Topics to cover:

1. Searching algorithms

1. Searching algorithms

1.1 What are 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:

1.1.1 Sequential Search

In this, the list or array is traversed sequentially and every element is checked. For example: Linear Search.

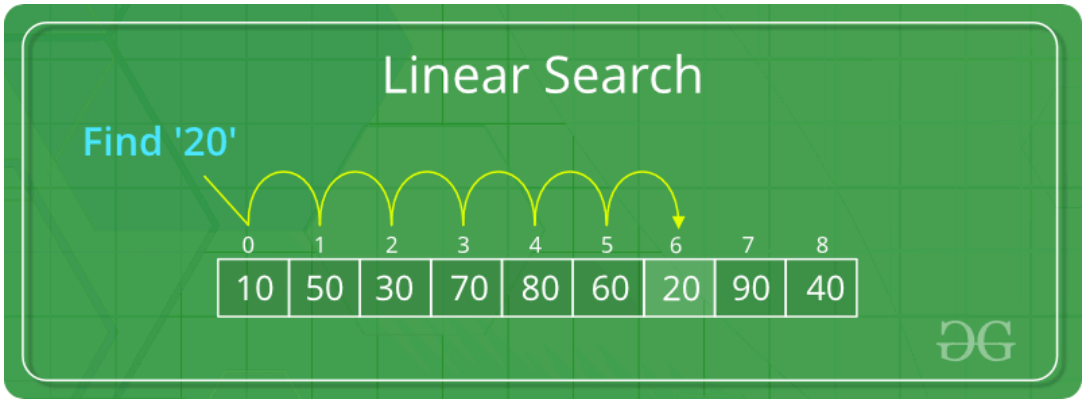
1.1.2 Interval Search:

These algorithms are specifically designed for searching in sorted data-structures. These types 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.

We will now discuss Linear search and Binary search algorithms.

1.1.1.1 Linear search

Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set.



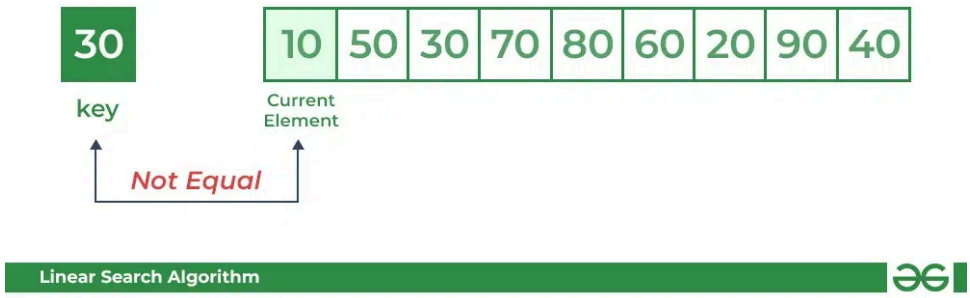
How Does Linear Search Algorithm Work?

- In Linear Search Algorithm, every element is considered as a potential match for the key and checked for the same.
- If any element is found equal to the key, the search is successful and the index of that element is returned.
- If no element is found equal to the key, the search yields “No match found”.

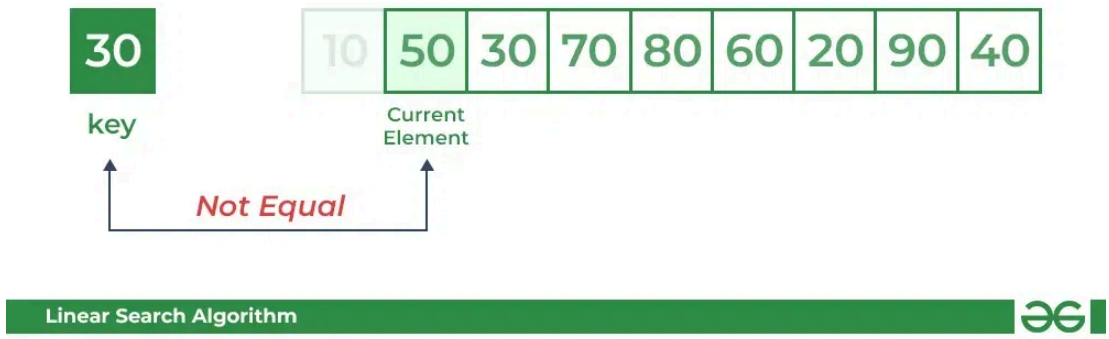
For example: Consider the array arr[] = {10, 50, 30, 70, 80, 20, 90, 40} and key = 30

Step 1:

- Start from the first element (index 0) and compare key with each element (arr[i]).
- Comparing key with first element arr[0]. SInce not equal, the iterator moves to the next element as a potential match.

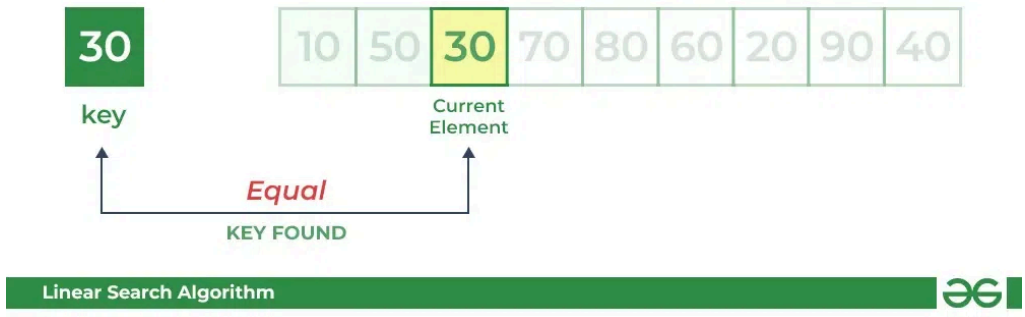


- Comparing key with next element arr[1]. SInce not equal, the iterator moves to the next element as a potential match.



Step 2:

- Now when comparing arr[2] with key, the value matches. So the Linear Search Algorithm will yield a successful message and return the index of the element when key is found (here 2).



Implementation of Linear Search Algorithm:

Below is the implementation of the linear search algorithm in Java

```
import java.io.*;

class LinearSearch {
    public static int search(int arr[], int N, int x)
    {
        for (int i = 0; i < N; i++) {
            if (arr[i] == x)
```

```
            return i;
        }
        return -1;
    }

    public static void main(String args[])
    {
        int arr[] = { 2, 3, 4, 10, 40 };
        int x = 10;

        // Function call
        int result = search(arr, arr.length, x);
        if (result == -1)
            System.out.print(
                "Element is not present in array");
        else
            System.out.print("Element is present at index "
                             + result);
    }
}
```

Output:

```
Element is present at index 3
```

Complexity Analysis of Linear Search:

Time Complexity:

- **Best Case:** In the best case, the key might be present at the first index. So the best case complexity is O(1)
- **Worst Case:** In the worst case, the key might be present at the last index i.e., opposite to the end from which the search has started in the list. So the worst-case complexity is O(N) where N is the size of the list.
- **Average Case:** O(N)

Auxiliary Space: O(1) as except for the variable to iterate through the list, no other variable is used.

Advantages of Linear Search:

- Linear search can be used irrespective of whether the array is sorted or not. It can be used on arrays of any data type.
- Does not require any additional memory.
- It is a well-suited algorithm for small datasets.

Drawbacks of Linear Search:

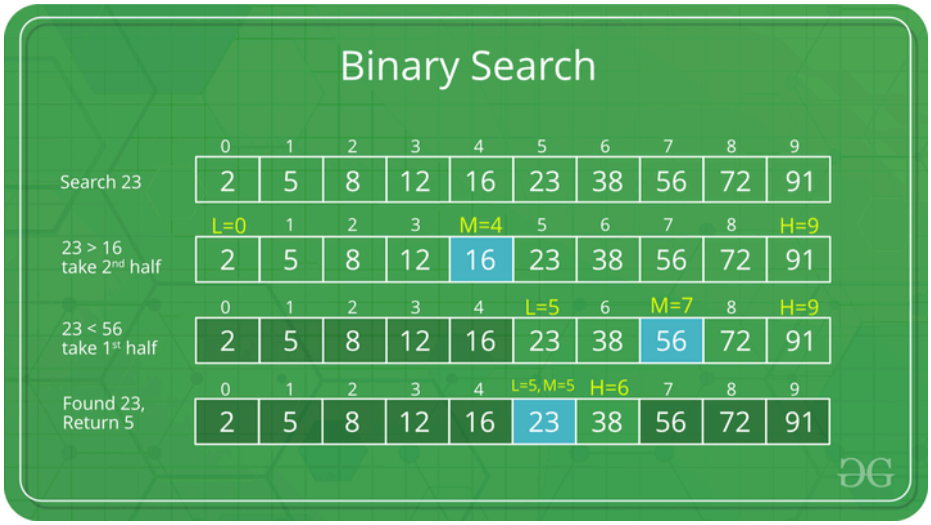
- Linear search has a time complexity of O(N), which in turn makes it slow for large datasets.
- Not suitable for large arrays.

When to use Linear Search?

- When we are dealing with a small dataset.
- When you are searching for a dataset stored in contiguous memory.

1.1.2.1 Binary search

Binary Search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log N).



Conditions for when to apply Binary Search in a Data Structure:

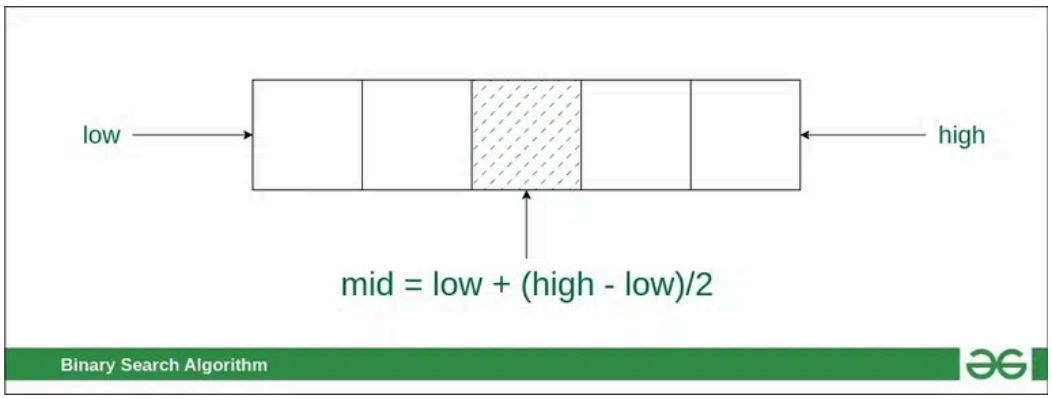
To apply Binary Search algorithm:

- The data structure must be sorted.
- Access to any element of the data structure takes constant time.

Binary Search Algorithm:

In this algorithm,

- Divide the search space into two halves by finding the middle index “mid”.



- Compare the middle element of the search space with the key.
- If the key is found at middle element, the process is terminated.
- If the key is not found at middle element, choose which half will be used as the next search space.
    - If the key is smaller than the middle element, then the left side is used for next search.
    - If the key is larger than the middle element, then the right side is used for next search.
- This process is continued until the key is found or the total search space is exhausted.

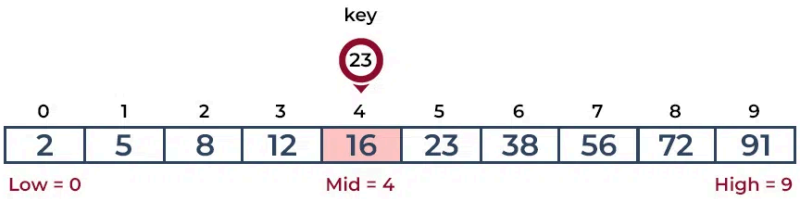
How does Binary Search work?

To understand the working of binary search, consider the following illustration:

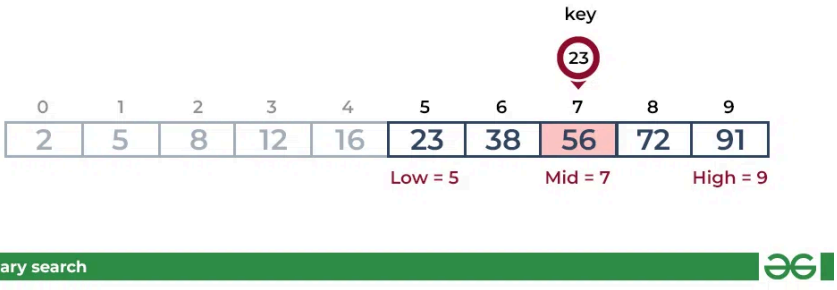
Consider an array arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91}, and the target = 23.

First Step: Calculate the mid and compare the mid element with the key. If the key is less than mid element, move to left and if it is greater than the mid then move search space to the right.

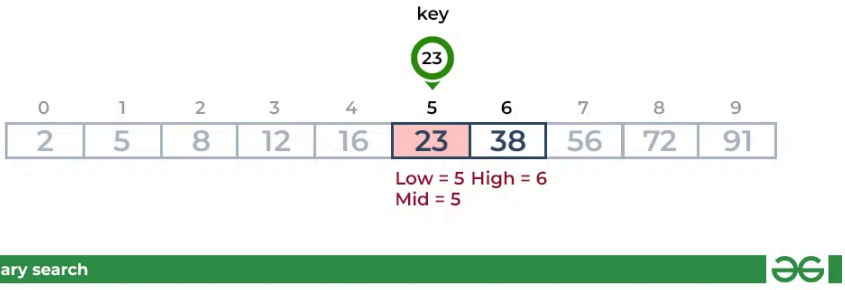
- Key (i.e., 23) is greater than current mid element (i.e., 16). The search space moves to the right.



- Key is less than the current mid 56. The search space moves to the left.



Second Step: If the key matches the value of the mid element, the element is found and stop search.



How to Implement Binary Search?
The Binary Search Algorithm can be implemented in the following two ways

- Iterative Binary Search Algorithm
- Recursive Binary Search Algorithm

We will implement the Recursive Binary Search Algorithm here.

Recursive Binary Search:
Create a recursive function and compare the mid of the search space with the key. And based on the result either return the index where the key is found or call the recursive function for the next search space.

```
class BinarySearch {

    // Returns index of x if it is present in arr[l..
    // r], else return -1
    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 code
    public static void main(String args[])
    {
        BinarySearch ob = new BinarySearch();
        int arr[] = { 2, 3, 4, 10, 40 };
        int n = arr.length;
        int x = 10;
```

```
        int result = ob.binarySearch(arr, 0, n - 1, x);
        if (result == -1)
            System.out.println(
                "Element is not present in array");
        else
            System.out.println(
                "Element is present at index " + result);
    }
}
```

Output

Element is present at index 3

Complexity Analysis of Binary Search:

Time Complexity:

- Best Case: O(1)
- Average Case: O(log N)
- Worst Case: O(log N)

Auxiliary Space: O(1), If the recursive call stack is considered then the auxiliary space will be O(logN).

Advantages of Binary Search:

- Binary search is faster than linear search, especially for large arrays.
- Binary search is well-suited for searching large datasets that are stored in external memory, such as on a hard drive or in the cloud.

Drawbacks of Binary Search:

- The array should be sorted.
- Binary search requires that the data structure being searched be stored in contiguous memory locations.
- Binary search requires that the elements of the array be comparable, meaning that they must be able to be ordered.

Applications of Binary Search:

- Binary search can be used as a building block for more complex algorithms used in machine learning, such as algorithms for training neural networks or finding the optimal hyperparameters for a model.
- It can be used for searching in computer graphics such as algorithms for ray tracing or texture mapping.
- It can be used for searching a database.