Day 1: Linear Search and Binary Search

Abhinav Yadav

"Programs must be written for people to read, and only incidentally for machines to execute."
— Harold Abelson

---

1 Introduction

Searching is a fundamental operation in computer science used to find the location of a target element in a collection of data. This document covers two search algorithms:

- **Linear Search:** Suitable for unsorted arrays, it checks each element sequentially.
- **Binary Search:** Efficient for sorted arrays, it reduces the search space by half in each iteration.

2 Problem Statement

Problem 1: Implement a program to search for an element in an unsorted array using Linear Search.
Problem 2: Implement Binary Search for a sorted array using both iterative and recursive approaches.

3 Algorithm

3.1 Linear Search Algorithm

1. Start from the first element of the array.
2. Compare the target element with the current element.
3. If a match is found, return the index.
4. If no match is found by the end of the array, return -1.

3.2 Binary Search Algorithm

Iterative Approach:

1. Initialize two pointers: `low` at the start and `high` at the end of the array.
2. Compute the midpoint: `mid = (low + high) / 2`.
3. Compare the target element with `arr[mid]`:
   - If they match, return `mid`.
   - If the target is smaller, set `high = mid - 1`.
   - If the target is larger, set `low = mid + 1`.
4. Repeat until `low > high`.

Recursive Approach:

1. Compute the midpoint of the current range.
2. If the target matches `arr[mid]`, return `mid`.
3. If the target is smaller, recursively search the left subarray.
4. If the target is larger, recursively search the right subarray.
5. Base case: If the range is invalid (`low > high`), return -1.

4 Code

```
import java.util.Scanner;

public class SearchMethods {

    // Linear Search Function
    public static int linearSearch(int[] arr, int n, int key) {
        for (int i = 0; i < n; i++) {
            if (arr[i] == key) {
                return i; // Element found, return index
            }
        }
        return -1; // Element not found
    }

    // Iterative Binary Search Function
    public static int binarySearchIterative(int[] arr, int n, int key) {
        int low = 0, high = n - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] == key) {
                return mid; // Element found, return index
```

```
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1; // Element not found
}

// Recursive Binary Search Function
public static int binarySearchRecursive(int[] arr, int low, int high, i
    if (low > high) {
        return -1; // Base case: Element not found
    }
    int mid = (low + high) / 2;
    if (arr[mid] == key) {
        return mid; // Element found
    } else if (arr[mid] > key) {
        return binarySearchRecursive(arr, low, mid - 1, key);
    } else {
        return binarySearchRecursive(arr, mid + 1, high, key);
    }
}

// Main Function
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter the number of elements in the array: ");
    int n = sc.nextInt();
    int[] arr = new int[n];

    System.out.println("Enter the elements of the array:");
    for (int i = 0; i < n; i++) {
        arr[i] = sc.nextInt();
    }

    System.out.print("\nEnter the element to search: ");
    int key = sc.nextInt();

    System.out.println("\nChoose the search method:");
    System.out.println("1. Linear Search");
    System.out.println("2. Binary Search (Iterative)");
    System.out.println("3. Binary Search (Recursive)");
    System.out.print("Enter your choice (1-3): ");
    int choice = sc.nextInt();

    if (choice == 1) {
```

```
            // Linear Search
            int result = linearSearch(arr, n, key);
            if (result != -1) {
                System.out.println("Element found at index " + result + "
            } else {
                System.out.println("Element not found using Linear Search.
            }
        } else if (choice == 2) {
            // Binary Search (Iterative)
            System.out.println("Ensure the array is sorted for Binary Searc
            int result = binarySearchIterative(arr, n, key);
            if (result != -1) {
                System.out.println("Element found at index " + result + "
            } else {
                System.out.println("Element not found using Binary Search (
            }
        } else if (choice == 3) {
            // Binary Search (Recursive)
            System.out.println("Ensure the array is sorted for Binary Searc
            int result = binarySearchRecursive(arr, 0, n - 1, key);
            if (result != -1) {
                System.out.println("Element found at index " + result + "
            } else {
                System.out.println("Element not found using Binary Search (
            }
        } else {
            System.out.println("Invalid choice!");
        }

        sc.close();
    }
}
import java.util.Scanner;

public class SearchMethods {

    // Linear Search Function
    public static int linearSearch(int[] arr, int n, int key) {
        for (int i = 0; i < n; i++) {
            if (arr[i] == key) {
                return i; // Element found, return index
            }
        }
        return -1; // Element not found
    }

    // Iterative Binary Search Function
    public static int binarySearchIterative(int[] arr, int n, int key) {
```

```
        int low = 0, high = n - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] == key) {
                return mid; // Element found, return index
            } else if (arr[mid] < key) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return -1; // Element not found
    }

    // Recursive Binary Search Function
    public static int binarySearchRecursive(int[] arr, int low, int high, i
        if (low > high) {
            return -1; // Base case: Element not found
        }
        int mid = (low + high) / 2;
        if (arr[mid] == key) {
            return mid; // Element found
        } else if (arr[mid] > key) {
            return binarySearchRecursive(arr, low, mid - 1, key);
        } else {
            return binarySearchRecursive(arr, mid + 1, high, key);
        }
    }

    // Main Function
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of elements in the array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        System.out.print("\nEnter the element to search: ");
        int key = sc.nextInt();

        System.out.println("\nChoose the search method:");
        System.out.println("1. Linear Search");
        System.out.println("2. Binary Search (Iterative)");
```

```
        System.out.println("3. Binary Search (Recursive)");
        System.out.print("Enter your choice (1-3): ");
        int choice = sc.nextInt();

        if (choice == 1) {
            // Linear Search
            int result = linearSearch(arr, n, key);
            if (result != -1) {
                System.out.println("Element found at index " + result + "
            } else {
                System.out.println("Element not found using Linear Search.
            }
        } else if (choice == 2) {
            // Binary Search (Iterative)
            System.out.println("Ensure the array is sorted for Binary Searc
            int result = binarySearchIterative(arr, n, key);
            if (result != -1) {
                System.out.println("Element found at index " + result + "
            } else {
                System.out.println("Element not found using Binary Search (
            }
        } else if (choice == 3) {
            // Binary Search (Recursive)
            System.out.println("Ensure the array is sorted for Binary Searc
            int result = binarySearchRecursive(arr, 0, n - 1, key);
            if (result != -1) {
                System.out.println("Element found at index " + result + "
            } else {
                System.out.println("Element not found using Binary Search (
            }
        } else {
            System.out.println("Invalid choice!");
        }

        sc.close();
    }
}
```

5 Complexity Analysis

5.1 Linear Search

- Time Complexity: $O(n)$ in the worst case (element not found).
- Space Complexity: $O(1)$.

5.2 Binary Search

- Time Complexity:
  - Iterative: $O(\log n)$.
  - Recursive: $O(\log n)$.
- Space Complexity:
  - Iterative: $O(1)$.
  - Recursive: $O(\log n)$ (due to recursive call stack).

6 Comparison

Criteria	**Linear Search**	**Binary Search**
Input Requirement	Works on unsorted arrays	Requires sorted arrays
Time Complexity	$O(n)$	$O(\log n)$
Space Complexity	$O(1)$	Iterative: $O(1)$, Recursive: $O(\log n)$
Use Case	Small datasets	Large datasets with sorted input

7 Conclusion

Linear Search is simple but inefficient for large datasets. Binary Search, though requiring a sorted array, is significantly faster with a time complexity of $O(\log n)$. Choosing the right algorithm depends on the dataset size and whether sorting is feasible.

8 Output

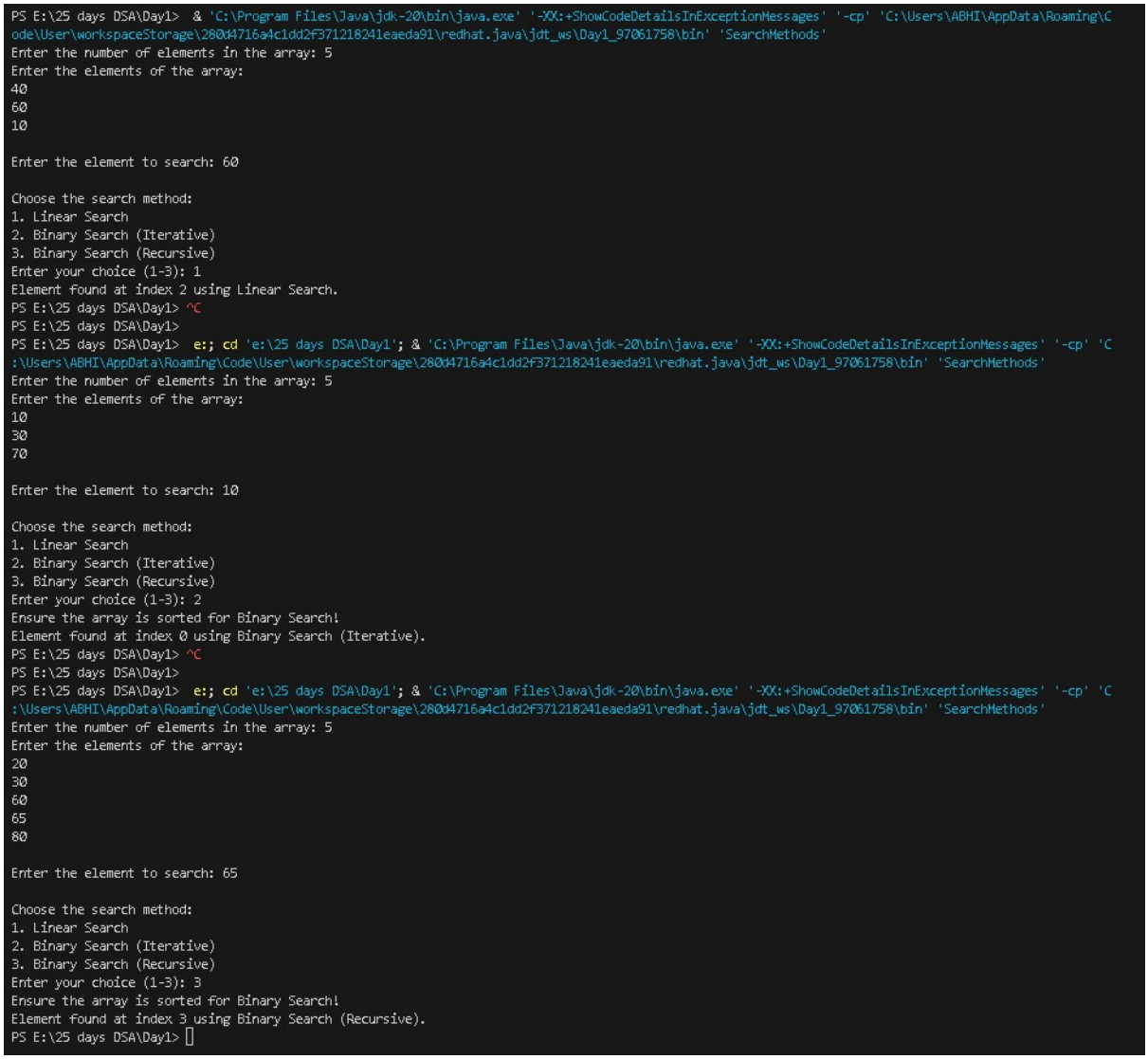

Figure 1: Output