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.**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:

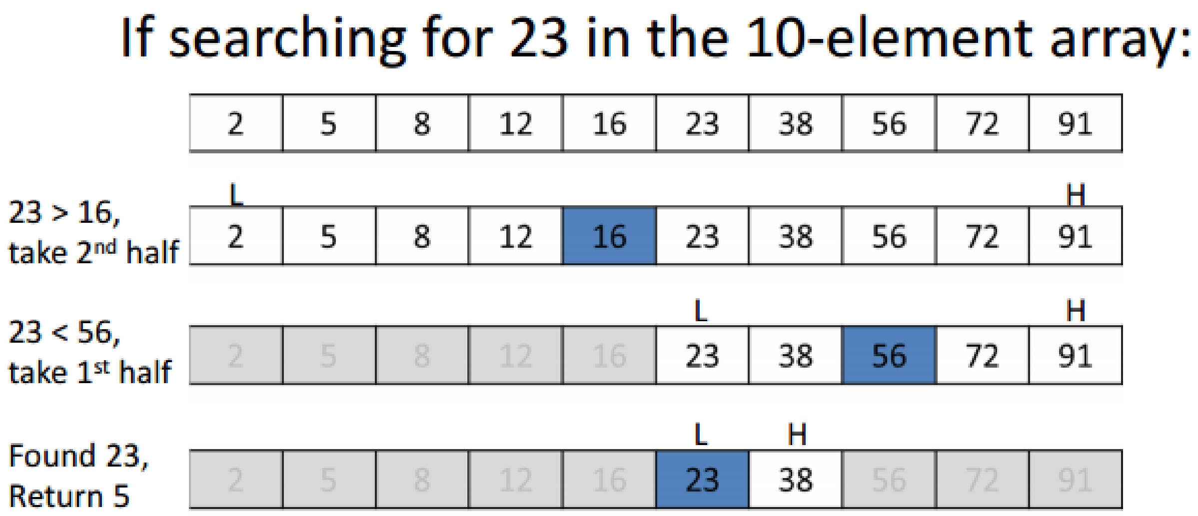

Image Source : http://www.nyckidd.com/bob/Linear%20Search%20and%20Binary%20Search_WorkingCopy.pdf

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

We strongly recommend that you click here and practice it, before moving on to the solution.

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/C++

```
#include <stdio.h>

// A recursive binary search function. It returns location of x in
// given array arr[l..r] is present, otherwise -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;
}

int main(void)
{
   int arr[] = {2, 3, 4, 10, 40};
   int n = sizeof(arr)/ sizeof(arr[0]);
   int x = 10;
   int result = binarySearch(arr, 0, n-1, x);
   (result == -1)? printf("Element is not present in array")
                 : printf("Element is present at index %d", result);
   return 0;
}
```

Run on IDE

Python

```
# Python Program for recursive binary search.

# Returns index of x in arr if present, else -1
def binarySearch (arr, l, r, x):

    # Check base case
    if r >= l:

        mid = l + (r - l)/2

        # If 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
        elif arr[mid] > x:
            return binarySearch(arr, l, mid-1, x)

        # Else the element can only be present in right subarray
        else:
            return binarySearch(arr, mid+1, r, x)

    else:
        # Element is not present in the array
        return -1

# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print "Element is present at index %d" % result
else:
    print "Element is not present in array"
```

Run on IDE

Java

```
// Java implementation of recursive Binary Search
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 method to test above
    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 not present");
        else
            System.out.println("Element found at index "+result);
    }
```

```
}
/* This code is contributed by Rajat Mishra */
```

Run on IDE

Output:

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

Iterative implementation of Binary Search

C/C++

```
#include <stdio.h>

// A iterative binary search function. It returns location of x in
// given array arr[l..r] if present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
  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;
}

int main(void)
{
   int arr[] = {2, 3, 4, 10, 40};
   int n = sizeof(arr)/ sizeof(arr[0]);
   int x = 10;
   int result = binarySearch(arr, 0, n-1, x);
   (result == -1)? printf("Element is not present in array")
                 : printf("Element is present at index %d", result);
   return 0;
}
```

Run on IDE

Python

```
# Iterative Binary Search Function
# It returns location of x in given array arr if present,
# else returns -1
def binarySearch(arr, l, r, x):

    while l <= r:

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

```
        # Check if x is present at mid
        if arr[mid] == x:
            return mid

        # If x is greater, ignore left half
        elif arr[mid] < x:
            l = mid + 1

        # If x is smaller, ignore right half
        else:
            r = mid - 1

    # If we reach here, then the element was not present
    return -1


# Test array
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print "Element is present at index %d" % result
else:
    print "Element is not present in array"
```

Run on IDE

Java

```
// Java implementation of iterative Binary Search
class BinarySearch
{
    // Returns index of x if it is present in arr[], else
    // return -1
    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(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, x);
        if (result == -1)
            System.out.println("Element not present");
```

```
        else
            System.out.println("Element found at index "+result);
    }
}
```

Run on IDE

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
```

The above recurrence can be solved either using Recurrence T ree method or Master method. It falls in case II of Master Method and solution of the recurrence is $\Theta(Logn)$.

Auxiliary Space: O(1) in case of iterative implementation. In case of recursive implementation, O(Logn) recursion call stack space.

Algorithmic Paradigm: Divide and Conquer

Interesting articles based on Binary Search.

- The Ubiquitous Binary Search
- Interpolation search vs Binary search
- Find the minimum element in a sorted and rotated array
- Find a peak element
- Find a Fixed Point in a given array

- Count the number of occurrences in a sorted array
- Median of two sorted arrays
- Floor and Ceiling in a sorted array
- Find the maximum element in an array which is first increasing and then decreasing

Coding Practice Questions on Binary Search

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner   Company Wise Coding Practice

Divide and Conquer Searching Binary-Search

Recommended Posts:

Linear Search
Jump Search
QuickSort
Selection Sort
The Ubiquitous Binary Search | Set 1

(Login to Rate and Mark)

2.5 Average Difficulty : **2.5/5.0**
Based on **2** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments   Share this post!