

NAME-Manvir Singh

UID-22BCS10450

BRANCH-BE-CSE

SECTION/GROUP-IOT-640-B

SEMESTER-6

DATE OF PERFORMANCE-12/02/2023

SUBJECT NAME-AP-LAB II

SUBJECT CODE-22CSP-351

ASSIGNMENT -5

PROBLEM 1: 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 <= n <= 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?

2.CODE

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
class Solution { public:
```

```
    void sortColors(vector<int>& nums) {
```

```

        int low = 0, mid = 0, high = nums.size() - 1;
while (mid <= high) {
    if (nums[mid] == 0) {
        swap(nums[low], nums[mid]);
        low++;
        mid++;
    } else if (nums[mid] == 1) {
        mid++;
    } else { // nums[mid] == 2
        swap(nums[mid], nums[high]);
        high--;
    }
}
};

```

```

int main() {
    vector<int> nums = {2,
0, 2, 1, 1, 0};
    Solution sol;
    sol.sortColors(nums);

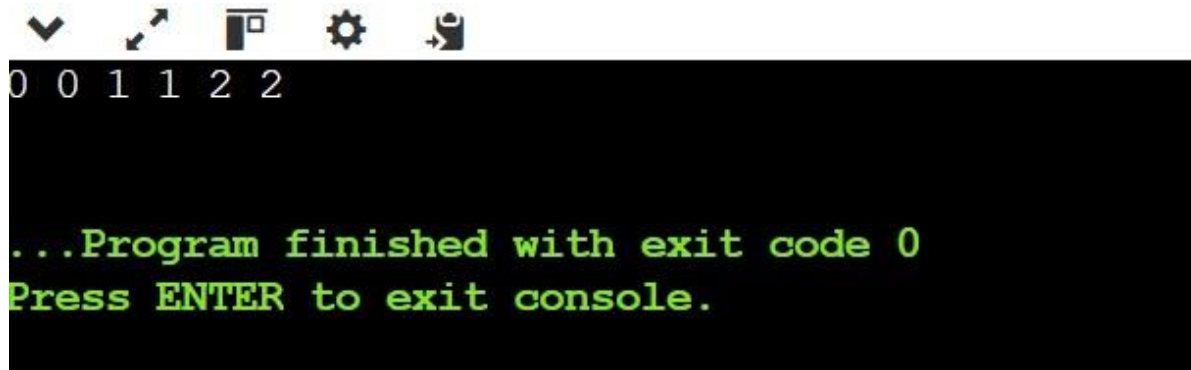
```

```

    for (int num : nums) {
        cout << num << " ";
    }
    cout << endl;
    return 0;
}

```

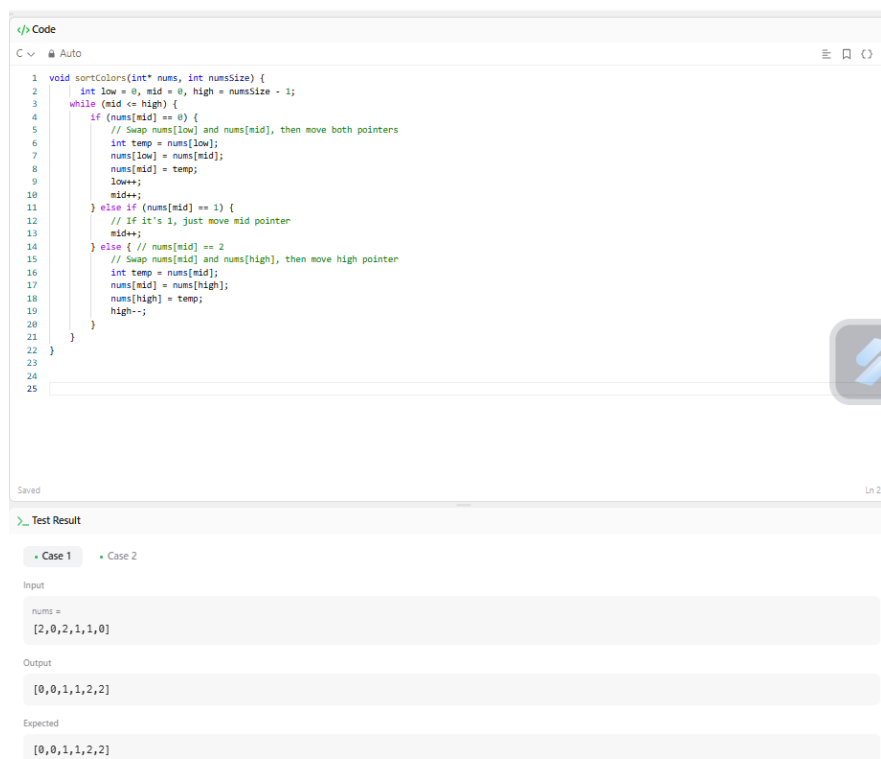
OUTPUT:



```
0 0 1 1 2 2

...Program finished with exit code 0
Press ENTER to exit console.
```

LEETCODE:



```
1 void sortColors(int* nums, int numsSize) {
2     int low = 0, mid = 0, high = numsSize - 1;
3     while (mid <= high) {
4         if (nums[mid] == 0) {
5             // Swap nums[low] and nums[mid], then move both pointers
6             int temp = nums[low];
7             nums[low] = nums[mid];
8             nums[mid] = temp;
9             low++;
10            mid++;
11        } else if (nums[mid] == 1) {
12            // If it's 1, just move mid pointer
13            mid++;
14        } else { // nums[mid] == 2
15            // Swap nums[mid] and nums[high], then move high pointer
16            int temp = nums[mid];
17            nums[mid] = nums[high];
18            nums[high] = temp;
19            high--;
20        }
21    }
22 }
23
24
25
```

Test Result

Case 1 Case 2

Input

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

Output

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

Expected

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

LEETCODE LINK:

<https://leetcode.com/problems/sort-colors/>LEARNING OUTCOMES:

LEARNING OUTCOMES:

1. **Understanding the Dutch National Flag Algorithm** – Learned how to efficiently sort an array containing three distinct elements (0, 1, 2) in **one pass ($O(n)$)** using a **three-pointer approach**.
2. **In-Place Sorting Without Extra Space** – Developed skills to sort the array **without using extra memory ($O(1)$)**, making the solution space-efficient.
3. **Efficient Array Manipulation with Swap Operations** – Gained hands-on experience in swapping elements strategically using **low, mid, and high pointers** to ensure correct ordering.
4. **Optimizing Sorting Without Using Built-in Functions** – Learned how to manually implement sorting logic **without relying on `sort()`**, which is useful for interviews and competitive programming.

PROBLEM 2: 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 Example

2:

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

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

CODE:

```
#include <iostream>

#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(); // Remove smallest element to maintain size k
    }
}

    return minHeap.top(); // The root of the min-heap is the kth largest element
}
};

int main() {
    vector<int> nums = {3, 2, 3, 1, 2, 4, 5, 5, 6};
int k = 4;
    Solution sol;
    cout << sol.findKthLargest(nums, k) << endl; // Output: 4
return 0;
}

```

OUTPUT:



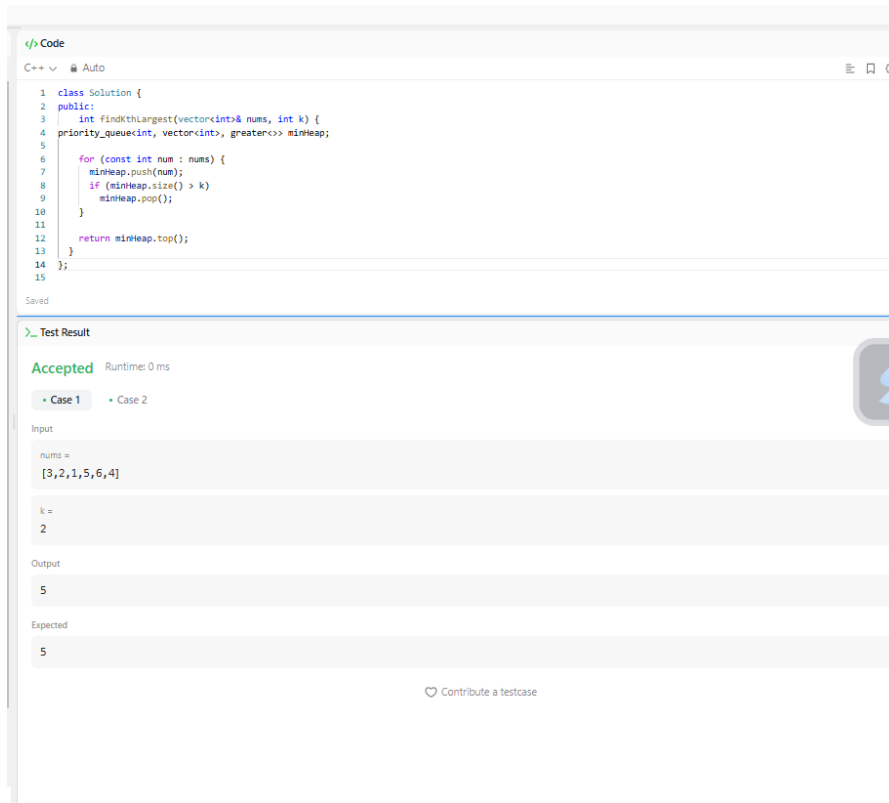
```

4

...Program finished with exit code 0
Press ENTER to exit console.

```

LEETCODE :



```
</> Code
C++ v Auto
1 class Solution {
2 public:
3     int findKthLargest(vector<int>& nums, int k) {
4         priority_queue<int, vector<int>, greater<>> minHeap;
5
6         for (const int num : nums) {
7             minHeap.push(num);
8             if (minHeap.size() > k)
9                 minHeap.pop();
10        }
11
12        return minHeap.top();
13    }
14 };
15
```

Saved

> Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

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

k =
2

Output

5

Expected

5

Contribute a testcase

LEETCODE LINK:

<https://leetcode.com/problems/kth-largest-element-in-an-array/>

LEARNING OUTCOMES:

- 1. Understanding Heap Data Structure** – Learned how to use a Min-Heap (Priority Queue) to efficiently find the k-th largest element in $O(n \log k)$ time complexity.
- 2. Optimized Selection Without Sorting** – Developed the ability to find the k-th largest element without sorting ($O(n \log n)$), using a more efficient approach like Heap or Quickselect ($O(n)$ average).
- 3. Efficient Space Utilization** – Gained experience in solving problems using $O(k)$ extra space for the Min-Heap, making it memory-efficient compared to full sorting.
- 4. Application of Quickselect Algorithm** – Learned how to apply the Quickselect Algorithm ($O(n)$ average case), a variation of QuickSort, to efficiently find the k-th largest element in an unordered list.

