Experiment 5

Student Name: Shivam Yadav **UID: 22BCS15259**
Branch: CSE **Section: 638/A**
Semester: 6th **DOP: 18-02-2014**
Subject: AP **Subject Code:22CSP-351**

Aim:

Problem-1: Sort Colors

Algorithm:

- **Start with Base Case**
- Begin with a list containing only [1] as the base case.
- **Build Odd and Even Sequences**
- Use a divide-and-conquer approach:
  - Generate odd numbers: 2 * num - 1 (as long as they are ≤ n).
  - Generate even numbers: 2 * num (as long as they are ≤ n).
- Append these numbers in order to ensure no three numbers satisfy 2 * nums[k] == nums[i] + nums[j].
- **Convert List to Array and Return**
- Store the result in an integer array and return it

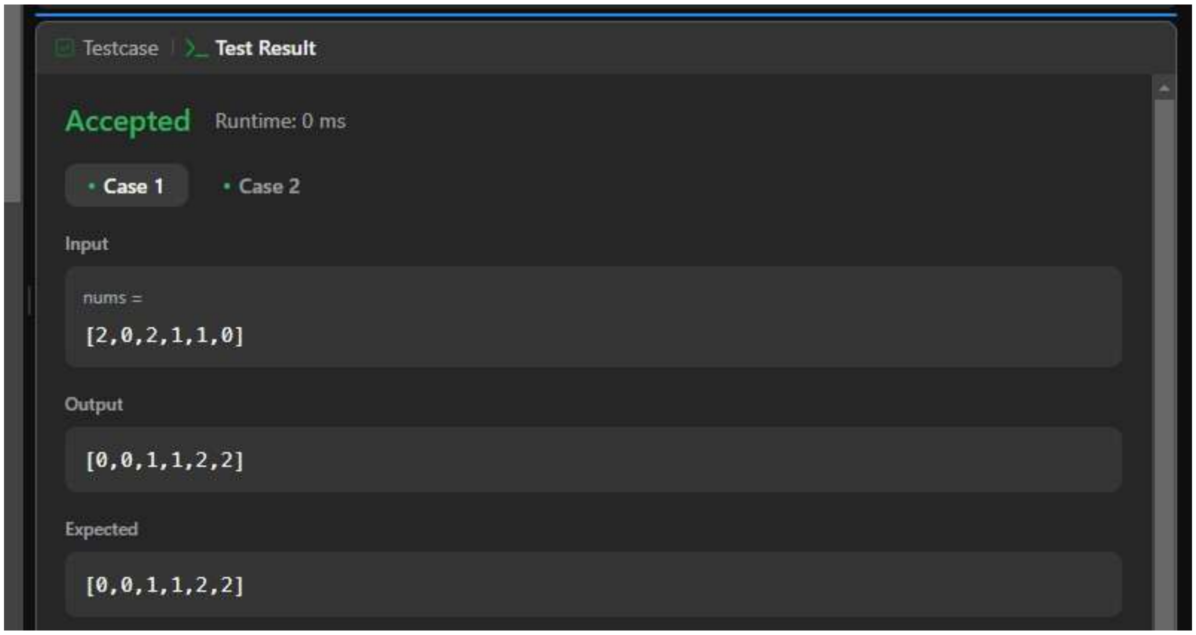
Code:

```
public class Solution {
    public void sortColors(int[] nums) {
        int start = 0;
        int end = nums.length - 1;
        int current = 0;
        while (current <= end) {
            if (nums[current] == 0) {
                moveToStart(nums, start, current);
                start++;
                current++;
            } else if (nums[current] == 1) {
                current++;
            } else {
                moveToEnd(nums, current, end);
                end--;
            }
        }
    }
    private void moveToStart(int[] nums, int start, int current) {
        int temp = nums[start];
        nums[start] = nums[current];
        nums[current] = temp;
    }
    private void moveToEnd(int[] nums, int current, int end) {
        int temp = nums[end];
        nums[end] = nums[current];
        nums[current] = temp;
    }
    public static void main(String[] args) {
        Solution sorter = new Solution();
        int[] nums = {2, 0, 2, 1, 1, 0};
        sorter.sortColors(nums);

        for (int num : nums) {
            System.out.print(num + " ");
        }
    }
}
```

Link:- https://leetcode.com/problems/sort-colors/

Output:



Accepted Runtime: 0 ms

Input

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

Output

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

Expected

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

Aim:

Problem-2: Kth Largest Element in an Array

Algorithm :

Use a Min-Heap (Priority Queue)

- Create a min-heap (PriorityQueue) to store the top k largest elements. Iterate Over the Array
- Insert each number into the min-heap.
- If the heap size exceeds k, remove the smallest element to keep only the top k largest elements.

Return the Kth Largest Element

- The root of the min-heap (peek()) gives the kth largest element in the array.

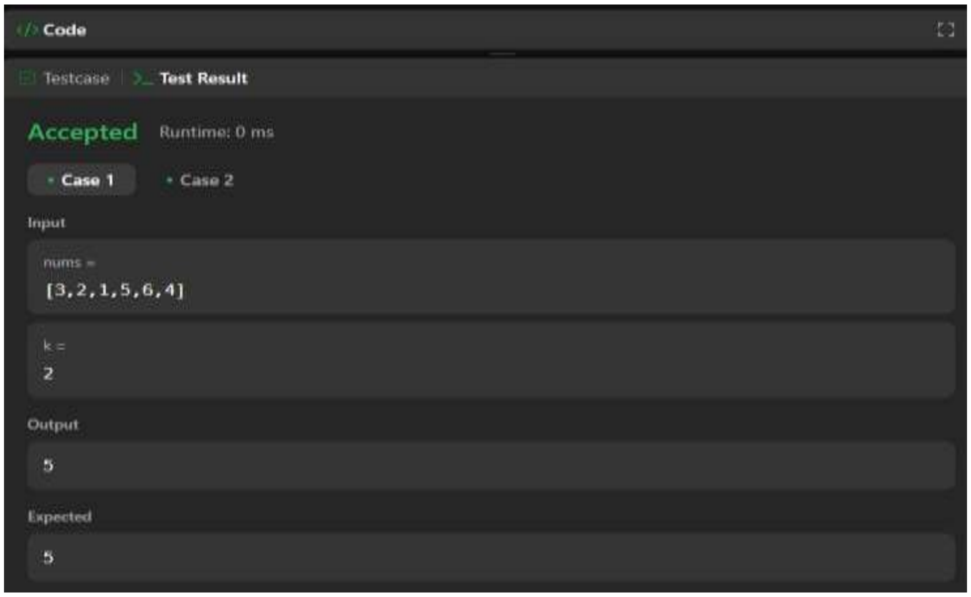
Code :

```
import java.util.PriorityQueue;

public class Solution {
    public int findKthLargest(int[] nums, int k) {
        PriorityQueue<Integer> minHeap = new PriorityQueue<>();
        for (int num : nums) {
            minHeap.add(num);
            if (minHeap.size() > k) {
                minHeap.poll();
            }
        }
        return minHeap.peek();
    }
    public static void main(String[] args) {
        Solution obj = new Solution();
        int[] nums = {3, 2, 1, 5, 6, 4};
        int k = 2;
        System.out.println("The " + k + "th largest element is: " + obj.findKthLargest(nums, k));
    }
}
```

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

Output:



```
Accepted   Runtime: 0 ms

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

k =
2

Output
5

Expected
5
```

Learing outcome:-

- You have learned how to efficiently find the k-th largest element in an unsorted array using a **min-heap**.
- You understand how to manage a heap of fixed size k and how the heap's properties help in solving this problem optimally.
- **Three-Pointer Technique:**
  The code introduces the concept of using three pointers (start, end, and current) to partition and sort the array in a single pass.
- **Swapping Elements:**
  The solution makes use of element swaps to move elements to their correct positions, showcasing how sorting can be done in place with minimal space usage.