Experiment 4-A

Student Name: Shristi Rai
Branch: CSE
Semester: 6th
Subject Name: AP Lab-2

UID: 22BCS15330
Section/Group: NTPP-602-A
Date of Performance:10/02/25
Subject Code: 22CSH-352

1. **TITLE:** Sort Colors
2. **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.

3. **Algorithm**

   - Initialize counters for zeros and ones.
   - Traverse the list and count the number of zeros and ones.
   - Overwrite the original list: first with zeros, then ones, and finally twos (the remainder).

4. **Implemetation/Code**

```
import java.util.*;

class Solution {
   public int search(int[] nums, int target) {
      int left = 0, right = nums.length - 1;

      while (left <= right) {
```

```
            int mid = left + (right - left) / 2;

            if (nums[mid] == target) {
                return mid;
            }
            if (nums[left] <= nums[mid]) {
                if (nums[left] <= target && target < nums[mid]) {

                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            } else {
                if (nums[mid] < target && target <= nums[right]) {
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }
            }   }

        return -1;
    }

    public void sortColors(int[] nums) {
        int zeros = 0, ones = 0, n = nums.length;

        // Count the occurrences of 0s and 1s
        for (int num : nums) {
            if (num == 0) {
                zeros++;
            } else if (num == 1) {
                ones++;
            }
        }

        // Place 0s, 1s, and 2s in the array
        for (int i = 0; i < zeros; i++) {
            nums[i] = 0;
        }
```

```
            for (int i = zeros; i < zeros + ones; i++) {
                nums[i] = 1;
            }
            for (int i = zeros + ones; i < n; i++) {
                nums[i] = 2;
            }
        }
    }
```

5. Output

6. Time Complexity: O(n)

7. Space Complexity: O(1)

8. Learning Outcomes:

- Learn the principles behind counting sort.
- Manipulate array indices and values to sort in-place

Experiment 4-B

1. **TITLE:** Search in Rotated Sorted Array

2. **AIM:**

   There is an integer array nums sorted in ascending order (with distinct values).

   Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

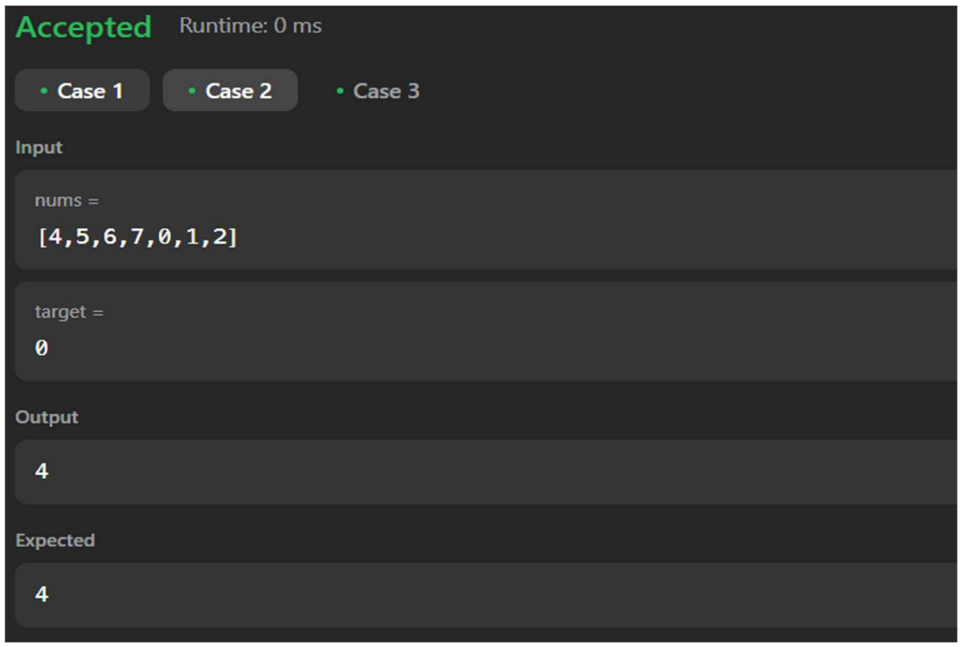
3. **Algorithm**
   - Initialize two pointers, left at the start and right at the end of the list.
   - Use a loop to repeatedly divide the list into halves.
   - If the middle element matches the target, return its index.
   - Adjust the left or right pointer based on the comparison between the target and the middle element.
   - If the target is not found by the end of the loop, return -1.

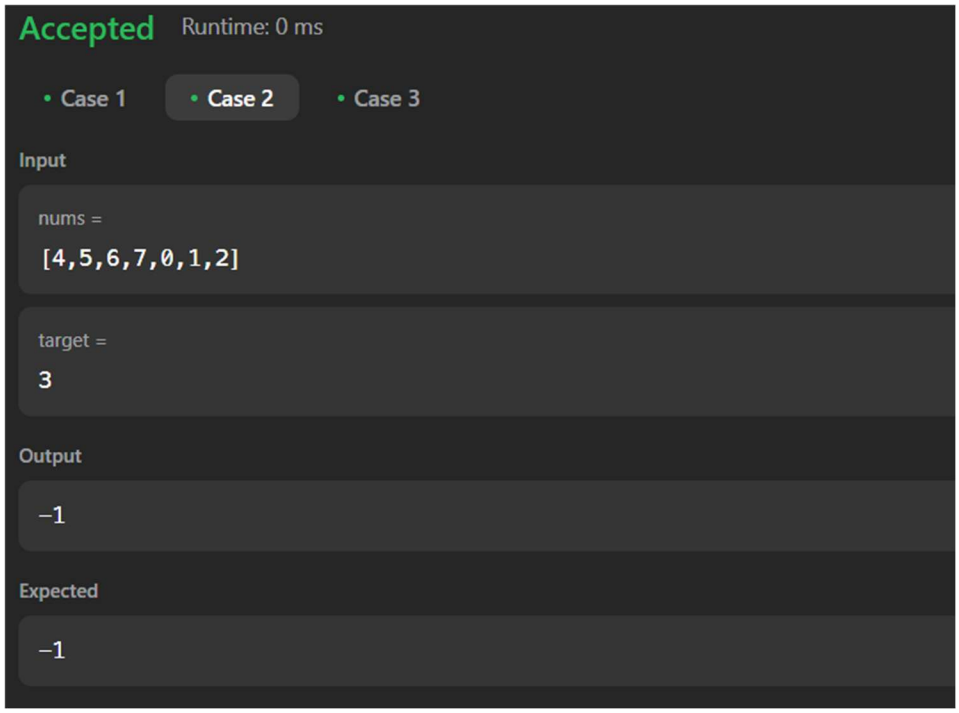
4. **Implemetation/Code:**

```
import java.util.*;

class Solution {
    public int search(int[] nums, int target) {
        int left = 0, right = nums.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] == target) {
                return mid;
            }

            if (nums[left] <= nums[mid]) {
                if (nums[left] <= target && target < nums[mid]) {
                    right = mid - 1;
```

```
                }
                else {
                    left = mid + 1;
                }
            } else {
                if (nums[mid] < target && target <= nums[right]) {
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }
            }
        }

        return -1;
    }

    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;
    }
}
```

5. Output





6. **Time Complexity**: O(log n)

7. **Space Complexity:** O(1)

8. **Learning Outcomes:**

- Learn how to implement and utilize binary search in a potentially rotated sorted array.
- Optimizing search operations significantly over linear scanning.
- Managing multiple conditions to direct search logic