Search in a Sorted and Rotated Array - GeeksforGeeks

Source: https://www.geeksforgeeks.org/search-an-element-in-a-sorted-and-pivoted-array/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Search in a Sorted and Rotated Array Last Updated : 3 Feb, 2026 Given a sorted and rotated array arr[] of distinct elements, find the index of given key in the array. If the key is not present in the array, return -1 . Examples: Input: arr[] = [5, 6, 7, 8, 9, 10, 1, 2, 3], key = 3 Output: 8 Explanation : 3 is present at index 8. Input: arr[] = [3, 5, 1, 2], key = 6 Output: -1 Explanation : 6 is not present. Input: arr[] = [33, 42, 72, 99], key = 42 Output: 1 Explanation: 42 is found at index 1. Try it on GfG Practice Table of Content [Naive Approach] Using Linear Search - O(n) Time and O(1) Space [Expected Approach 1] Using Binary Search Twice - O(log n) Time and O(1) Space [Expected Approach 2] Using Single Binary Search - O(log n) Time and O(1) Space [Naive Approach] Using Linear Search - O(n) Time and O(1) Space A simple approach is to iterate through the array and check for each element, if it matches the target then return the index, otherwise return -1. To know more about the implementation, please refer Introduction to Linear Search Algorithm . C++ #include <iostream> #include <vector> using namespace std ; int search (vector < int >& arr , int key) { for (int i = 0 ; i < arr . size (); i ++) { // Check each element one by one if (arr [i] == key) return i ; } // Key not found return -1 ; } int main () { vector < int > arr = { 5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3 }; int key = 3 ; int index = search (arr , key) ; cout << index << endl ; return 0 ; } Java public class GfG { static int search (int [] arr , int key) { for (int i = 0 ; i < arr . length ; i ++) { // Check each element one by one if (arr [i] == key) return i ; } // Key not found return - 1 ; } public static void main (String [] args) { int [] arr = { 5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3 }; int key = 3 ; int index = search (arr , key); System . out . println (index); } } Python def search (arr , key): for i in range (len (arr)): # Check each element one by one if arr [i] == key : return i # Key not found return - 1 if __name__ == "__main__" : arr = [5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3] key = 3 index = search (arr , key) print (index) C# using System ; class GfG { static int search (int [] arr , int key) { for (int i = 0 ; i < arr . Length ; i ++) { // Check each element one by one if (arr [i] == key) return i ; } // Key not found return - 1 ; } static void Main () { int [] arr = { 5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3 }; int key = 3 ; int index = search (arr , key); Console . WriteLine (index); } } JavaScript function search (arr , key) { for (let i = 0 ; i < arr . length ; i ++) { // Check each element one by one if (arr [i] === key) return i ; } // Key not found return - 1 ; } const arr = [5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3]; const key = 3 ; const index = search (arr , key); console . log (index); [Expected Approach 1] Using Binary Search Twice - O(log n) Time and O(1) Space The main Idea is to first finds the index of the smallest element (pivot), which is also the number of rotations applied to the sorted array. Once the pivot is known, the array is split into two sorted subarrays. If the key is equal to the pivot element, its index is returned. If the pivot is at index 0, the entire array is already sorted, so a standard binary search is applied to the whole array. Otherwise, the key is compared with the first element: if it's greater than or equal, binary search is performed on the left half; if not, on the right half. C++ #include <iostream> #include <vector> using namespace std ; // An iterative binary search function int binarySearch (vector < int > & arr , int lo , int hi , int x) { while (lo <= hi) { int mid = lo + (hi - lo) / 2 ; if (arr [mid] == x) return mid ; if (arr [mid] < x) lo = mid + 1 ; else hi = mid - 1 ; } return -1 ; } // Function to return pivot (index of the smallest element) int findPivot (vector < int > & arr , int lo , int hi) { while (lo <= hi) { // The current subarray is already sorted, // the minimum is at the low index if (arr [lo] <= arr [hi]) return lo ; int mid = (lo + hi) / 2 ; // The right half is not sorted. So // the minimum element must be in the // right half. if (arr [mid] > arr [hi]) lo = mid + 1 ; //

The right half is sorted. Note that in // this case, we do not change high to mid - 1 // but keep it to mid. The mid element // itself can be the smallest else hi = mid ; } return lo ; } // Searches an element key in a pivoted // sorted array arr of size n int search (vector < int > & arr , int key) { int n = arr . size (); int pivot = findPivot (arr , 0 , n - 1); // 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 the minimum element is present at index // 0, then the whole array is sorted if (pivot == 0) return binarySearch (arr , 0 , n - 1 , key); if (arr [0] <= key) return binarySearch (arr , 0 , pivot - 1 , key); return binarySearch (arr , pivot + 1 , n - 1 , key); } int main () { vector < int > arr = { 5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3 }; int key = 3 ; cout << search (arr , key); return 0 ; } Java class GfG { // An iterative binary search function static int binarySearch (int [] arr , int lo , int hi , int x) { while (lo <= hi) { int mid = lo + (hi - lo) / 2 ; if (arr [mid] == x) return mid ; if (arr [mid] < x) lo = mid + 1 ; else hi = mid - 1 ; } return - 1 ; } // Function to return pivot (index of the smallest element) static int findPivot (int [] arr , int lo , int hi) { while (lo <= hi) { // The current subarray is already sorted, // the minimum is at the low index if (arr [lo] <= arr [hi]) return lo ; int mid = (lo + hi) / 2 ; // The right half is not sorted. So // the minimum element must be in the // right half if (arr [mid] > arr [hi]) lo = mid + 1 ; // The right half is sorted. Note that in // this case, we do not change high to mid - 1 // but keep it to mid. The mid element // itself can be the smallest else hi = mid ; } return lo ; } // Searches an element key in a pivoted // sorted array arr of size n static int search (int [] arr , int key) { int n = arr . length ; int pivot = findPivot (arr , 0 , n - 1); // 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 the minimum element is present at index // 0, then the whole array is sorted if (pivot == 0) return binarySearch (arr , 0 , n - 1 , key); if (arr [0] <= key) return binarySearch (arr , 0 , pivot - 1 , key); return binarySearch (arr , pivot + 1 , n - 1 , key); } public static void main (String [] args) { int [] arr = { 5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3 }; int key = 3 ; System . out . println (search (arr , key)); } } Python # An iterative binary search function def binarySearch (arr , lo , hi , x): while lo <= hi : mid = lo + (hi - lo) // 2 if arr [mid] == x : return mid if arr [mid] < x : lo = mid + 1 else : hi = mid - 1 return - 1 # Function to return pivot (index of the smallest element) def findPivot (arr , lo , hi): while lo <= hi : # The current subarray is already sorted, # the minimum is at the low index if arr [lo] <= arr [hi]: return lo mid = (lo + hi) // 2 # The right half is not sorted. So # the minimum element must be in the # right half. if arr [mid] > arr [hi]: lo = mid + 1 # The right half is sorted. Note that in # this case, we do not change high to mid - 1 # but keep it to mid. The mid element # itself can be the smallest else : hi = mid return lo # Searches an element key in a pivoted # sorted array arr of size n def search (arr , key): n = len (arr) pivot = findPivot (arr , 0 , n - 1) # 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 the minimum element is present at index # 0, then the whole array is sorted if pivot == 0 : return binarySearch (arr , 0 , n - 1 , key) if arr [0] <= key : return binarySearch (arr , 0 , pivot - 1 , key) return binarySearch (arr , pivot + 1 , n - 1 , key) if __name__ == "__main__" : arr = [5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3] key = 3 print (search (arr , key)) C# using System ; class GfG { // An iterative binary search function static int binarySearch (int [] arr , int lo , int hi , int x) { while (lo <= hi) { int mid = lo + (hi - lo) / 2 ; if (arr [mid] == x) return mid ; if (arr [mid] < x) lo = mid + 1 ; else hi = mid - 1 ; } return - 1 ; } // Function to return pivot (index of the smallest element) static int findPivot (int [] arr , int lo , int hi) { while (lo <= hi) { // The current subarray is already sorted, // the minimum is at the low index if (arr [lo] <= arr [hi]) return lo ; int mid = (lo + hi) / 2 ; // The right half is not sorted. So // the minimum element must be in the // right half. if (arr [mid] > arr [hi]) lo = mid + 1 ; // The right half is sorted. Note that in // this case, we do not change high to mid - 1 // but keep it to mid. The mid element // itself can be the smallest else hi = mid ; } return lo ; } // Searches an element key in a pivoted // sorted array arr of size n static int search (int [] arr , int key) { int n = arr . Length ; int pivot = findPivot (arr , 0 , n - 1); // 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 the minimum element is present at index // 0, then the whole array is sorted if (pivot == 0) return binarySearch (arr , 0 , n - 1 , key); if (arr [0] <= key) return binarySearch (arr , 0 , pivot - 1 , key); return binarySearch (arr , pivot + 1 , n - 1 , key); } static void Main (string [] args) { int [] arr = { 5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3 }; int key = 3 ; Console . WriteLine (search (arr , key)); } } JavaScript // An iterative binary search function function binarySearch (arr , lo , hi , x) { while (lo <= hi) { let mid = lo + Math . floor ((hi - lo) / 2) ; if (arr [mid] === x) return mid ; if (arr [mid] < x) lo = mid + 1 ; else hi = mid - 1 ; } return - 1 ; } // Function to return pivot (index of the smallest element) function findPivot (arr , lo , hi) { while (lo <= hi) { // The current subarray is already sorted, // the minimum is at the low index if (arr [lo] <= arr [hi]) return lo ; let mid = Math . floor ((lo + hi) / 2); // The right half is not sorted. So // the minimum element must be in the // right half. if (arr [mid] > arr [hi]) lo = mid + 1 ; // The right half is

sorted. Note that in // this case, we do not change high to mid - 1 // but keep it to mid. The mid element // itself can be the smallest else hi = mid ; } return lo ; } // Searches an element key in a pivoted // sorted array arr of size n function search (arr , key) { let n = arr . length ; let pivot = findPivot (arr , 0 , n - 1); // 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 the minimum element is present at index // 0, then the whole array is sorted if (pivot === 0) return binarySearch (arr , 0 , n - 1 , key); if (arr [0] <= key) return binarySearch (arr , 0 , pivot - 1 , key); return binarySearch (arr , pivot + 1 , n - 1 , key); } // Driver code let arr = [5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3]; let key = 3 ; console . log (search (arr , key)); Output 8 [Expected Approach 2] Using Single Binary Search - O(log n) Time and O(1) Space This approach applies a modified version of binary search directly to the entire rotated array. At every iteration, the middle element is checked against the key. If it's not the key, we determine whether the left half or right half is sorted by comparing values at arr[lo] and arr[mid]. If the left half is sorted and the key lies within its range, we adjust hi = mid - 1; otherwise, we shift lo = mid + 1. If the right half is sorted and the key lies within its range, we move lo = mid + 1; else, hi = mid - 1. C++ #include <iostream> #include <vector> using namespace std ; int search (vector < int >& arr , int key) { // Initialize two pointers, lo and hi, at the start // and end of the array int lo = 0 , hi = arr . size () - 1 ; while (lo <= hi) { int mid = lo + (hi - lo) / 2 ; // If key found, return the index if (arr [mid] == key) return mid ; // If Left half is sorted if (arr [mid] >= arr [lo]) { // If the key lies within this sorted half, // move the hi pointer to mid - 1 if (key >= arr [lo] && key < arr [mid]) hi = mid - 1 ; // Otherwise, move the lo pointer to mid + 1 else lo = mid + 1 ; } // If Right half is sorted else { // If the key lies within this sorted half, // move the lo pointer to mid + 1 if (key > arr [mid] && key <= arr [hi]) lo = mid + 1 ; // Otherwise, move the hi pointer to mid - 1 else hi = mid - 1 ; } } // Key not found return -1 ; } int main () { vector < int > arr = { 5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3 }; int key = 3 ; cout << search (arr , key) << endl ; return 0 ; } Java class GfG { static int search (int [] arr , int key) { // Initialize two pointers, lo and hi, at the start // and end of the array int lo = 0 , hi = arr . length - 1 ; while (lo <= hi) { int mid = lo + (hi - lo) / 2 ; // If key found, return the index if (arr [mid] == key) return mid ; // If Left half is sorted if (arr [mid] >= arr [lo]) { // If the key lies within this sorted half, // move the hi pointer to mid - 1 if (key >= arr [lo] && key < arr [mid]) hi = mid - 1 ; // Otherwise, move the lo pointer to mid + 1 else lo = mid + 1 ; } // If Right half is sorted else { // If the key lies within this sorted half, // move the lo pointer to mid + 1 if (key > arr [mid] && key <= arr [hi]) lo = mid + 1 ; // Otherwise, move the hi pointer to mid - 1 else hi = mid - 1 ; } } // Key not found return - 1 ; } public static void main (String [] args) { int [] arr = { 5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3 }; int key = 3 ; System . out . println (search (arr , key)); } } Python def search (arr , key): # Initialize two pointers, lo and hi, at the start # and end of the array lo = 0 hi = len (arr) - 1 while lo <= hi : mid = lo + (hi - lo) // 2 # If key found, return the index if arr [mid] == key : return mid # If Left half is sorted if arr [mid] >= arr [lo]: # If the key lies within this sorted half, # move the hi pointer to mid - 1 if key >= arr [lo] and key < arr [mid]: hi = mid - 1 # Otherwise, move the lo pointer to mid + 1 else : lo = mid + 1 # If Right half is sorted else : # If the key lies within this sorted half, # move the lo pointer to mid + 1 if key > arr [mid] and key <= arr [hi]: lo = mid + 1 # Otherwise, move the hi pointer to mid - 1 else : hi = mid - 1 # Key not found return - 1 if __name__ == "__main__" : arr = [5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3] key = 3 print (search (arr , key)) C# using System ; class GfG { static int search (int [] arr , int key) { // Initialize two pointers, lo and hi, at the start // and end of the array int lo = 0 , hi = arr . Length - 1 ; while (lo <= hi) { int mid = lo + (hi - lo) / 2 ; // If key found, return the index if (arr [mid] == key) return mid ; // If Left half is sorted if (arr [mid] >= arr [lo]) { // If the key lies within this sorted half, // move the hi pointer to mid - 1 if (key >= arr [lo] && key < arr [mid]) hi = mid - 1 ; // Otherwise, move the lo pointer to mid + 1 else lo = mid + 1 ; } // If Right half is sorted else { // If the key lies within this sorted half, // move the lo pointer to mid + 1 if (key > arr [mid] && key <= arr [hi]) lo = mid + 1 ; // Otherwise, move the hi pointer to mid - 1 else hi = mid - 1 ; } } // Key not found return - 1 ; } static void Main (string [] args) { int [] arr = { 5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3 }; int key = 3 ; Console . WriteLine (search (arr , key)); } } JavaScript function search (arr , key) { // Initialize two pointers, lo and hi, at the start // and end of the array let lo = 0 , hi = arr . length - 1 ; while (lo <= hi) { let mid = lo + Math . floor ((hi - lo) / 2); // If key found, return the index if (arr [mid] === key) return mid ; // If Left half is sorted if (arr [mid] >= arr [lo]) { // If the key lies within this sorted half, // move the hi pointer to mid - 1 if (key >= arr [lo] && key < arr [mid]) hi = mid - 1 ; // Otherwise, move the lo pointer to mid + 1 else lo = mid + 1 ; } // If Right half is sorted else { // If the key lies within this sorted half, // move the lo pointer to mid + 1 if (key > arr [mid] && key <= arr [hi]) lo = mid + 1 ; // Otherwise, move the hi pointer to mid - 1 else hi = mid - 1 ; } } // Key not found return - 1 ; } // Driver code let arr = [5 , 6 , 7 , 8 , 9 , 10 , 1 , 2 , 3]; let key = 3 ; console . log (search (arr , key)); Output 8 Comment Article Tags: Article Tags: Searching DSA Arrays Microsoft Amazon Adobe Flipkart

Samsung D-E-Shaw Snapdeal Paytm Hike BankBazaar SAP Labs MakeMyTrip Binary Search FactSet Times Internet rotation + 15 More