



## Data Structures and Algorithms

### Department of Computer Science

### University of Engineering and Technology, Lahore



#### Lab Task 1: Implementation of Searching Algorithms (Linear & Binary Search)

Mapping CLOs: CLO1, CLO2

#### Lab Objectives

By completing this lab, you will be able to:

- Understand the concept of **Linear Search** and **Binary Search**.
- Implement searching algorithms in **C++**.
- Compare their efficiency in terms of **time and steps**.
- Practice errors handling unsuccessful searches.

#### Description

Searching is the process of **finding a target element** in a list of elements.

Two basic searching algorithms are:

1. **Linear Search** → checks each element one by one until target is found.
2. **Binary Search** → works on sorted data; repeatedly divides the list into halves to quickly locate the target.

Searching algorithms are fundamental in computer science, forming the basis of database queries, dictionaries, contact searching, and much more.

#### Linear Search

Linear search, also known as sequential search, is a straightforward algorithm used to find a target element within a list or array. It operates by sequentially checking each element of the list, starting from the beginning, until a match is found or the entire list has been traversed.

#### Working on Linear Search

Let's understand the working of linear search with an example of an unsorted array:

Here we have an array of elements

0	1	2	3	4	5	6	7	8
29	87	89	21	23	17	11	10	14

Let **K = 17** be the element to be searched

Now, we will compare K with each of the elements until we find K=17

0	1	2	3	4	5	6	7	8
29	87	89	21	23	17	11	10	14

↑  
 $K \neq 17$

The value of K is not matched with the first element of the array. So, we will switch to the next element and keep repeating the same step until K=17.

0	1	2	3	4	5	6	7	8
29	87	89	21	23	17	11	10	14

↑  
 $K \neq 87$

0	1	2	3	4	5	6	7	8
29	87	89	21	23	17	11	10	14

↑  
 $K \neq 89$

0	1	2	3	4	5	6	7	8
29	87	89	21	23	17	11	10	14

↑  
 $K \neq 21$

0	1	2	3	4	5	6	7	8
29	87	89	21	23	17	11	10	14

↑  
 $K \neq 23$

0	1	2	3	4	5	6	7	8
29	87	89	21	23	17	11	10	14

↑  
 $K \neq 17$

0	1	2	3	4	5	6	7	8
29	87	89	21	23	17	11	10	14

↑  
 $K = 17$

Now, as the element to be searched, i.e., K=17 is found, it will return the index of the matched element.

## Time Complexity

Case	Time Complexity
Best Case	$O(1)$
Average Case	$O(n)$
Worst Case	$O(n)$

## Space Complexity

Space Complexity	$O(1)$
------------------	--------

## Application of Linear Search Algorithm

- The linear search is applicable on both single and multi-dimensional arrays.
- It is less complex, effective, and easy to implement when the array contains a few elements.
- It is efficient when we need to search for a single element in an unordered array.

## Flowcharts

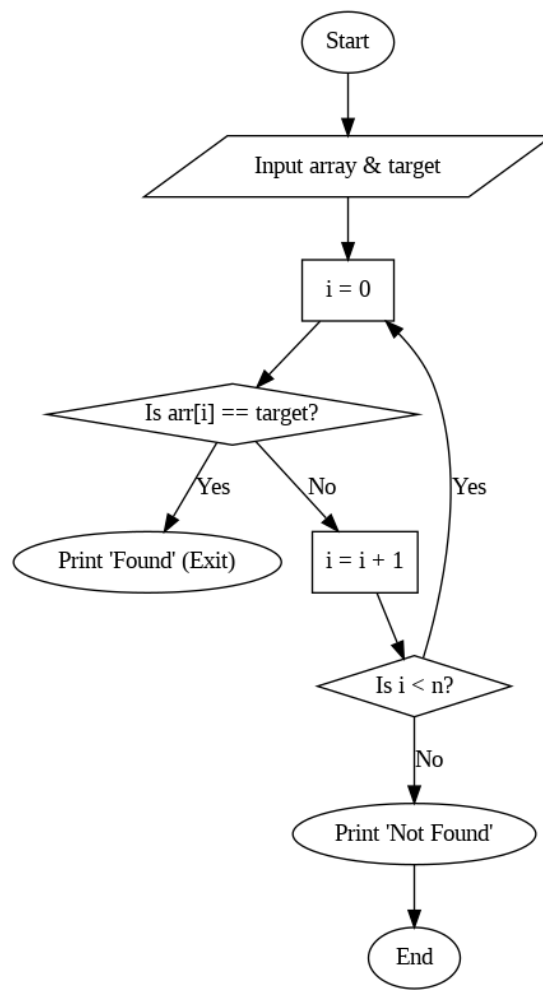


Figure 1 Flowchart Linear Search

## Code

```
int linearSearch(int arr[], int n, int target) {  
    for(int i = 0; i < n; i++) {  
        if(arr[i] == target)  
            return i;  
    }  
    return -1;  
}
```

**Question:** Search for an integer in an unsorted array using Linear Search. Example: {15, 7, 23, 9, 42} → Search 23

```
#include <iostream>  
using namespace std;  
  
int linearSearch(int arr[], int n, int target) {  
    for (int i = 0; i < n; i++) {  
        if (arr[i] == target)  
            return i; // return index if found  
    }  
    return -1; // return -1 if not found  
}  
  
int main() {  
    int arr[] = {15, 7, 23, 9, 42};  
    int n = 5;  
    int target = 23;  
  
    int result = linearSearch(arr, n, target);  
  
    if (result != -1)  
        cout << "Element " << target << " found at index " << result << endl;  
    else  
        cout << "Element not found!" << endl;  
  
    return 0;  
}
```

## Output

```
Element 23 found at index 2
```

## Binary Search

Binary search is an efficient algorithm for finding a target value within a **sorted array** or **list**. It operates on the principle of repeatedly dividing the search interval in half.

The element is always searched in the middle of a portion of an array.

- Binary search can be implemented only on a sorted list of items.
- **If the elements are not sorted already, we need to sort them first.**

### Working of Binary Search

1. The array in which searching is to be performed is:



Let  $x = 4$  be the element to be searched for.

2. Set two pointers low and high at the lowest and the highest positions respectively.



3. Find the middle position mid of the array ie.  $\text{mid} = (\text{low} + \text{high})/2$  and  $\text{arr}[\text{mid}] = 6$ .



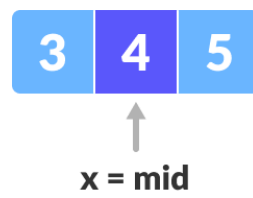
4. If  $x == \text{arr}[\text{mid}]$ , then return mid. Else, compare the element to be searched with  $\text{arr}[\text{mid}]$ .
5. If  $x > \text{arr}[\text{mid}]$ , compare  $x$  with the middle element of the elements on the right side of  $\text{arr}[\text{mid}]$ . This is done by setting low to  $\text{low} = \text{mid} + 1$ .
6. Else, compare  $x$  with the middle element of the elements on the left side of  $\text{arr}[\text{mid}]$ . This is done by setting high to  $\text{high} = \text{mid} - 1$ .



7. Repeat steps 3 to 6 until **low** meets **high**.



8. **x = 4** is found.



## Binary Search Flowchart

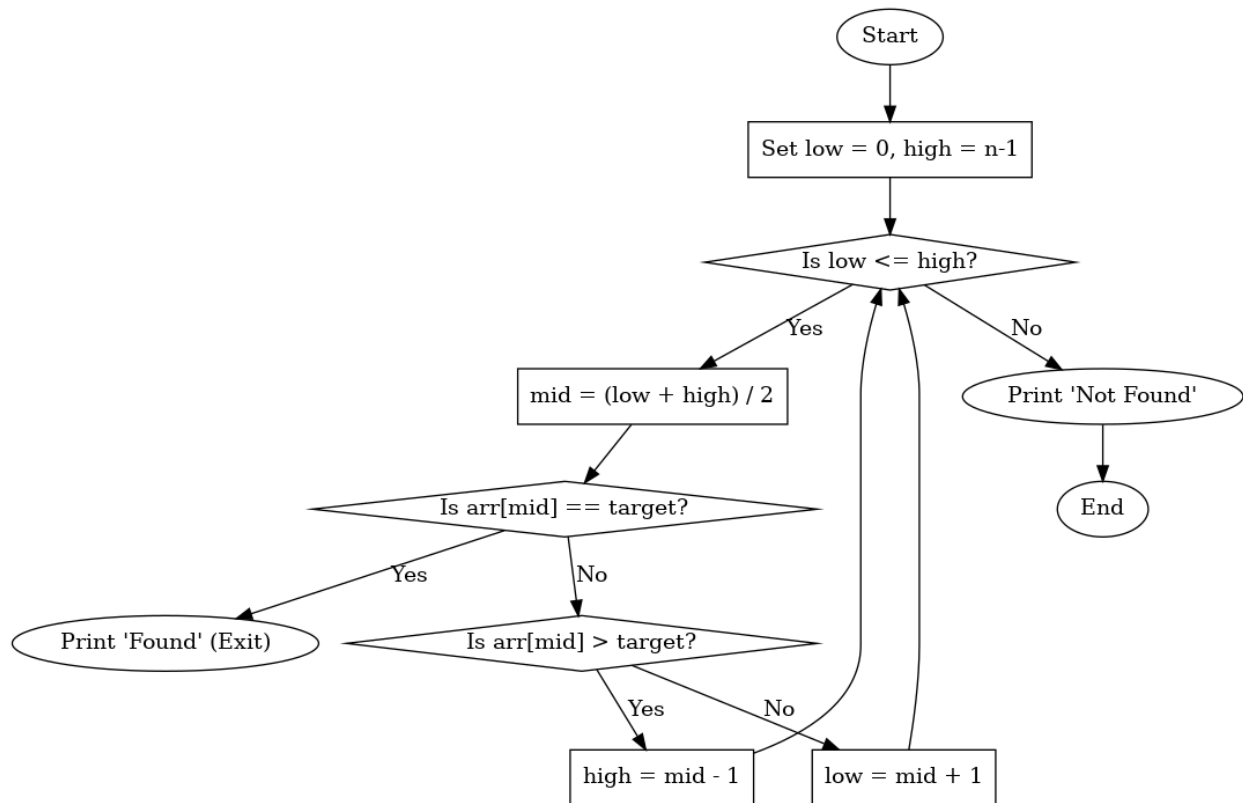


Figure 2 Binary Search Flowchart

## Code

```
int binarySearch(int arr[], int n, int target) {  
    int low = 0, high = n-1;  
    while(low <= high) {  
        int mid = (low + high) / 2;  
        if(arr[mid] == target) return mid;  
        else if(arr[mid] > target) high = mid - 1;  
        else low = mid + 1;  
    }  
    return -1;  
}
```

**Question:** Search for a string in a sorted array using Binary Search. Example: { "Ali", "Ayesha", "Hassan", "Omar", "Sara" } → Search "Omar"

```

#include <iostream>
#include <string>
using namespace std;
int binarySearchString(string arr[], int n, string target) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == target)
            return mid; // found
        else if (arr[mid] > target) // compare lexicographically
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1; // not found
}
int main() {
    string names[] = {"Ali", "Ayesha", "Hassan", "Omar", "Sara"};
    int n = 5;
    string target = "Omar";
    int result = binarySearchString(names, n, target);
    if (result != -1)
        cout << "Name \" " << target << "\" found at index " << result << endl;
    else
        cout << "Name not found!" << endl;
    return 0;
}

```

## Output

▲ Name "Omar" found at index 3

## Real-Life Examples

- **Linear Search:** Searching for your friend's name in an **unsorted attendance sheet**.
- **Binary Search:** Looking for a word in a **dictionary** or a contact in a **phone book**.

## Implementation Details

1. Define functions for **linearSearch()** and **binarySearch()**.
2. Input: Array of integers + target value.
3. Output: Index of element (if found) or -1 (if not found).
4. Handle unsuccessful searches with a proper message.

## Functions to Implement

- `int linearSearch(int arr[], int n, int target)`
- `int binarySearch(int arr[], int n, int target)`
- `void displaySteps()` → to show number of comparisons.

## Practice Tasks

1. Count how many times an integer occurs in an array.  
Example: {4, 7, 4, 2, 9, 4} → Search 4 → occurs 3 times
2. Search for a character in an array of characters using Linear Search.  
Example: {'a', 'e', 'i', 'o', 'u'} → Search 'o'
3. Search for a string (name) in an unsorted array of names using Linear Search.  
Example: { "Ali", "Sara", "Omar", "Ayesha" } → Search "Sara"
4. Find all indices of an element in an array (multiple occurrences).  
Students must return an array of indices where the element is found.  
Example: {5, 2, 7, 2, 9, 2} → Search 2 → Indices: {1, 3, 5}
5. Search in a 2D array (row-wise or column-wise).  
Example: Matrix = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ] Search 5 → Found at (row=1, col=1)