Course Name:	**Data Structures Laboratory**	**Semester:**	**III**
Date of Performance:		**Batch No:**	**A1**
Faculty Name:	**Prof. Sushma Kadge**	**Roll No:**	**20**
Faculty Sign & Date:		**Grade/Marks:**	**/25**

Experiment No: 8
Title: Searching Techniques

Aim and Objective of the Experiment:

Write a C/C++ program for linear search and binary search.

COs to be achieved:

CO4: Demonstrate sorting and searching methods

Books/Journals/Websites referred:

1. GeeksForGeeks

Tools required:

DEV C/C++ compiler/ Code blocks C compiler

Theory:

To search an element in a given array, there are two popular algorithms available:
1. **Linear Search:** Linear search is a very basic and simple search algorithm. In Linear search, we search an element or value in a given array by traversing the array from the starting, till the desired element or value is found.
2. **Binary Search:** Binary Search is used with sorted array or list. In binary search, we follow the following steps:
(a) We start by comparing the element to be searched with the element in the middle of the list/array.
(b) If we get a match, we return the index of the middle element.
(c) If we do not get a match, we check whether the element to be searched is less or greater than in value than the middle element.

(d) If the element/number to be searched is greater in value than the middle number, then we pick the elements on the right side of the middle element(as the list/array is sorted, hence on the right, we will have all the numbers greater than the middle number), and start again from the step 1.
(e) If the element/number to be searched is lesser in value than the middle number, then we pick the elements on the left side of the middle element, and start again from the step 1.
Binary Search is useful when there are large numbers of elements in an array and they are sorted.

So a necessary condition for Binary search to work is that the list/array should be sorted.

Implementation details:

1. Enlist all the Steps followed and various options explored
2. Explain your program logic and methods used.
3. Explain the Importance of the approach followed by you

Details:

Steps Followed:

1. Accept array size and elements from the user
2. Accept element to search
3. Implement Linear Search
4. Sort the array (if needed)
5. Implement Binary Search
6. Display results of both methods

Program Logic & Methods:

- Used **for loop** for Linear Search
- Used **while loop** with mid-index calculation for Binary Search
- Searching continues until element is found or search space finishes

Importance of the Approach:

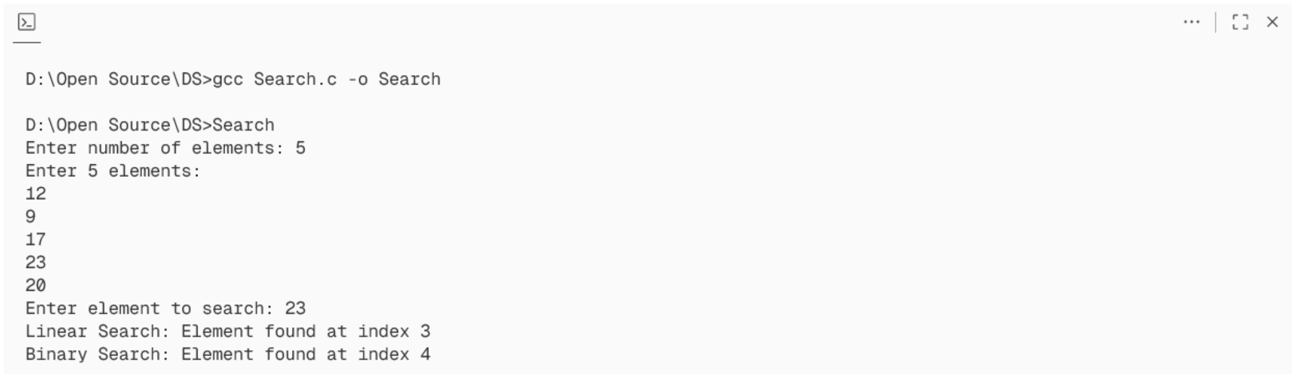
- Shows difference between sorted vs unsorted searching
- Highlights efficiency of binary search

C/C++ Code implemented:

```
#include <stdio.h>

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

void sortArray(int arr[], int n)
{
    int i, j, temp;
    for(i = 0; i < n - 1; i++)
    {
        for(j = 0; j < n - i - 1; j++)
        {
            if(arr[j] > arr[j + 1])
            {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

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

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

int main()
{
    int n, key, i, pos, sortedIndex;
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    printf("Enter element to search: ");
    scanf("%d", &key);

    pos = linearSearch(arr, n, key);
    if(pos != -1)
        printf("Linear Search: Element found at index %d\n", pos);
    else
        printf("Linear Search: Element not found\n");

    sortArray(copy, n);
    sortedIndex = binarySearchIndex(copy, n, key);

    if(sortedIndex != -1)
    {
        pos = findOriginalIndex(arr, n, key);
        if(pos != -1)
            printf("Binary Search: Element found at index %d\n", pos);
        else
            printf("Binary Search: Element not found\n");
    }
    else
    {
        printf("Binary Search: Element not found\n");
    }
    return 0;
}
```

Output:



```
D:\Open Source\DS>gcc Search.c -o Search

D:\Open Source\DS>Search
Enter number of elements: 5
Enter 5 elements:
12
9
17
23
20
Enter element to search: 23
Linear Search: Element found at index 3
Binary Search: Element found at index 4
```

Conclusion:

This experiment was successfully conducted to implement and understand two fundamental searching algorithms: linear and binary search. Through the implementation, it was observed that linear search is a straightforward method that sequentially checks each element in an array, which is simple but can be inefficient for large datasets. In contrast, binary search requires a pre-sorted array and operates by repeatedly dividing the search interval in half, making it significantly more efficient, especially for arrays with a large number of elements. This lab effectively demonstrated the trade-off between the simplicity of linear search and the efficiency of binary search, fulfilling the objective of demonstrating different searching methods.

Post Lab:

1. Compare various searching techniques.
Ans:

Parameter	Linear Search	Binary Search
Prerequisite	No specific order is required; it can be performed on unsorted arrays.	A necessary condition is that the array must be sorted.
Method	It involves traversing the array sequentially from the start and comparing each element with the key value until it is found.	It compares the key element with the middle element of the array. If they don't match, it eliminates one half of the array and continues the search in the other half.
Efficiency	Less efficient for large arrays, as it may need to scan the entire array in the worst case.	More efficient and useful when there are a large number of elements, as it halves the search space with each comparison.
Simplicity	It is a very basic and simple search algorithm to implement.	It is more complex than linear search as it requires a sorted list and logic to manage the search space (low, high, mid indices).

Signature of faculty in-charge with Date: