Tutorial Link https://codequotient.com/tutorials/Searching an Element - Binary Search/5a12ef3746765b2b63e3477e

TUTORIAL

Searching an Element - Binary Search

Chapter

1. Searching an Element - Binary Search

Topics

If the list is sorted, then instead of applying a linear search, we can optimize the things a bit. We can go for binary search. It will search a sorted array by repeatedly dividing the search interval in half. Begin with an interval of the whole array. Compare the middle value of array with the searched element. If the value of the search key is less than the item in the middle of the interval, search in the interval to the lower half. Otherwise search in the interval to the upper half. Repeatedly check until the value is found or the interval is empty. The idea of binary search is to take the benefit of sorted property of array and search efficiently i.e. reduce the time complexity to **O(Log(n))**. We basically ignore half of the elements after each comparison. For example, the following array is given and we have to search element 7 in it: -

```
Value   1   2   4   5   6   7   9   12   15
Index   0   1   2   3   4   5   6    7    8
```

First the algorithm will find the middle index and check the element at middle with the searched element. middle element is 6 which is lesser than searched element 7, so algorithm search in the upper half of the array again as below: -

```
Value   7   9   12   15
Index   5   6    7    8
```

Now repeating the same procedure, the middle index is calculated, say 6 is chosen and element at index 6 is 9 which is greater than the searched element so algorithm will search in the left part of this array now as below: -

```
Value   7
Index   5
```

As there is only one element, it will be compared with the searched element. It is equal to the searched element so index 5 is returned.

General algorithm for binary search

```
X = searched_Element
left = 0
right = length_of_array
mid = left + (right-left) / 2
While(left <= right)
     If: X == array[mid] then RETURN mid
     Else: If X < array[mid] right = mid-1
     Else: left = mid +1
End
RETURN -1
```

The iterative implementation of binary search is as below: -

```c
#include<stdio.h>

int binary_search(int A[], int left, int right, int key)
{
  int m;
  while( left <= right )
  {
    m = left + (right-left)/2;
    if( A[m] == key )     // Element found
      return m;
    if( A[m] < key )         // Search in right part of list
      left = m + 1;
    else            // Search in left part of list
      right = m - 1;
  }
  return -1;
}

int main()
{
  int loc, x, array[]={10,11,12,13,14,25,26,37,48,59};
  x = 26;        // element to be searched in the array
  loc=binary_search(array,0,10,x);
  if(loc != -1)
    printf("Element found at location : %d",loc);
  else
    printf("Element not present in the array.");
  return 0;
```

```
}

```

Java

```
import java.util.Scanner;
// Other imports go here
// Do NOT change the class name
class Main{
  static int binary_search(int A[], int left, int right, int key)
  {
    int m;
    while( left <= right )
    {
      m = left + (right-left)/2;
      if( A[m] == key )     // Element found
        return m;
      if( A[m] < key )         // Search in right part of list
        left = m + 1;
      else            // Search in left part of list
        right = m - 1;
    }
    return -1;
  }

  public static void main(String[] args)
  {
    int loc, x, array[]={10,11,12,13,14,25,26,37,48,59};
    x = 26;        // element to be searched in the array
    loc=binary_search(array,0,10,x);
    if(loc != -1)
      System.out.print("Element found at location : " + loc);
    else
      System.out.print("Element not present in the array.");

  }
}
```

The output of above program is as below for different runs: -

```
Element found at location :6
```

Recursive implementation of Binary Search

C

```
#include<stdio.h>

int rec_binary_search(int arr[], int left, int right, int x) {
  int result;
  if (right >= left) {
    int mid = left + (right - left)/2;
```

```
    if (arr[mid] == x)  return mid;
    if (arr[mid] > x) return rec_binary_search(arr, left, mid-1, x);
    result = rec_binary_search(arr, mid+1, right, x);
    return result;
  }
  return -1;       // when element is not present in array.
}

int main() {
  int loc,x,array[]={10,11,12,13,14,25,26,37,48,59};
  x=11;        // element to be searched in the array
  loc=rec_binary_search(array,0,10,x);
  if(loc != -1)
    printf("Element found at location : %d",loc);
  else
    printf("Element not present in the array.");
  return 0;
}
```

Java

```
import java.util.Scanner;
// Other imports go here
// Do NOT change the class name
class Main{
  static int rec_binary_search(int arr[], int left, int right, int x) {
  int result;
  if (right >= left) {
    int mid = left + (right - left)/2;
    if (arr[mid] == x)  return mid;
    if (arr[mid] > x) return rec_binary_search(arr, left, mid-1, x);
    result = rec_binary_search(arr, mid+1, right, x);
    return result;
  }
  return -1;       // when element is not present in array.
}

  public static void main(String[] args)
  {
    int loc, x, array[]={10,11,12,13,14,25,26,37,48,59};
    x = 11;        // element to be searched in the array
    loc=rec_binary_search(array,0,10,x);
    if(loc != -1)
      System.out.print("Element found at location : " + loc);
    else
      System.out.print("Element not present in the array.");

  }
```

```
28 }
```

The output of above program is as below for different runs: -

```
Element found at location :1
```

Common differences between Linear search and Binary Search

- Input data needs to be sorted in Binary Search but not in Linear Search, so we can apply according to situation.
- Linear search does the sequential access whereas Binary search does the random access, so the list must provide the accessing in same manner.
- Time complexity of linear search is **O(n)** whereas Binary search has time complexity **O(log n)**.
- Linear search performs equality comparisons whereas Binary search performs ordering comparisons.

The following figure shows the search process in two cases: -