DAA LAB-2

Amit Rajkumar Ingle

S.Y.B.Tech Computer

231070020

AIM: To write algorithm of linear and binary search, implementing algorithm via program including google c++ coding style.

THEORY:

1)Linear search :

Linear search, or sequential search, is a simple algorithm that checks each element in a list one by one to find a target value. It doesn't require the list to be sorted and is easy to implement. However, with a time complexity of O(n), it can be inefficient for large lists, as it may need to examine every element in the worst case. It's useful for small or unsorted lists where simplicity is preferred over speed.

2)Binary search:

Binary search is an efficient algorithm for finding an element in a sorted list. It works by repeatedly dividing the search interval in half, comparing the target value to the middle element. If the target is less than the middle element, the search continues in the lower half; if greater, in the upper half. With a time complexity of O(log n), binary search is much faster than linear search for large datasets. However, it requires the list to be sorted beforehand, which can add extra overhead if sorting is needed.

ALGORITHM: 1)Linear search :

algorithm : Linearsearch (arr[], n, k)

// Input : Array arr[] of size n &
element k to Search in Array.

// output : if k is found then Index
else -1

i ← 0

while i<n do // Loop

if arr[i] == k // checking wheather
then return i ; arr[i] matches with k

i ← i+1 // Increment

if all n elements not matches with k
then return -1

2)Binary search:

```
algorithm : Binarysearch (arr[], n, k)

// Input : Array arr[] with size n,
           element k to search in array

// output : Index of k if found
            else -1 //not found

s ← 0
e ← n-1
m ← (s+e)/2 // mid element

while s ≤ e do // loop

if arr[m]==k then //mid element matches
return m;               with k so
                        return index m

else if arr[m] < k // k is greater than
then s=m+1;               mid element

else e=m-1; // k is smaller than
                 mid element

m ← (s+e)/2 // updation of mid
               as per new values of
               s&e

return -1; // element k not foun
```

TEST CASES:

1)Linear search:

Testcases :

	arr[]	K	Expected output
Testcase 1 :	[1, 2, 3, 4, 5]	4	3
Testcase 2 :	[-5, -3, -1, -2, -7]	-5	0
Testcase 3 :	[1, 2, -1, -2]	-3	-1
Testcase 4 :	[10, 15, 20, 25]	15	1
Testcase 5 :	[6, 7, -1, -2, 3]	2	-1

2)Binary search:

Testcases :

	arr[]	K	Expected output
Testcase 1 :	[1, 2, 3, 4]	4	3
Testcase 2 :	[-5, -4, -3, -2]	-5	0
Testcase 3 :	[10, 20, 30, 40]	50	-1
Testcase 4 :	[-1, -3, -5, -8]	-3	1
Testcase 5 :	[3, 6, 9, 12, 15]	16	-1

TIME COMPLEXITY :

1)Linear search:

In linear search, we start at the beggining of list & check each element one by one untill
① we find target element or
② we reach the end of list

• Best case complexity:

① Target element is at first position
② algorithm checks only one element
③ Time complexity : $O(1)$

• Worst case complexity :

① Target element at → last position / → not present
② algorithm checks for all n elements
③ Time complexity : $O(n)$

• Avg case complexity :

① Target at any position.
② on average, algo checks for $n/2$ elements
③ In Big-O notation, we neglect constant factor $1/2$ so Time complexity still $O(n)$

conclusion : $O(n)$

In worst case scenerio, time to complete search linearly grow with size of array.

2)Binary search:

- Best case complexity:

① target element is middle element of array in first comparison.

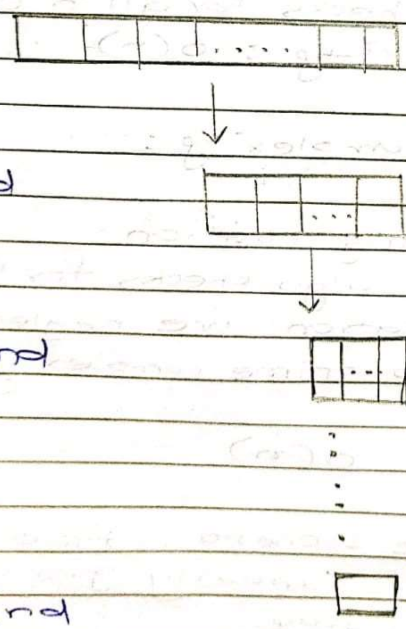
② Time complexity : $O(1)$

- worst case complexity:

① algorithm repeatedly halves search space untill it narrows down to one element.

② If array has n element, Time complexity is no. of times we need to halve array untill we get down to one eleme

③ complexity : $O(\log_2 n)$



	size
1^{st} round	$n \rightarrow n/2^0$
2^{nd} round	$n/2 \rightarrow n/2^1$
.	
.	
3^{rd} Round	$n/4 \rightarrow n/2^2$
.	.
.	.
k^{th} Round	$1 \rightarrow n/2^{k-1}$

$$\therefore \frac{n}{2^{k-1}} = 1$$

$$\therefore n = 2^{k-1}$$

$$n = 2^{k} \quad \ldots \text{ignoring } -1$$

$$\therefore \log_2 n = k \quad \ldots \left\{ \begin{array}{l} \because x = n^y \\ \because y = \log_n x \end{array} \right\}$$

$$\therefore \boxed{k = \log_2 n}$$

- Avg case complexity:

Like worst case, Binary search will have logn complexity since it depend How many times halves array for searching element.

Conclusion:

Time complexity : $O(\log_2 n)$

Note: Binary search is more efficient than linear search because it reduces the search space by half with each iteration, leading to a time complexity of (O(log n)) compared to the linear search's (O(n)).

PROGRAM :

1)Linear search:

```
#include <iostream>

using namespace std;

// Performs a linear search on an array.
// Returns the index of the target if found, or -1 if not found.
int LinearSearch(const int arr[], int n, int target) {
    for (int i = 0; i < n; ++i) {
        if (arr[i] == target) {
            return i;
        }
    }
    return -1;
}
```

```
int main() {
    int n;
    int k;

    cout << "Enter size of array: ";
    cin >> n;

    // Allocate array with size n.
    int* arr = new int[n];

    for (int i = 0; i < n; ++i) {
        cout << "Enter element at index " << i << ": ";
        cin >> arr[i];
    }

        cout<<"array is ";
     for (int i = 0; i < n; ++i) {
       cout<<arr[i]<<" ";
    }

    cout<<endl;

    cout << "Enter element k to search in array: ";
    cin >> k;

    int ans = LinearSearch(arr, n, k);

        cout << "Index of element by linear search is " << ans << endl;

    // Free allocated memory.
    delete[] arr;

    return 0;
}
```

2)Binary search:

```cpp
#include <iostream>

using namespace std;

// Performs a binary search on a sorted array.
// Returns the index of the target if found, or -1 if not found.
int BinarySearch(const int arr[], int n, int target) {
    int s = 0;
    int e = n - 1;
    int m = s + (e - s) / 2;

    while (s <= e) {
        if (arr[m] == target) {
            return m;
        } else if (arr[m] < target) {
            s = m + 1;
        } else {
            e = m - 1;
        }
        m = s + (e - s) / 2;
    }

    return -1;
}
```

```cpp
int main() {
    int n;
    int target;

    cout << "Enter size of array: ";
    cin >> n;

    // Allocate array with size n.
    int* arr = new int[n];

    for (int i = 0; i < n; ++i) {
        cout << "Enter element at index " << i << ": ";
        cin >> arr[i];
    }

        cout<<"array is ";
     for (int i = 0; i < n; ++i) {
       cout<<arr[i]<<" ";
    }
    cout<<endl;

    cout << "Enter element to search in array: ";
    cin >> target;

    int ans = BinarySearch(arr, n, target);

        cout << "Index of element by binary search is " << ans << endl;

    // Free allocated memory.
    delete[] arr;

    return 0;
}
```

SCREENSHOT:

1)Linear search:

Testcase1:

```
Enter size of array: 5
Enter element at index 0: 1
Enter element at index 1: 2
Enter element at index 2: 3
Enter element at index 3: 4
Enter element at index 4: 5
array is 1 2 3 4 5
Enter element k to search in array: 4
Index of element by linear search is 3
```

Testcase2:

```
Enter size of array: 5
Enter element at index 0: -5
Enter element at index 1: -3
Enter element at index 2: -1
Enter element at index 3: -2
Enter element at index 4: -7
array is -5 -3 -1 -2 -7
Enter element k to search in array: -5
Index of element by linear search is 0
```

Testcase3:

```
Enter size of array: 4
Enter element at index 0: 1
Enter element at index 1: 2
Enter element at index 2: -1
Enter element at index 3: -2
array is 1 2 -1 -2
Enter element k to search in array: -3
Index of element by linear search is -1
```

Testcase4:

```
Enter size of array: 4
Enter element at index 0: 10
Enter element at index 1: 15
Enter element at index 2: 20
Enter element at index 3: 25
array is 10 15 20 25
Enter element k to search in array: 15
Index of element by linear search is 1
```

Testcase5:

```
Enter size of array: 5
Enter element at index 0: 6
Enter element at index 1: 7
Enter element at index 2: -1
Enter element at index 3: -2
Enter element at index 4: 3
array is 6 7 -1 -2 3
Enter element k to search in array: 2
Index of element by linear search is -1
```

2)Binary search:

Testcase1:

```
Enter size of array: 4
Enter element at index 0: 1
Enter element at index 1: 2
Enter element at index 2: 3
Enter element at index 3: 4
array is 1 2 3 4
Enter element to search in array: 4
Index of element by binary search is 3
```

Testcase2:

```
Enter size of array: 4
Enter element at index 0: -5
Enter element at index 1: -4
Enter element at index 2: -3
Enter element at index 3: -2
array is -5 -4 -3 -2
Enter element to search in array: -5
Index of element by binary search is 0
```

Testcase3:

```
Enter size of array: 4
Enter element at index 0: 10
Enter element at index 1: 20
Enter element at index 2: 30
Enter element at index 3: 40
array is 10 20 30 40
Enter element to search in array: 50
Index of element by binary search is -1
```

Testcase4:

```
Enter size of array: 4
Enter element at index 0: -1
Enter element at index 1: -3
Enter element at index 2: -5
Enter element at index 3: -8
array is -1 -3 -5 -8
Enter element to search in array: -3
Index of element by binary search is 1
```

Testcase5:

```
Enter size of array: 5
Enter element at index 0: 3
Enter element at index 1: 6
Enter element at index 2: 9
Enter element at index 3: 12
Enter element at index 4: 15
array is 3 6 9 12 15
Enter element to search in array: 16
Index of element by binary search is -1
```

CONCLUSION:

In this way ,we understood algorithm of linear and binary search and understood how to write code in google c++ style and we also test 5 testcases each for linear and binary search and program output matches with expected output.

We also calculate time complexity of linear and binary search and concluded that binary search is more efficient than linear search.

END