Lecture

NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

Searching Algorithms

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

Sequential Search:

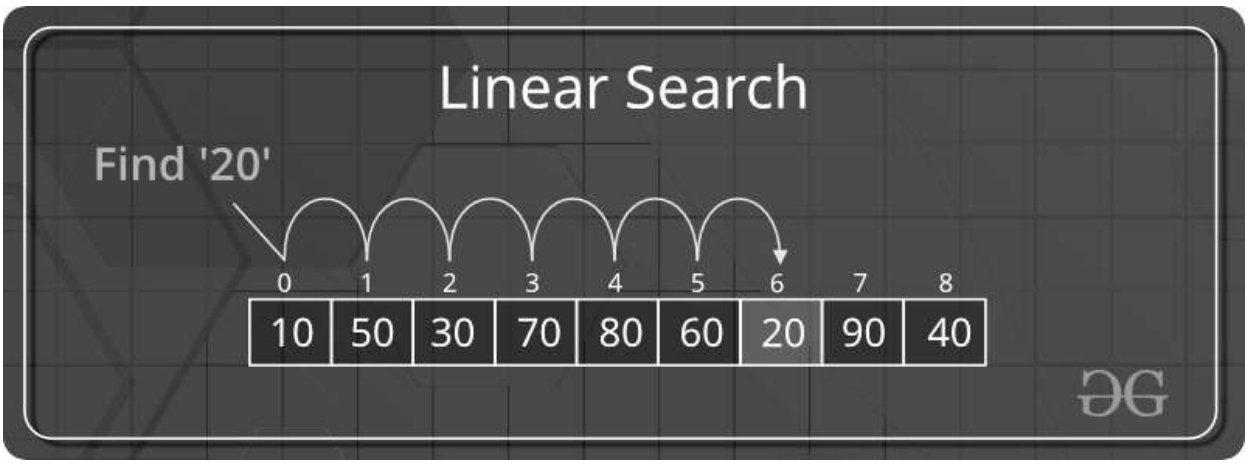
In this algorithm, the list or array is traversed sequentially and every element is checked. The search process starts at the first element and it will continue until the item is found or it reaches the end of data structure. The example for this type of search is ***Linear Search*** which we will discuss later in this lecture.

Interval Search: (Divide-and-Conquer)

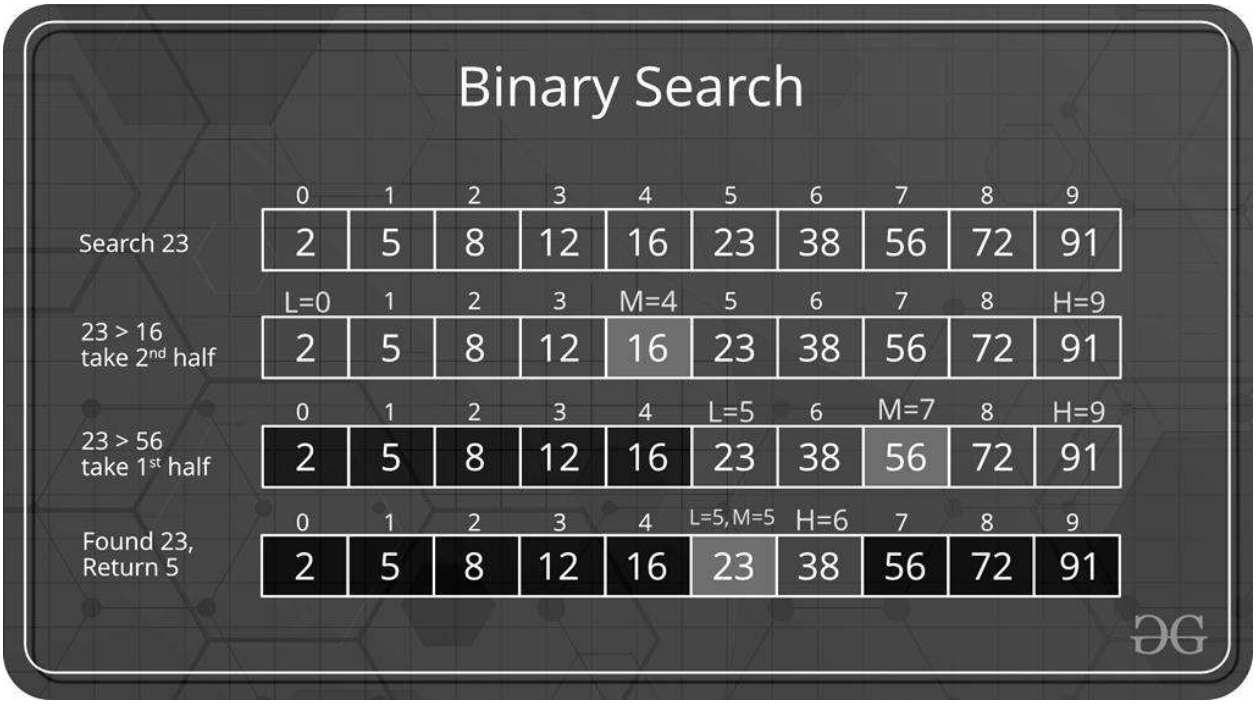
These algorithms are specifically designed for searching in sorted data-structures. This type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half. The example for interval search is ***Binary Search***.

Below are the example to present the graphically representation of the process.

Example 1: Linear Search to find the element/key `'20'` in a given list of numbers.



Example 2: Binary Search to find the element `'23'` in a given list of numbers



Linear Search Code:

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

```
public int linear_Search(int arr[], int key)
{
   for(int i = 0; i < arr.length; i++)
   {
        if(arr[i] == key)
        {
            return i;
        }
   }
   return -1;
}
```

Binary Search Code:

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

```
public int binarySearch(int arr[], int low, int high, int key)
{
    if (high >= low)
    {
        int mid = low + (high - low) / 2;

        if(arr[mid] == key)
        {
            return mid;
        }
        else if(arr[mid] > key)
        {
            return binarySearch(arr, low, mid - 1, key);
        }
        else
        {
            return binarySearch(arr, mid + 1, high, key);
        }
    }
    return -1;
}
```

Main method Code

Below is the main program code to find a key using both methods in java.

```
class Searching
{
    public static int linearSearch(int arr[], int key)
    {
        for(int i = 0; i < arr.length; i++)
        {
            if(arr[i] == key)
                return i;
        }
        return -1;
    }

     public static int binarySearch(int arr[], int low, int high, int key)
     {
         if (high >= low)
         {
             int mid = low + (high - low) / 2;

             if(arr[mid] == key)
                 return mid;
```

```
            else if(arr[mid] > key)
                return binarySearch(arr, low, mid - 1, key);

            else
                return binarySearch(arr, mid + 1, high, key);
        }
        return -1;
    }

    public static void main(String args[])
    {
        int arr[] = {2, 3, 4, 10, 40, 45, 56, 70};
        int x = 10;
        int result = 0;

        // Search using Linear Search
        result = linearSearch(arr, x);

        if(result == -1)
            System.out.print("Linear Search: Element not Found");
        else
            System.out.print("Linear Search: Element at Index: " + result);

        // Search using Binary Search
        result = 0;
        result = binarySearch(arr, 0, arr.length - 1, x);

        if(result == -1)
            System.out.print("\nBinary Search: Element not Found");
        else
            System.out.print("\nBinary Search: Element at Index: " + result);
    }
}
```

Output:

```
Linear Search: Element at Index: 3
Binary Search: Element at Index: 3
```