Aim: Searching Techniques.

Objectives: The main objective of this assignment is to understand and implement various searching techniques such as linear search and binary search in C++. The goal is to understand how different algorithms can be used to find an element in a given data set.

Tools Used: Online C++ Compiler.

Concept:

Linear search is a search algorithm that checks each element in an array sequentially until the desired element is found or till the end of an array.

Process:

1.) Start from the first element of the array [index 0].
2.) Compare the target element (key) with each element of the array.
3.) If a match is found, return the index of the element.
4.) If the target element is not found by the end of the array, return a "Key not found" result.

Example:

Array: [2,8,1,9,5]
Key: 8

1.) Compare the first element (2) with 8 → No match.

2.) Compare the second element (8) with 8 → Match found.

The algorithm stops, and the index 2 is returned.

Binary search is a more efficient search algorithm compared to linear search. It works by repeatedly dividing a sorted array in half and comparing the middle element with the target. If the middle element is not the target, the algorithm decides whether to search in the left or right half based on the target's value. Here the given array must be sorted in ascending order.

Process:

1.) Start with the entire sorted array.
2.) Find the middle element of the array.
3.) Compare the target element (key) with the middle element.
    - If a match is found, return the index of the middle element.
    - If the key is less than the middle element, repeat the process on the left half of the array.
    - If the key is greater than the middle element, repeat the process on the right half of the array.
4.) If the target is not found, return a "Key not found" result.

Example:

Array: [1, 2, 5, 8, 9]
Key: 8

1. Start with the entire array: [1, 2, 5, 8, 9]
2. Calculate the middle index: (0 + 4) / 2 = 2, so the middle element is 5.
   - Compare the middle element (5) with 8 → No match.
   - Since 8 > 5, search in the right half of the array: [8, 9]
3. Now, the new sub-array is [8, 9].
   - Calculate the new middle index: (3 + 4) / 2 = 3, so the middle element is 8.
   - Compare the middle element (8) with 8 → Match found.

The algorithm stops, and the index 3 is returned.

Problem Statement:

1.) Implement Linear Search.
2.) Implement Binary Search.

Solution:

1.) Linear Search.

```
#include<iostream>

#define max 100

using namespace std;

int n,arr[max],key,choice;


class LinearSearch

{

public:

        void input()

        {

                cout<<"Enter how many elements you want to store: ";

                cin>>n;

                for (int i=0; i<n; i++)

                {
```

```
                cout<<"Enter the data for "<<i <<" Index: ";
                cin>>arr[i];
        }
        cout << "Array ";
        for(int i = 0; i < n; i++) {
                cout << arr[i] << " ";
        }
        cout << endl;
}

void menu() {
        cout<<"Enter choice\n";
        cout<<"1.) Press 1 to do Linear Search\n";
        cout<<"2.) Press 2 to Exit\n";
        cin>>choice;

}
void search()
{
        cout<<"Enter the element you want to search: ";
        cin>>key;
        int pos=-1;
        for (int i=0; i<=n; i++)
        {
                if(key==arr[i])
                {
                        pos=i;
                        break;
                }
        }
        if(pos==-1)
```

```
                {
                        cout<<"Element "<< key <<" is not found\n";
                        cout<<"Elemnt will be inserted at the end\n";

                        arr[n] = key;
                        n++;

                        cout << "New Array ";
                        for(int i = 0; i < n; i++) {
                                cout << arr[i] << " ";
                        }
                        cout << endl;
                }
                else
                {
                        cout<<"Element "<< key <<" is found at Index "<< pos << endl ;
                }
        }

};
int main()
{
        LinearSearch ls;
        ls.input();
        do {
                ls.menu();
                if (choice == 1) {
                        ls.search();
                }
        } while (choice != 2);
        return 0;
```

}

```
Enter how many elements you want to store: 5
Enter the data for 0 Index: 2
Enter the data for 1 Index: 8
Enter the data for 2 Index: 1
Enter the data for 3 Index: 9
Enter the data for 4 Index: 5
Array 2 8 1 9 5
Enter choice
1.) Press 1 to do Linear Search
2.) Press 2 to Exit
1
Enter the element you want to search: 8
Element 8 is found at Index 1
Enter choice
1.) Press 1 to do Linear Search
2.) Press 2 to Exit
1
Enter the element you want to search: 7
Element 7 is not found
Elemnt will be inserted at the end
New Array 2 8 1 9 5 7
Enter choice
1.) Press 1 to do Linear Search
2.) Press 2 to Exit
2
```

2.) Binary Search.

```
#include<iostream>
#define max 100
using namespace std;
int n,arr[max],low,high,mid,pos=-1,key,choice;

class BinarySearch
{
public:
    void input()
    {
        cout<<"Enter how many elements you want to store: ";
        cin>>n;
        for (int i=0; i<n; i++)
        {
            cout<<"Enter the data for "<<i <<" Index: ";
            cin>>arr[i];
        }
        cout << "Array: ";
        for(int i = 0; i < n; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
    void sort()
    {
        for(int i=0; i<n; i++)
        {
            for(int j=0; j<n-i-1; j++)
            {
```

```
                        if(arr[j]>arr[j+1])
                        {
                                int temp;
                                temp=arr[j];
                                arr[j]=arr[j+1];
                                arr[j+1]=temp;
                        }
                }
        }
        cout << "Sorted Array: ";
        for(int i = 0; i < n; i++) {
                cout << arr[i] << " ";
        }
        cout << endl;
}

void search()
{
        cout<<"Enter key to be searched: ";
        cin>>key;
        low=0,high=n-1;
        while(high>=low)
        {
                mid=(low+high)/2;
                if(key==arr[mid])
                {
                        pos=mid;
                        break;
                }
                else if(key<arr[mid])
                {
```

```
                        high=mid-1;
                }
                else
                {
                        low=mid+1;
                }
        }
        if(pos==-1)
        {
                cout<<"Key not found\n";
        }
        else
        {
                cout<<"Element "<< key <<" is found at Index "<< pos << endl ;
        }
    }

    void menu() {
        cout<<"Enter choice\n";
        cout<<"1.) Press 1 to do Linear Search\n";
        cout<<"2.) Press 2 to Exit\n";
        cin>>choice;

    }
};
int main()
{
    BinarySearch bs;
    bs.input();
    bs.sort();
    do {
```

```
            bs.menu();
            if (choice == 1) {
                bs.search();
            }
        } while (choice != 2);

        return 0;
    }
```

```
Enter how many elements you want to store: 5
Enter the data for 0 Index: 2
Enter the data for 1 Index: 8
Enter the data for 2 Index: 1
Enter the data for 3 Index: 9
Enter the data for 4 Index: 5
Array: 2 8 1 9 5
Sorted Array: 1 2 5 8 9
Enter choice
1.) Press 1 to do Linear Search
2.) Press 2 to Exit
8
Enter choice
1.) Press 1 to do Linear Search
2.) Press 2 to Exit
1
Enter key to be searched: 7
Key not found
Enter choice
1.) Press 1 to do Linear Search
2.) Press 2 to Exit
2
```

Observation: In this practical session I learned about different searching techniques such as Linear search and Binary Search. Linear search checks each element in an array sequentially, making it simple and easy to implement. It works on both sorted and unsorted data but can be slow for large data sets. whereas, binary search is more efficient, as it repeatedly divides a sorted array in half. However, it requires the data to be sorted beforehand. Linear search is best for small datasets or unsorted arrays, while binary search is ideal for larger or sorted arrays.