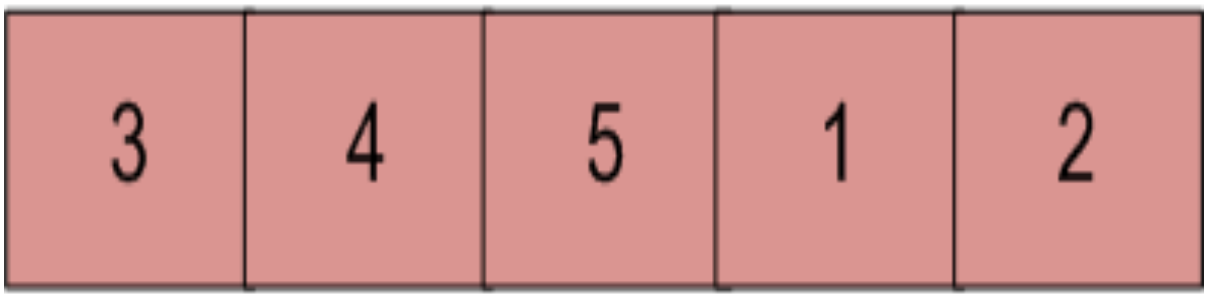
Search an element in a sorted and rotated array

Difficulty Level : Medium • Last Updated : 11 Jul, 2022

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.



Example:

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*

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

Basic Solution:

Approach:

1. The idea is to find the pivot point, divide the array into two sub-arrays and perform a binary search.
2. The main idea for finding a pivot is – for a sorted (in increasing order) and pivoted array, the pivot element is the only element for which the next element to it is smaller than it.
3. Using the above statement and binary search pivot can be found.
4. After the pivot is found divide the array into two sub-arrays.
5. Now the individual sub-arrays are sorted so the element can be searched using Binary Search.

Implementation:

```
Input arr[] = {3, 4, 5, 1, 2}
Element to Search = 1
```

```
        (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.
```

Below is the implementation of the above approach:

C++

```
/* C++ Program to search an element
   in a sorted and pivoted array*/
#include <bits/stdc++.h>
using namespace std;

/* 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);

    // else
    return binarySearch(arr, low, (mid - 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
```

```
        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);
}

/* Searches an element key in a pivoted
   sorted array arr[] 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);
}

/* 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 };
```

```
    // Function calling
    cout << "Index of the element is : "
         << pivotedBinarySearch(arr1, n, key);

    return 0;
}
```

C

```
/* 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)
{
```

```
    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 of the element is : %d",
           pivotedBinarySearch(arr1, n, key));
    return 0;
}
```

Java

```
/* Java program to search an element
```

```
/* Searches an element key in a
   pivoted sorted array arrp[]
   of size n */
static 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) */
static int findPivot(int arr[], int low, int high)
{
    // base cases
    if (high < low)
        return -1;
    if (high == low)
        return low;

    /* low + (high - low)/2; */
    int mid = (low + high) / 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);
```

```
    /* Standard Binary Search function */
    static int binarySearch(int arr[], int low, int high, int key)
    {
        if (high < low)
            return -1;

        /* low + (high - low)/2; */
        int mid = (low + high) / 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);
    }

    // main function
    public static void main(String args[])
    {
        // Let us search 3 in below array
        int arr1[] = { 5, 6, 7, 8, 9, 10, 1, 2, 3 };
        int n = arr1.length;
        int key = 3;
        System.out.println("Index of the element is : "
                           + pivotedBinarySearch(arr1, n, key));
    }
}
```

```
# Python Program to search an element
# in a sorted and pivoted array

# Searches an element key in a pivoted
# sorted array arrp[] of size n
def pivotedBinarySearch(arr, n, key):

    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)
def findPivot(arr, low, high):

    # base cases
    if high < low:
        return -1
    if high == low:
        return low

    # low + (high - low)/2;
    mid = int((low + high)/2)

    if mid < high and arr[mid] > arr[mid + 1]:
        return mid
    if mid > low and arr[mid] < arr[mid - 1]:
        return (mid-1)
```

```
# Standard Binary Search function*/
def binarySearch(arr, low, high, key):

    if high < low:
        return -1

    # low + (high - low)/2;
    mid = int((low + high)/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 */
# Let us search 3 in below array
arr1 = [5, 6, 7, 8, 9, 10, 1, 2, 3]
n = len(arr1)
key = 3
print("Index of the element is : ",
      pivotedBinarySearch(arr1, n, key))

# This is contributed by Smitha Dinesh Semwal
```

C#

```
// C# program to search an element
// in a sorted and pivoted array
using System;

class main {

    // Searches an element key in a
    // pivoted sorted array arrp[]
    // of size n
    static int pivotedBinarySearch(int[] arr,
                                   int n, int key)
```

```
        // 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) */
    static int findPivot(int[] arr, int low, int high)
    {
        // base cases
        if (high < low)
            return -1;
        if (high == low)
            return low;

        /* low + (high - low)/2; */
        int mid = (low + high) / 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);
    }
```

```
                                int high, int key)
    {
        if (high < low)
            return -1;

        /* low + (high - low)/2; */
        int mid = (low + high) / 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 Code
    public static void Main()
    {
        // Let us search 3 in below array
        int[] arr1 = { 5, 6, 7, 8, 9, 10, 1, 2, 3 };
        int n = arr1.Length;
        int key = 3;
        Console.Write("Index of the element is : "
                      + pivotedBinarySearch(arr1, n, key));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP Program to search an element
// in a sorted and pivoted array

// Standard Binary Search function
function binarySearch($arr, $low,
                      $high, $key)
{
    if ($high < $low)
```

```
    $mid = floor($low + $high) / 2;

    if ($key == $arr[$mid])
        return $mid;

    if ($key > $arr[$mid])
        return binarySearch($arr, ($mid + 1),
                                $high, $key);

    else
        return binarySearch($arr, $low,
                    ($mid -1), $key);
}

// Function to get pivot.
// For array 3, 4, 5, 6, 1, 2
// it returns 3 (index of 6)
function findPivot($arr, $low, $high)
{

    // base cases
    if ($high < $low)
        return -1;
    if ($high == $low)
        return $low;

    /*low + (high - low)/2;*/
    $mid = ($low + $high)/2;
    if ($mid < $high and $arr[$mid] >
                    $arr[$mid + 1])
        return $mid;

    if ($mid > $low and $arr[$mid] <
                  $arr[$mid - 1])
        return ($mid - 1);

    if ($arr[$low] >= $arr[$mid])
        return findPivot($arr, $low,
                         $mid - 1);

    return findPivot($arr, $mid + 1, $high);
}
```

```php
// arr[] of size n */
function pivotedBinarySearch($arr, $n, $key)
{
    $pivot = findPivot($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);
}

// Driver Code
// Let us search 3
// in below array
$arr1 = array(5, 6, 7, 8, 9, 10, 1, 2, 3);
$n = count($arr1);
$key = 3;

// Function calling
echo "Index of the element is : ",
      pivotedBinarySearch($arr1, $n, $key);

// This code is contributed by anuj_67.
?>
```

```
/* JavaScript Program to search an element
   in a sorted and pivoted array*/

/* Standard Binary Search function*/
function binarySearch( arr, low,
                  high, key){
    if (high < low)
        return -1;

    let mid = Math.floor((low + high) / 2); /*low + (high - low)/2;*/
    if (key == arr[mid])
        return mid;

    if (key > arr[mid])
        return binarySearch(arr, (mid + 1), high, key);

    // else
    return binarySearch(arr, low, (mid - 1), key);
}

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

    let mid = Math.floor((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);
}
```

```
    let 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);
}

/* Driver program to check above functions */
// Let us search 3 in below array
let arr1 = [ 5, 6, 7, 8, 9, 10, 1, 2, 3 ];
let n = arr1.length;
let key = 3;
// Function calling
document.write( "Index of the element is : "
         + pivotedBinarySearch(arr1, n, key));

</script>
```

Output

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

Complexity Analysis:

- **Time Complexity:** O(log n).
  Binary Search requires log n comparisons to find the element. So time complexity is O(log n).
- **Space Complexity:** O(1), No extra space is required.

Improved Solution:

Approach: Instead of two or more passes of binary search the result can be found in one pass of binary search. The binary search needs to be modified to perform the search. The idea is to create a recursive function that takes l and r as a range in input and the key.

```
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..h]
4) Else (arr[mid+1..h] must be sorted)
    a) If key to be searched lies in range from arr[mid+1]
       to arr[h], recur for arr[mid+1..h].
    b) Else recur for arr[l..mid]
```

Below is the implementation of the above idea:

C++

```
// 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)
```

```
        /* 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);
        /*If key not lies in first half subarray,
           Divide other half  into two subarrays,
           such that we can quickly check if key lies
           in other half */
        return search(arr, mid + 1, h, key);
    }

    /* If arr[l..mid] first subarray is not sorted, then
    arr[mid... h] must be sorted subarray */
    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 = 3;
    int i = search(arr, 0, n - 1, key);

    if (i != -1)
        cout << "Index: " << i << endl;
    else
        cout << "Key not found";
}

// This code is contributed by Aditya Kumar (adityakumar129)
```

C

```
// Search an element in sorted and rotated
// array using single pass of Binary Search
#include <stdio.h>
```

```
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);
        /*If key not lies in first half subarray,
           Divide other half  into two subarrays,
           such that we can quickly check if key lies
           in other half */
        return search(arr, mid + 1, h, key);
    }

    /* If arr[l..mid] first subarray is not sorted, then
    arr[mid... h] must be sorted subarray */
    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 = 3;
    int i = search(arr, 0, n - 1, key);

    if (i != -1)
        printf("Index: %d\n", i);
    else
        printf("Key not found");
```

Java

```
/* Java program to search an element in
   sorted and rotated array using
   single pass of Binary Search*/

class Main {
    // Returns index of key in arr[l..h]
    // if key is present, otherwise returns -1
    static 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] first subarray 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);
            /*If key not lies in first half subarray,
           Divide other half  into two subarrays,
           such that we can quickly check if key lies
           in other half */
            return search(arr, mid + 1, h, key);
        }

        /* If arr[l..mid] first subarray is not sorted,
           then arr[mid... h] must be sorted subarray*/
        if (key >= arr[mid] && key <= arr[h])
            return search(arr, mid + 1, h, key);

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

    // main function
```

```
        int n = arr.length;
        int key = 3;
        int i = search(arr, 0, n - 1, key);
        if (i != -1)
            System.out.println("Index: " + i);
        else
            System.out.println("Key not found");
    }
}

// This code is contributed by Aditya Kumar (adityakumar129)
```

Python3

```
# Search an element in sorted and rotated array using
# single pass of Binary Search

# Returns index of key in arr[l..h] if key is present,
# otherwise returns -1
def search (arr, l, h, key):
    if l > h:
        return -1

    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] and 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] and key <= arr[h]:
        return search(arr, mid + 1, h, key)
```

```
arr = [4, 5, 6, 7, 8, 9, 1, 2, 3]
key = 3
i = search(arr, 0, len(arr)-1, key)
if i != -1:
    print ("Index: % d"% i)
else:
    print ("Key not found")

# This code is contributed by Shreyanshi Arun
```

C#

```
/* C# program to search an element in
sorted and rotated array using
single pass of Binary Search*/
using System;

class GFG {

    // Returns index of key in arr[l..h]
    // if key is present, otherwise
    // returns -1
    static 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);
```

```
        /* 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);
    }

    // main function
    public static void Main()
    {
        int[] arr = { 4, 5, 6, 7, 8, 9, 1, 2, 3 };
        int n = arr.Length;
        int key = 3;
        int i = search(arr, 0, n - 1, key);

        if (i != -1)
            Console.WriteLine("Index: " + i);
        else
            Console.WriteLine("Key not found");
    }
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// Search an element in sorted and rotated
// array using single pass of Binary Search

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

    $mid = floor(($l + $h) / 2);
    if ($arr[$mid] == $key)
```

```
    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] and
            $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] and
          $key <= $arr[$h])
        return search($arr, $mid + 1,
                           $h, $key);

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

    // Driver Code
    $arr = array( 5, 6, 7, 8, 9, 10, 1, 2, 3 );
    $n = sizeof($arr);
    $key = 3;
    $i = search($arr, 0, $n-1, $key);

    if ($i != -1)
        echo "Index: ", $i, " \n";
    else
        echo "Key not found";

// This code is contributed by ajit
?>
```

```
<script>

// Search an element in sorted and rotated
// array using single pass of Binary Search

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

    let mid = Math.floor((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);
        /*If key not lies in first half subarray,
           Divide other half  into two subarrays,
           such that we can quickly check if key lies
           in other half */
        return search(arr, mid + 1, h, key);
    }

    /* If arr[l..mid] first subarray is not sorted,
    then arr[mid... h]
    must be sorted subarray */
    if (key >= arr[mid] && key <= arr[h])
        return search(arr, mid + 1, h, key);

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

// Driver program
let arr = [ 4, 5, 6, 7, 8, 9, 1, 2, 3 ];
let n = arr.length;
let key = 3;
let i = search(arr, 0, n - 1, key);
```

```
    document.write("Key not found");

</script>
```

Output

```
Index: 8
```

Complexity Analysis:

- **Time Complexity:** O(log n).
  Binary Search requires log n comparisons to find the element. So time complexity is O(log n).
- **Space Complexity:** O(1).
  As no extra space is required.

Thanks to Gaurav Ahirwar for suggesting the 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, 2}.
It doesn't look possible to decide whether to recur for the left half or right half by doing a constant number of comparisons at the middle.

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 the above codes/algorithms, or find other ways to solve the same problem.

Next

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

RECOMMENDED ARTICLES Page : 1 2 3

01 **C# Program for Search an element in a sorted and rotated array**
15, Nov 21

02 **Javascript Program for Search an element in a sorted and rotated array**
15, Nov 21

03 **C++ Program for Search an element in a sorted and rotated array**
15, Nov 21

04 **Java Program for Search an element in a sorted and rotated array**
15, Oct 21

05 **Python3 Program for Search an element in a sorted and rotated array**
15, Nov 21

06 **Php Program for Search an element in a sorted and rotated array**
15, Nov 21

07 **C Program for Search an element in a sorted and rotated array**
15, Nov 21

08 **Search an element in a sorted and rotated array with duplicates**
27, Jan 20

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : jit_t, vt_m, voletiananthkumar, andrew1234, rohan07, rohitsingh07052, vknwl077, anikakapoor, adityakumar129, harendrakumar123, hardikkoriintern

Article Tags : Adobe, Amazon, BankBazaar, Binary Search, D-E-Shaw, FactSet, Flipkart, Hike, MakeMyTrip, Microsoft, Paytm, rotation, Samsung, SAP Labs, Snapdeal, Times Internet, Arrays, Searching

Practice Tags : Paytm, Flipkart, Amazon, Microsoft, Samsung, Snapdeal, D-E-Shaw, FactSet, Hike, MakeMyTrip, Adobe, BankBazaar, Times Internet, SAP Labs, Arrays, Searching, Binary Search

Improve Article | Report Issue

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

Load Comments

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

NodeJS