**DAAOA Assignment 2**

**Name: Atharva Salitri**

**Roll No.: 029**

**Branch: TY CSAI-B**

**Batch: B2**

**PRN: 12310120**

**Title: Implement Recursive and Non Recursive Binary Search with Time and Space Complexity Analysis**

import java.util.Scanner;

public class BinarySearchExample {

    public static int recursiveBinarySearch(int[] arr, int low, int high, int x) {

        if (high >= low) {

            int mid = low + (high - low) / 2;

            if (arr[mid] == x)

                return mid;

            if (arr[mid] > x)

                return recursiveBinarySearch(arr, low, mid - 1, x);

            return recursiveBinarySearch(arr, mid + 1, high, x);

        }

        return -1;

    }

    public static int iterativeBinarySearch(int[] arr, int x) {

        int low = 0, high = arr.length - 1;

        while (low <= high) {

            int mid = low + (high - low) / 2;

            if (arr[mid] == x)

                return mid;

            if (arr[mid] < x)

                low = mid + 1;

            else

                high = mid - 1;

        }

        return -1;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++)

            arr[i] = sc.nextInt();

        int key = sc.nextInt();

        int resRec = recursiveBinarySearch(arr, 0, n - 1, key);

        System.out.println(resRec);

        int resIter = iterativeBinarySearch(arr, key);

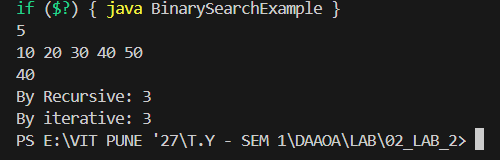
        System.out.println(resIter);

        sc.close();

    }

}

**OUTPUT:**

****

**Time and Space Complexity Analysis:**

**Recursive Binary Search:**

* Time Complexity:
  + Best Case: **O(1)** because if the middle element is the target, it takes just one comparison.
  + Average Case: **O(log n)** since the search space halves with each recursive call, reducing the problem size logarithmically.
  + Worst Case: **O(log n)** arises when the element is at one of the ends, requiring maximum recursive steps.
* Space Complexity: **O(log n)** due to the recursion call stack that holds at most one call per level down to log n levels.

**Iterative (Non-Recursive) Binary Search:**

* Time Complexity:
  + Best Case: **O(1)** when the middle element matches the target immediately.
  + Average Case: **O(log n)** because the halving of the search space continues iteratively until found.
  + Worst Case: **O(log n)** when the search narrows down to the edges of the array.
* Space Complexity: **O(1)** since only a few variables are used regardless of input size, with no recursion stack overhead.