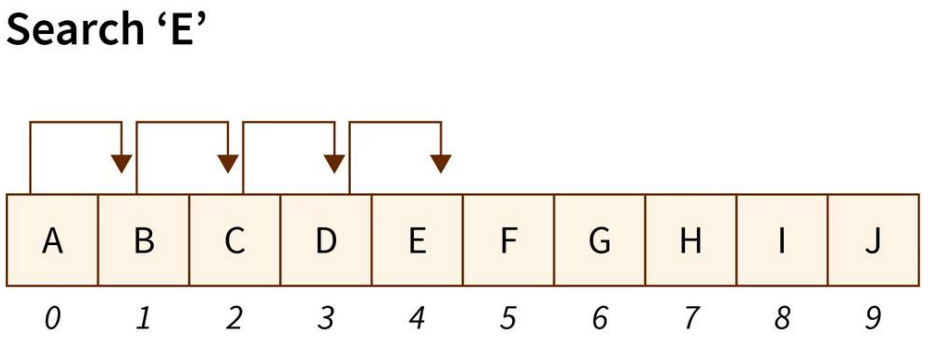
Data Structures and Algorithms TICT2113

Linear Search

- A linear search, sometimes referred to as a sequential search.
- simply scans each element one at a time.
- In this search, an array is sequentially traversed from the first element until the element is found or the end of the array is reached.
- This method can only be suitable for searching over a small array or an unsorted array.

How Linear Search Works?

- Sequentially checks each element in the list until it finds a match or exhausts the list.



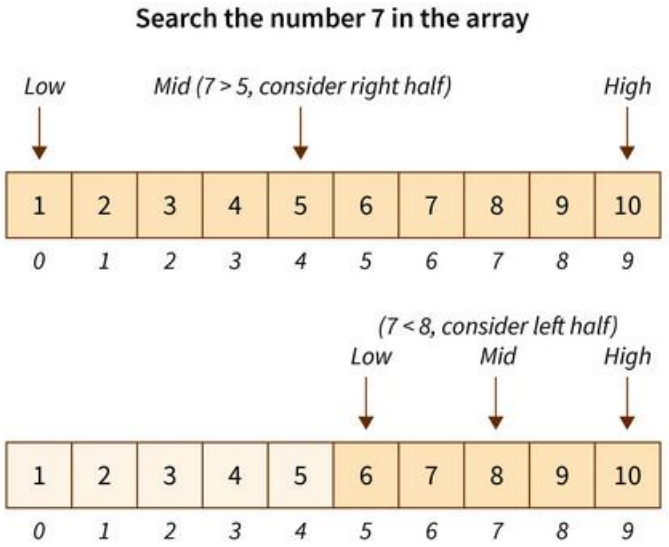
Binary Search

- The Binary search method is only suitable for **searching in a sorted array**.
- In this method, the element that has to be searched is compared to the array's middle element. Search is considered successful only if it matches the target.
- The binary search algorithm uses the divide-and-conquer approach, it does not scan every element in the list, it only searches half of the list instead of going through each element.
- Hence said to be the best searching algorithm because it is faster to execute as compared to Linear search.

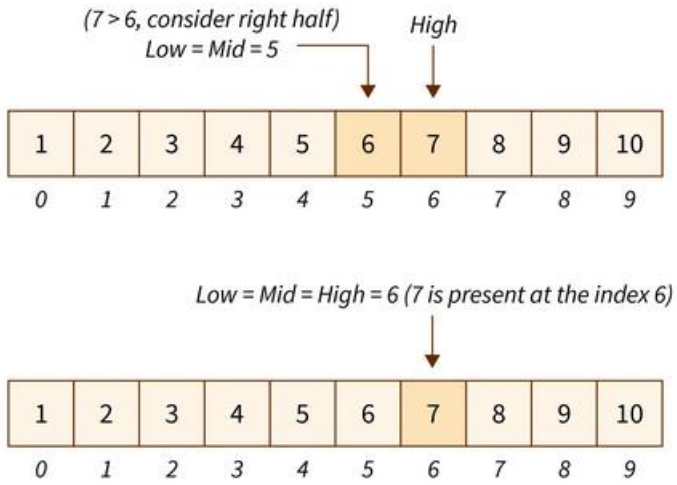
How Binary Search Works?

Consider a sorted list of numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. To find the number 7:

1.Determine the middle element (5).
2.Compare the middle element with the target element (7).
3.Since 7 is greater than 5, eliminate the left half of the list.
4.Repeat the process with the remaining half until the target element is found.



How Binary Search Works? (Cont.)



Important Differences

Linear Search	Binary Search
Input data need not to be in sorted.	Input data need to be in sorted order.
It is also called sequential search.	It is also called half-interval search.
The time complexity of linear search **O(n)**.	The time complexity of binary search **O(log n)**.
Multidimensional array can be used.	Only single dimensional array is used.
Linear search performs equality comparisons	Binary search performs ordering comparisons
It is less complex.	It is more complex.
It is very slow process.	It is very fast process.

Linear Search Example in Java

LinearSearchExample.java

```java
public class LinearSearchExample{

    public static void main(String a[]){
        int[] a1= {10,20,30,50,70,90};    // Initialize an array with some values
        int key = 21;    // Define the key to be searched for

        // Call the linearSearch method and print the result
        System.out.println(key+" is found at index: "+linearSearch(a1, key));
    }

    public static int linearSearch(int[] arr, int key){    // Define a method for linear search
        for(int i=0;i<arr.length;i++){    // Iterate through the array
            if(arr[i] == key){    // Check if the current element is equal to the key
                return i;    // If found, return the index of the element
            }
        }
        return -1;    // If key is not found, return -1
    }
}
```

Binary Search Example in Java

BinarySearchExample.java

```java
public class BinarySearchExample{

    // Main method to test the binary search implementation
    public static void main(String args[]){
        int arr[] = {10,20,30,40,50};
        int key = 30;
        int last=arr.length-1;

        // Calling the binarySearch method with the array, starting index, ending index, and key to search
        binarySearch(arr,0,last,key);
    }

    // Method to perform binary search on an array
    public static void binarySearch(int arr[], int first, int last, int key){
        // Calculating the middle index of the array
        int mid = (first + last)/2;

        // Loop until the first index is less than or equal to the last index
        while( first <= last ){
            // If the element at mid index is less than the key, update the first index
            if ( arr[mid] < key ){
                first = mid + 1;
            }
            // If the element at mid index is equal to the key, the element is found
            else if ( arr[mid] == key ){
                System.out.println("Element is found at index: " + mid);
                // Exit the loop
                break;
            }
            // If the element at mid index is greater than the key, update the last index
            else{
                last = mid - 1;
            }
            // Recalculate the mid index
            mid = (first + last)/2;
        }

        // If first index becomes greater than last index, the element is not found
        if ( first > last ){
            System.out.println("Element is not found!");
        }
    }

}
```