Linear Vs Binary Search + Code in C Language (With Notes)

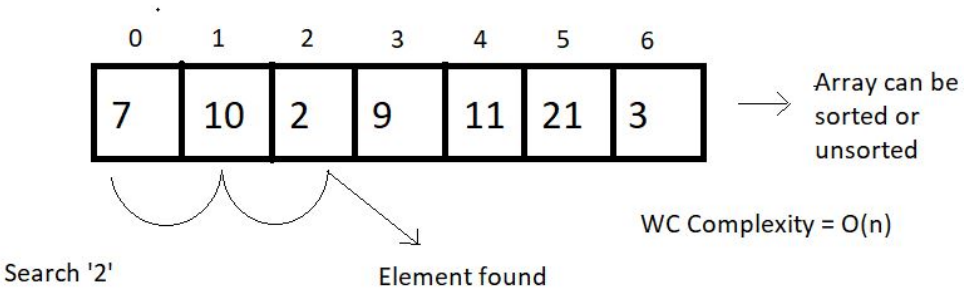
codewithharry.com/videos/data-structures-and-algorithms-in-hindi-12

We have already covered the first three operators in an array, namely traversal, insertion, and deletion. Today, we will learn about the search operations in an array.

You must already be familiar with these two methods we have for searching in an array, linear and binary search. We had used them quite a bit in our previous tutorials. We had analyzed them and got the result that for a sorted array, the fastest method to search is the binary one. Today, we'll learn how to code them and practically use them to search.

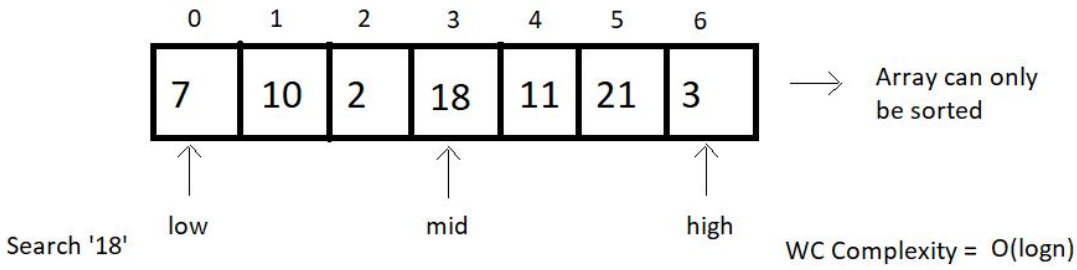
Linear Search:

This search method searches for an element by visiting all the elements sequentially until the element is found or the array finishes. It follows the array traversal method.



Binary Search:

This search method searches for an element by breaking the search space into half each time it finds the wrong element. This method is limited to a sorted array. The search continues towards either side of the mid, based on whether the element to be searched is lesser or greater than the mid element of the current search space.



From the above illustrations, we can draw a comparison between both the search methods based on their choice of arrays, operations, and worst-case complexities.

	Linear Search	**Binary Search**
1.	Works on both sorted and unsorted arrays	Works only on sorted arrays
2.	Equality operations	Inequality operations
3.	O(n) WC Complexity	O(log n) WC Complexity

Table 1: Linear Search VS Binary Search

Let us now move on to the coding part of these methods. I have attached the snippet below. Refer to it while understanding.

Understanding the code snippet 1:

Linear Search:

1. We'll start with coding the linear search. Create an integer function *linearSearch*. This function will receive the array, its size, and the element to be searched as its parameters.

2. Run a *for* loop from its 0 to the last index, checking the *if* condition at every index whether the element at that index equals the search element. If *yes,* return the index, else continue the search.

3. If the element could not be found until the last, return -1.

Binary Search:

1. Create a function named *binarySearch* and pass the same three parameters as we did in linear search. Here, we will maintain three integer variables *low*, *mid,* and *high*. *Low* stores are the beginning of the search space, and *high* stores the end. *Mid* stores the

middle element of our search space, which is *mid = (low+high)/2*.

2. Check whether the *mid* element equals the search element. If yes, return mid, else if the mid element is greater than the search element, then the search element must lie on the left side of the current space and *high* becomes *mid-1*, else if the mid element is less than the search element, then we'll shift to the right side, and *low* becomes *mid+1*.

3. This way, we reduce our search space into half every time we repeat step 2. Now our new *mid* becomes (*low+high)/2,* and we repeat step 2. And keep repeating until either we find the search element or the *low* becomes greater than the *high*.

```
#include<stdio.h>

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

int binarySearch(int arr[], int size, int element){
    int low, mid, high;
    low = 0;
    high = size-1;
    // Keep searching until low <= high
    while(low<=high){
        mid = (low + high)/2;
        if(arr[mid] == element){
            return mid;
        }
        if(arr[mid]<element){
            low = mid+1;
        }
        else{
            high = mid -1;
        }
    }
    return -1;

}

int main(){
    // Unsorted array for linear search
    // int arr[] = {1,3,5,56,4,3,23,5,4,54634,56,34};
    // int size = sizeof(arr)/sizeof(int);

    // Sorted array for binary search
    int arr[] = {1,3,5,56,64,73,123,225,444};
    int size = sizeof(arr)/sizeof(int);
    int element = 444;
    int searchIndex = binarySearch(arr, size, element);
    printf("The element %d was found at index %d \n", element, searchIndex);
    return 0;
}
```

Code Snippet 1: Linear search and Binary search codes.

Let’s check if it works. Refer to the output below:

```
The element 444 was found at index 8
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

So from the above output, you can conclude that our code works all fine. So, we are done implementing both these search methods. Binary Search holds great importance in the world of programming. We’ll come across several algorithms following binary search. If

you couldn’t grasp it all at once, repeat.

Thank you for being with me throughout. I hope you enjoyed the tutorial. If you appreciate my work, please let your friends know about this course too. Don’t forget to download the notes from the link below. If you haven’t checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial, where we’ll get introduced to a new topic, **Linked Lists in Data Structures.** Till then, keep learning.

Download the notes here