# Rotation Count in a Rotated Sorted array - GeeksforGeeks

**Source:** https://www.geeksforgeeks.org/find-rotation-count-rotated-sorted-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 Rotation Count in a Rotated Sorted array Last Updated : 8 Aug, 2025 Given a sorted array arr[] (in strictly increasing order) that has been right-rotated k times. A right rotation means the last element is moved to the first position, and the remaining elements are shifted one position to the right. Find the value of k the number of times the array was right-rotated from its originally sorted form. Examples: Input: arr[] = [15, 18, 2, 3, 6, 12] Output: 2 Explanation: Original sorted array = [2, 3, 6, 12, 15, 18] After 2 right rotations → [15, 18, 2, 3, 6, 12] Input: arr[] = [7, 9, 11, 12, 5] Output: 4 Explanation: Original sorted array = [5, 7, 9, 11, 12] After 4 right rotations → [7, 9, 11, 12, 5] Input: arr[] = [7, 9, 11, 12, 15] Output: 0 Explanation: Array is already sorted, so k = 0 Try it on GfG Practice Table of Content [Naive Approach] - O(n) Time and O(1) Space [Expected Approach] - O(log n) Time and O(1) Space [Naive Approach] - O(n) Time and O(1) Space The idea is that in a right-rotated sorted array, the smallest element marks the point of rotation. By scanning through the array to find this smallest element, its index directly gives the number of rotations performed. C++ #include <iostream> #include <vector> using namespace std ; int findKRotation ( vector < int >& arr ) { // Find index of minimum element int min = arr [ 0 ], minIndex = 0 ; for ( int i = 0 ; i < arr . size (); i ++ ) { if ( min > arr [ i ]) { min = arr [ i ]; minIndex = i ; } } return minIndex ; } int main () { vector < int > arr = { 15 , 18 , 2 , 3 , 6 , 12 } ; cout << findKRotation ( arr ) ; return 0 ; } Java class GfG { static int findKRotation ( int arr [] , int n ) { // We basically find index of minimum // element int min = arr [ 0 ] , minIndex = 0 ; for ( int i = 0 ; i < n ; i ++ ) { if ( min > arr [ i ] ) { min = arr [ i ] ; minIndex = i ; } } return minIndex ; } public static void main ( String [] args ) { int arr [] = { 15 , 18 , 2 , 3 , 6 , 12 }; int n = arr . length ; System . out . println ( findKRotation ( arr , n )); } } Python def findKRotation ( arr , n ): # We basically find index # of minimum element min = arr [ 0 ] minIndex = 0 for i in range ( 0 , n ): if ( min > arr [ i ]): min = arr [ i ] minIndex = i return minIndex if __name__ == "__main__" : arr = [ 15 , 18 , 2 , 3 , 6 , 12 ] n = len ( arr ) print ( findKRotation ( arr , n )) C# using System ; class GfG { static int findKRotation ( int [] arr , int n ) { // We basically find index of minimum // element int min = arr [ 0 ], minIndex = 0 ; for ( int i = 0 ; i < n ; i ++ ) { if ( min > arr [ i ]) { min = arr [ i ]; minIndex = i ; } } return minIndex ; } public static void Main () { int [] arr = { 15 , 18 , 2 , 3 , 6 , 12 }; int n = arr . Length ; Console . WriteLine ( findKRotation ( arr , n )); } } JavaScript function findKRotation ( arr , n ) { // We basically find index of minimum // element let min = arr [ 0 ], minIndex = 0 for ( let i = 0 ; i < n ; i ++ ) { if ( min > arr [ i ]) { min = arr [ i ]; minIndex = i ; } } return minIndex ; } // Driver Code let arr = [ 15 , 18 , 2 , 3 , 6 , 12 ]; let n = arr . length ; console . log ( findKRotation ( arr , n )); Output 2 [Expected Approach] - O(log n) Time and O(1) Space The idea is to use binary search to quickly locate the smallest element, which reveals the rotation count. In a rotated sorted array, the smallest element is the only one smaller than both its neighbors. At each step, we check if the current range is already sorted if it is, the first element is the smallest, and its index is the answer. Otherwise, we decide which half to explore by comparing the middle element with the last element: if arr[mid] is greater than arr[high], the smallest lies to the right; otherwise, it lies to the left. C++ #include <iostream> #include <vector> using namespace std ; int findKRotation ( vector < int > & arr ) { int low = 0 , high = arr . size () - 1 ; while ( low <= high ) { // If subarray is already sorted, // smallest is at low if ( arr [ low ] <= arr [ high ]) return low ; int mid = ( low + high ) / 2 ; // Minimum is in the right half if ( arr [ mid ] > arr [ high ]) low = mid + 1 ; // Minimum is in the left half (could be mid) else high = mid ; } return low ; } int main () { vector < int > arr = { 15 , 18 , 2 , 3 , 6

, 12 }; cout << findKRotation ( arr ); return 0 ; } Java public class GfG { static int findKRotation ( int [] arr ) { int low = 0 , high = arr . length - 1 ; while ( low <= high ) { // If subarray is already sorted, // smallest is at low if ( arr [ low ] <= arr [ high ] ) return low ; int mid = ( low + high ) / 2 ; // Minimum is in the right half if ( arr [ mid ] > arr [ high ] ) low = mid + 1 ; // Minimum is in the left half (could be mid) else high = mid ; } return low ; } public static void main ( String [] args ) { int [] arr = { 15 , 18 , 2 , 3 , 6 , 12 }; System . out . println ( findKRotation ( arr )); } } Python def findKRotation ( arr ): low = 0 high = len ( arr ) - 1 while low <= high : # If subarray is already sorted, # smallest is at low if arr [ low ] <= arr [ high ]: return low mid = ( low + high ) // 2 # Minimum is in the right half if arr [ mid ] > arr [ high ]: low = mid + 1 # Minimum is in the left half (could be mid) else : high = mid return low if __name__ == "__main__" : arr = [ 15 , 18 , 2 , 3 , 6 , 12 ] print ( findKRotation ( arr )) C# using System ; class GfG { static int findKRotation ( int [] arr ) { int low = 0 , high = arr . Length - 1 ; while ( low <= high ) { // If subarray is already sorted, // smallest is at low if ( arr [ low ] <= arr [ high ]) return low ; int mid = ( low + high ) / 2 ; // Minimum is in the right half if ( arr [ mid ] > arr [ high ]) low = mid + 1 ; // Minimum is in the left half (could be mid) else high = mid ; } return low ; } static void Main () { int [] arr = { 15 , 18 , 2 , 3 , 6 , 12 }; Console . WriteLine ( findKRotation ( arr )); } } JavaScript function findKRotation ( arr ) { let low = 0 , high = arr . length - 1 ; while ( low <= high ) { // If subarray is already sorted, // smallest is at low if ( arr [ low ] <= arr [ high ]) return low ; let mid = Math . floor (( low + high ) / 2 ); // Minimum is in the right half if ( arr [ mid ] > arr [ high ]) low = mid + 1 ; // Minimum is in the left half (could be mid) else high = mid ; } return low ; } // Driver Code const arr = [ 15 , 18 , 2 , 3 , 6 , 12 ] ; console . log ( findKRotation ( arr )) ; Output 2 Comment Article Tags: Article Tags: Divide and Conquer DSA Arrays Amazon Binary Search ABCO rotation + 3 More