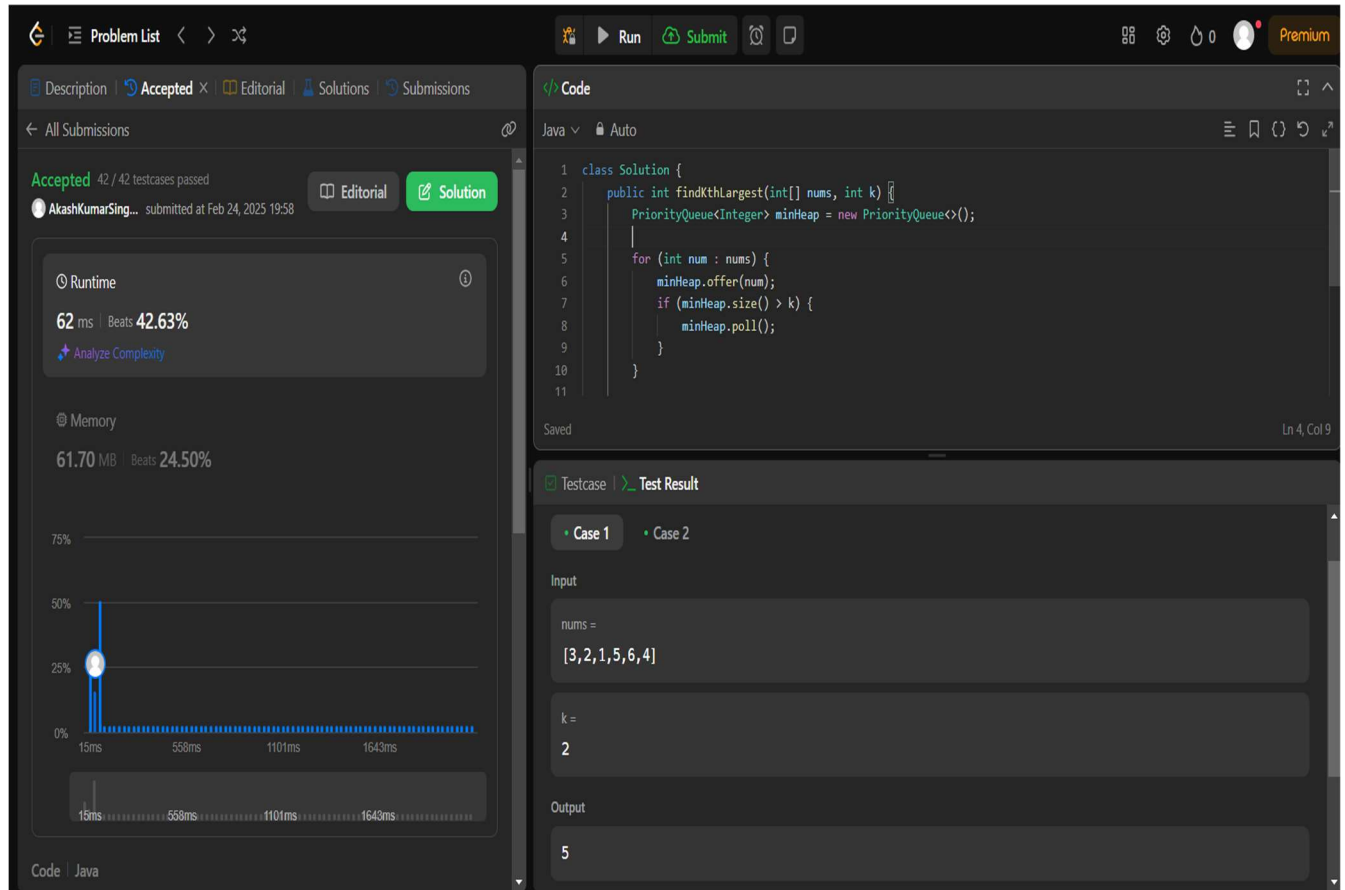
1. Objective:

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.

2. Implementation/Code:

```
class Solution {  
    public int findKthLargest(int[] nums, int k) {  
        PriorityQueue<Integer> minHeap = new PriorityQueue<>();  
  
        for (int num : nums) {  
            minHeap.offer(num);  
            if (minHeap.size() > k) {  
                minHeap.poll();  
            }  
        }  
  
        return minHeap.peek();  
    }  
}
```

3. Output:



4. Learning Outcomes:

1. Understand and implement in-place sorting algorithms like the **Dutch National Flag Algorithm** for efficiently sorting a three-color array.
2. Learn how to find the **Kth largest element** using optimized approaches like **QuickSelect** or **Min-Heap**, avoiding full sorting.
3. Analyze and compare **time and space complexity** of different sorting and selection algorithms for better optimization.
4. Develop problem-solving skills in **array manipulation and partitioning techniques** for constrained problems.
5. Gain experience in handling **edge cases and constraints** while ensuring algorithm correctness and efficiency.