Day 1: Linear Search and Binary Search

Gati Goyal

"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

```c
#include <stdio.h>

// Linear Search Function
int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i; // RETURN INDEX WHERE IT WILL BE FOUND
        }
    }
    return -1;
}

// Iterative Binary Search Function
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; // RETURN INDEX WHERE IT WILL BE FOUND
        } else if (arr[mid] < key) {
            low = mid + 1;
```

```
        } else {
            high = mid - 1;
        }
    }
    return -1;
}

// Recursive Binary Search Function
int binarySearchRecursive(int arr[], int low, int high, int key) {
    if (low > high) {
        return -1; // 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);
    }
}


int main() {
    int choice, n, key;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    printf("\nEnter the element to search: ");
    scanf("%d", &key);

    printf("\nChoose the search method:\n");
    printf("1. Linear Search\n");
    printf("2. Binary Search (Iterative)\n");
    printf("3. Binary Search (Recursive)\n");
    printf("Enter your choice (1-3): ");
    scanf("%d", &choice);

    if (choice == 1) {
        // Linear Search
        int result = linearSearch(arr, n, key);
        if (result != -1) {
```

```
            printf("Element found at index %d using Linear Search.\n", resu
        } else {
            printf("Element not found using Linear Search.\n");
        }
    } else if (choice == 2) {
        // Binary Search (Iterative)
        printf("Ensure the array is sorted for Binary Search!\n");
        int result = binarySearchIterative(arr, n, key);
        if (result != -1) {
            printf("Element found at index %d using Binary Search (Iterativ
        } else {
            printf("Element not found using Binary Search (Iterative).\n");
        }
    } else if (choice == 3) {
        // Binary Search (Recursive)
        printf("Ensure the array is sorted for Binary Search!\n");
        int result = binarySearchRecursive(arr, 0, n - 1, key);
        if (result != -1) {
            printf("Element found at index %d using Binary Search (Recursiv
        } else {
            printf("Element not found using Binary Search (Recursive).\n");
        }
    } else {
        printf("Invalid choice\n");
    }

    return 0;
}
```

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 Screenshots of VS Code Output

To ensure that the implementations of the algorithms are correct, the output from VS Code is provided below. Screenshots of the terminal window with program outputs are attached.

7.1 Output

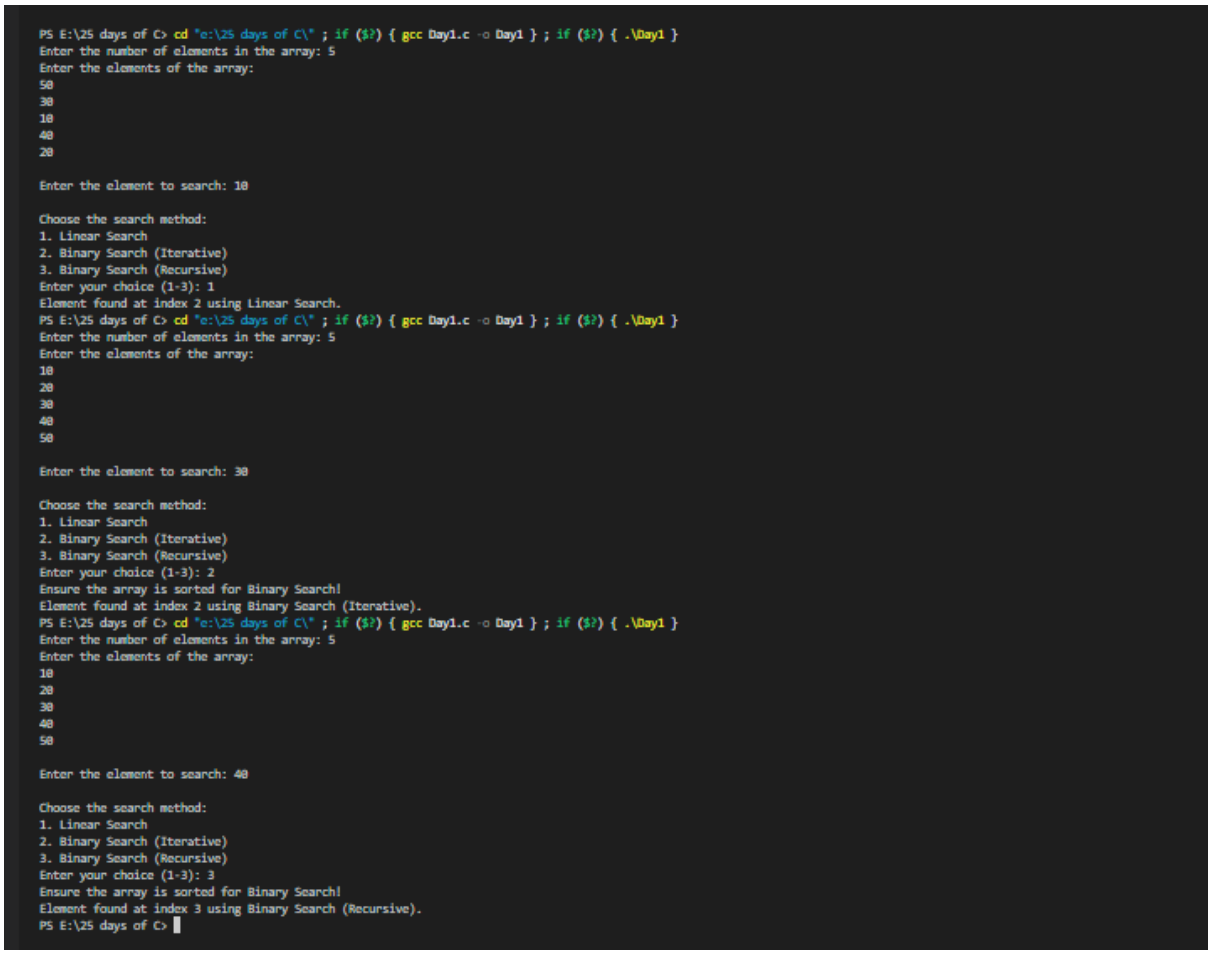

Figure 1: Output n VS Code

8 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.