June2

01 June 2020 10:50

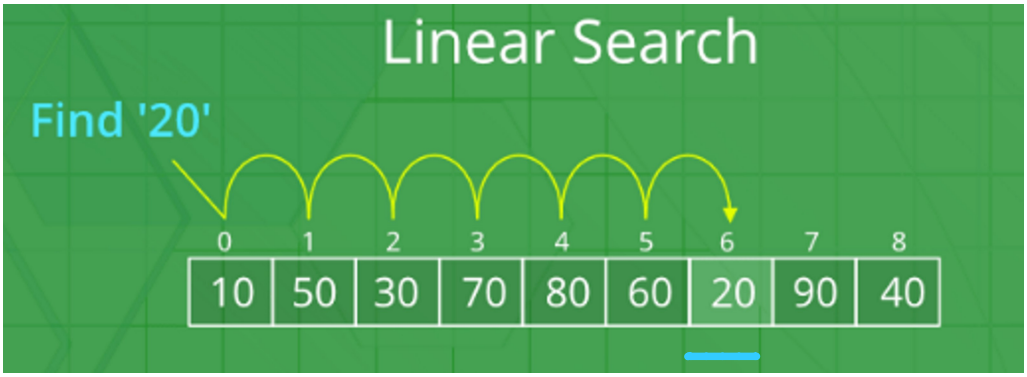
Search Algorithms

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.
2. **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 centre of the search structure and divide the search space in half. For Example: Binary Search.

- Linear Search
- Binary Search

Linear Search -:



Algorithm

- 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

The time complexity of above algorithm is O(n)

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.

Algorithm

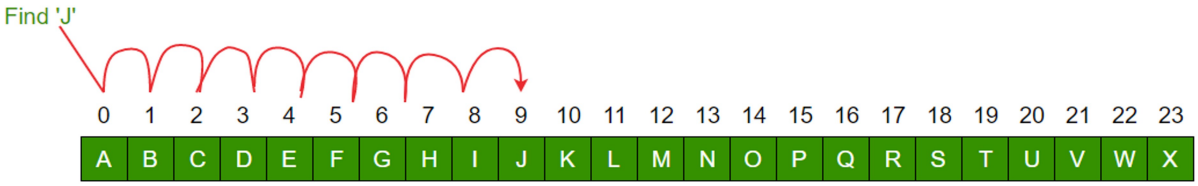
- Compare x with the middle element.
- If x matches with middle element, we return the mid index.
- Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
- Else (x is smaller) recur for the left half.

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

Linear Search vs Binary Search

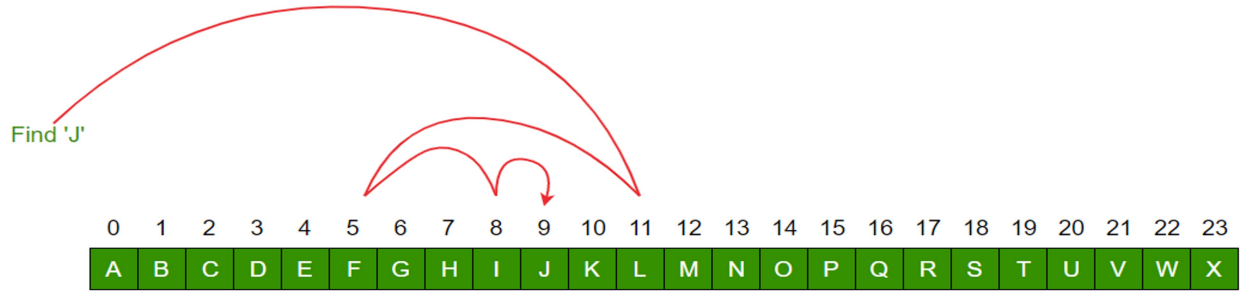
A linear search scans one item at a time, without jumping to any item .

1. The worst case complexity is O(n), sometimes known an O(n) search
2. Time taken to search elements keep increasing as the number of elements are increased.



A binary search however, cut down your search to half as soon as you find middle of a sorted list.

1. The middle element is looked to check if it is greater than or less than the value to be searched.
2. Accordingly, search is done to either half of the given list

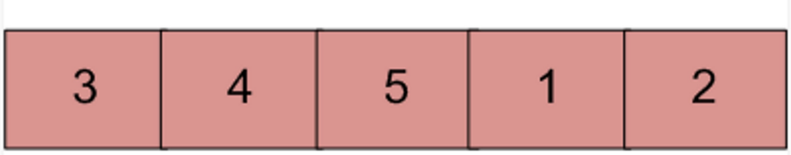


Important Differences

- Input data needs to be sorted in Binary Search and not in Linear Search
- Linear search does the sequential access whereas Binary search access data randomly.
- Time complexity of linear search O(n) , Binary search has time complexity O(log n).
- Linear search performs equality comparisons and Binary search performs ordering comparisons

Search an element in a sorted and rotated array

An element in a sorted array can be found in O(log n) time via binary search. But suppose we rotate an ascending order sorted array at some pivot unknown to you beforehand. So for instance, 1 2 3 4 5 might become 3 4 5 1 2. Devise a way to find an element in the rotated array in O(log n) time.



Examples for reference -

```
Input  : arr[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};
         key = 3
Output : Found at index 8

Input  : arr[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};
         key = 30
Output : Not found

Input : arr[] = {30, 40, 50, 10, 20}
        key = 10
Output : Found at index 3
```

The idea is to find the pivot point, divide the array in two sub-arrays and call binary search.
The main idea for finding pivot is – for a sorted (in increasing order) and pivoted array, pivot element is the only element for which next element to it is smaller than it.

Using above criteria and binary search methodology we can get pivot element in O(logn) time

Algorithm

```
Input arr[] = {3, 4, 5, 1, 2}
Element to Search = 1
  1) Find out pivot point and divide the array in two
      sub-arrays. (pivot = 2) /*Index of 5*/
  2) Now call binary search for one of the two sub-arrays.
      (a) If element is greater than 0th element then
             search in left array
      (b) Else Search in right array
          (1 will go in else as 1 < 0th element(3))
  3) If element is found in selected sub-array then return index
     Else return -1.
```

Questions to be completed

1. Given an array of positive integers. All numbers occur even number of times except one number which occurs odd number of times. Find the number in O(n) time & constant space.

2. Given a sorted array of n integers that has been rotated an unknown number of times, give a O(logn) algorithm that finds an clement in the array. Example: Find 7 in array (15 16 19 20 25 13 4 5 7 10 14) Output: 8 (the index of 7 in the array)

3. Implement Recursive and Iterative binary search an element in a given array