

**CH 06: Searching Algorithms**

HND COMPUTING (SW)

BCAS Campus, Kalmunai

By: Eng. A.L. Jubailah Begum

DATA STRUCTURES & ALGORITHMS

Searching Algorithms

Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored. Based on the type of search operation, these algorithms are generally classified into two categories:

1. **Sequential Search**: In this, the list or array is traversed sequentially and every element is checked.

For example: [Linear Search](https://www.geeksforgeeks.org/linear-search/).

1. **Interval Search**: These algorithms are specifically designed for searching in sorted data-structures. These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half.

For Example: [Binary Search](https://www.geeksforgeeks.org/binary-search/).

**Linear Search to find the element “20” in a given list of numbers**



**Binary Search to find the element “23” in a given list of numbers**



**Linear Search**

**Problem:** Given an array arr[] of n elements, write a function to search a given element x in arr[].

**Examples:**

1. **Input:** arr[] = {10, 20, 80, 30, 60, 50,

110, 100, 130, 170}

x = 110;

**Output:** 6

Element x is present at index 6

1. **Input:** arr[] = {10, 20, 80, 30, 60, 50,

110, 100, 130, 170}

x = 175;

**Output:** -1

Element x is not present in arr[].

A simple approach is to do a **linear search**, i.e

* Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
* If x matches with an element, return the index.
* If x doesn’t match with any of elements, return -1.



Let us look at some C# implementations of linear search examples.

Example 1:

// C# code to linearly search x in arr[]. If x

// is present then return its location, otherwise

// return -1

using System;

class LinearSearch {

    public static int search(int[] arr, int x)

    {

        int n = arr.Length;

        for (int i = 0; i < n; i++)

        {

            if (arr[i] == x)

                return i;

        }

        return -1;

    }

    // Driver code

    public static void Main()

    {

        int[] arr = { 2, 3, 4, 10, 40 };

        int x = 10;

        // Function call

        int result = search(arr, x);

        if (result == -1)

            Console.WriteLine(

                "Element is not present in array");

        else

            Console.WriteLine("Element is present at index "

                              + result);

Console.ReadKey();

    }

}

**Output:**

Element is present at index 3

The **time complexity** of the above algorithm is O(n).

Linear search is rarely used practically because other search algorithms such as the binary search algorithm and hash tables allow significantly faster-searching comparison to Linear search.

**Improve Linear Search Worst-Case Complexity**

1. if element Found at last,  O(n) to O(1)
2. if element Not found, O(n) to O(n/2)

Example 2:

// C# program for linear search

using System;

class GFG

{

    public static void search(int []arr,

                              int search\_Element)

    {

        int left = 0;

        int length = arr.Length;

        int right = length - 1;

        int position = -1;

        // run loop from 0 to right

        for (left = 0; left <= right;)

        {

            // if search\_element is found with left varaible

            if (arr[left] == search\_Element)

            {

                position = left;

                Console.WriteLine(

                    "Element found in Array at "

                    + (position + 1) + " Position with "

                    + (left + 1) + " Attempt");

                break;

            }

            // if search\_element is found with right varaible

            if (arr[right] == search\_Element)

            {

                position = right;

                Console.WriteLine(

                    "Element found in Array at "

                    + (position + 1) + " Position with "

                    + (length - right) + " Attempt");

                break;

            }

            left++;

            right--;

        }

        // if element not found

        if (position == -1)

            Console.WriteLine("Not found in Array with "

                               + left + " Attempt");

    }

    // Driver code

    public static void Main(String[] args)

    {

        int []arr = { 1, 2, 3, 4, 5 };

        int search\_element = 5;

        // Function call

        search(arr,search\_element);

    }

}

**Output**

Element found in Array at 5 Position with 1 Attempt

**Binary Search**

Given a sorted array arr[] of n elements, write a function to search a given element x in arr[].

A simple approach is to do [linear search](http://quiz.geeksforgeeks.org/linear-search/)**.** The time complexity of above algorithm is O(n). Another approach to perform the same task is using Binary Search.

**Binary Search:** Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.

Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Example:



The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
4. Else (x is smaller) recur for the left half.

**Recursive**implementation of Binary Search

// C# implementation of recursive Binary Search

using System;

class GFG {

    // Returns index of x if it is present in

    // arr[l..r], else return -1

    static int binarySearch(int[] arr, int l,

                            int r, int x)

    {

        if (r >= l) {

            int mid = l + (r - l) / 2;

            // If the element is present at the

            // middle itself

            if (arr[mid] == x)

                return mid;

            // If element is smaller than mid, then

            // it can only be present in left subarray

            if (arr[mid] > x)

                return binarySearch(arr, l, mid - 1, x);

            // Else the element can only be present

            // in right subarray

            return binarySearch(arr, mid + 1, r, x);

        }

        // We reach here when element is not present

        // in array

        return -1;

    }

    // Driver method to test above

    public static void Main()

    {

        int[] arr = { 2, 3, 4, 10, 40 };

        int n = arr.Length;

        int x = 10;

        int result = binarySearch(arr, 0, n - 1, x);

        if (result == -1)

            Console.WriteLine("Element not present");

        else

            Console.WriteLine("Element found at index "

                              + result);

    }

}

**Output:**

Element is present at index 3

**Iterative**implementation of Binary Search

// C# implementation of iterative Binary Search

using System;

class GFG {

    // Returns index of x if it is present in arr[],

    // else return -1

    static int binarySearch(int[] arr, int x)

    {

        int l = 0, r = arr.Length - 1;

        while (l <= r) {

            int m = l + (r - l) / 2;

            // Check if x is present at mid

            if (arr[m] == x)

                return m;

            // If x greater, ignore left half

            if (arr[m] < x)

                l = m + 1;

            // If x is smaller, ignore right half

            else

                r = m - 1;

        }

        // if we reach here, then element was

        // not present

        return -1;

    }

    // Driver method to test above

    public static void Main()

    {

        int[] arr = { 2, 3, 4, 10, 40 };

        int n = arr.Length;

        int x = 10;

        int result = binarySearch(arr, x);

        if (result == -1)

            Console.WriteLine("Element not present");

        else

            Console.WriteLine("Element found at "

                              + "index " + result);

    }

}

**Output:**

Element is present at index 3

**Time Complexity:**  
The time complexity of Binary Search can be written as

T(n) = T(n/2) + c