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.


```
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
```

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

All solutions provided here assume that all elements in array are distinct.

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 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

```
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.
```

Implementation:

```
/* Program to search an element in a sorted and pivoted array*/
#include <stdio.h>

int findPivot(int[], int, int);
int binarySearch(int[], int, int, int);

/* Searches an element key in a pivoted sorted array arrp[]
   of size n */
int pivotedBinarySearch(int arr[], int n, int key)
{
   int pivot = findPivot(arr, 0, n-1);

   // If we didn't find a pivot, then array is not rotated at all
   if (pivot == -1)
       return binarySearch(arr, 0, n-1, key);

   // If we found a pivot, then first compare with pivot and then
   // search in two subarrays around pivot
   if (arr[pivot] == key)
       return pivot;
   if (arr[0] <= key)
       return binarySearch(arr, 0, pivot-1, key);
   return binarySearch(arr, pivot+1, n-1, key);
}

/* Function to get pivot. For array 3, 4, 5, 6, 1, 2 it returns
   3 (index of 6) */
int findPivot(int arr[], int low, int high)
{
   // base cases
   if (high < low)  return -1;
   if (high == low) return low;

   int mid = (low + high)/2;   /*low + (high - low)/2;*/
   if (mid < high && arr[mid] > arr[mid + 1])
       return mid;
   if (mid > low && arr[mid] < arr[mid - 1])
       return (mid-1);
   if (arr[low] >= arr[mid])
       return findPivot(arr, low, mid-1);
   return findPivot(arr, mid + 1, high);
}

/* Standard Binary Search function*/
int binarySearch(int arr[], int low, int high, int key)
{
   if (high < low)
       return -1;
   int mid = (low + high)/2;  /*low + (high - low)/2;*/
   if (key == arr[mid])
       return mid;
   if (key > arr[mid])
       return binarySearch(arr, (mid + 1), high, key);
   return binarySearch(arr, low, (mid -1), key);
}

/* Driver program to check above functions */
```

```
int main()
{
   // Let us search 3 in below array
   int arr1[] = {5, 6, 7, 8, 9, 10, 1, 2, 3};
   int n = sizeof(arr1)/sizeof(arr1[0]);
   int key = 3;
   printf("Index: %d\n", pivotedBinarySearch(arr1, n, key));
   return 0;
}
```

Run on IDE

Output:

```
Index of the element is 8
```

Time Complexity O(logn). Thanks to Ajay Mishra for initial solution.

Improved Solution:

We can search an element in one pass of Binary Search. The idea is to search

```
1) Find middle point mid = (l + h)/2
2) If key is present at middle point, return mid.
3) Else If arr[l..mid] is sorted
    a) If key to be searched lies in range from arr[l]
       to arr[mid], recur for arr[l..mid].
    b) Else recur for arr[mid+1..r]
4) Else (arr[mid+1..r] must be sorted)
    a) If key to be searched lies in range from arr[mid+1]
       to arr[r], recur for arr[mid+1..r].
    b) Else recur for arr[l..mid]
```

Below is C++ implementation of above idea.

```
// Search an element in sorted and rotated array using
// single pass of Binary Search
#include<bits/stdc++.h>
using namespace std;

// Returns index of key in arr[l..h] if key is present,
// otherwise returns -1
int search(int arr[], int l, int h, int key)
{
    if (l > h) return -1;

    int mid = (l+h)/2;
    if (arr[mid] == key) return mid;

    /* If arr[l...mid] is sorted */
    if (arr[l] <= arr[mid])
    {
        /* As this subarray is sorted, we can quickly
           check if key lies in half or other half */
        if (key >= arr[l] && key <= arr[mid])
           return search(arr, l, mid-1, key);

        return search(arr, mid+1, h, key);
    }

    /* If arr[l..mid] is not sorted, then arr[mid... r]
       must be sorted*/
    if (key >= arr[mid] && key <= arr[h])
```

```
        return search(arr, mid+1, h, key);

    return search(arr, l, mid-1, key);
}

// Driver program
int main()
{
    int arr[] = {4, 5, 6, 7, 8, 9, 1, 2, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    int key = 6
    int i = search(arr, 0, n-1, key);
    if (i != -1) cout << "Index: " << i << endl;
    else cout << "Key not found\n";
}
```

Run on IDE

Output:

```
Index: 2
```

Thanks to Gaurav Ahirwar for suggesting above solution.

How to handle duplicates?

It doesn't look possible to search in O(Logn) time in all cases when duplicates are allowed. For example consider searching 0 in {2, 2, 2, 2, 2, 2, 2, 2, 0, 2} and {2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2}. It doesn't look possible to decide whether to recur for left half or right half by doing constant number of comparisons at the middle.

Search an element in a sorted and rotated array | GeeksforGeeks

Asked in: Adobe, Amazon, BankBazaar, Bharti SoftBank, DE Shaw, Flipkart, Intuit , Microsoft, Snapdeal, Times Internet

Similar Articles:

Find the minimum element in a sorted and rotated array

Given a sorted and rotated array, find if there is a pair with a given sum.

Please write comments if you find any bug in above codes/algorithms, or find other ways to solve the same problem.

GATE CS Corner Company Wise Coding Practice

Arrays Searching Binary-Search rotation

Recommended Posts:

Merge an array of size n into another array of size m+n
Find the minimum element in a sorted and rotated array
Median of two sorted arrays
Find the Missing Number
Given a sorted and rotated array, find if there is a pair with a given sum

(Login to Rate and Mark)

3.2 Average Difficulty : **3.2/5.0**
Based on **181** 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!