Minimum in a Sorted and Rotated Array - GeeksforGeeks

Source: https://www.geeksforgeeks.org/find-minimum-element-in-a-sorted-and-rotated-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 Minimum in a Sorted and Rotated Array Last Updated : 23 Jul, 2025 Given a sorted array of distinct elements arr[] of size n that is rotated at some unknown point, the task is to find the minimum element in it. Examples: Input: arr[] = [5, 6, 1, 2, 3, 4] Output: 1 Explanation: 1 is the minimum element present in the array. Input: arr[] = [3, 1, 2] Output: 1 Explanation : 1 is the minimum element present in the array. Input: arr[] = [4, 2, 3] Output: 2 Explanation: 2 is the only minimum element in the array. Try it on GfG Practice Table of Content [Naive Approach] Linear Search - O(n) Time and O(1) Space [Expected Approach] Binary Search - O(log n) Time and O(1) Space [Naive Approach] Linear Search - O(n) Time and O(1) Space A simple solution is to use linear search to traverse the complete array and find a minimum . C++ // C++ program to find minimum element in a // sorted rotated array using linear search #include <iostream> #include <vector> using namespace std ; int findMin (vector < int >& arr) { int res = arr [0]; // Traverse over arr[] to find minimum element for (int i = 1 ; i < arr . size (); i ++) res = min (res , arr [i]); return res ; } int main () { vector < int > arr = { 5 , 6 , 1 , 2 , 3 , 4 }; int n = arr . size (); cout << findMin (arr) << endl ; return 0 ; } C // C program to find minimum element in a // sorted rotated array using linear search #include <stdio.h> int findMin (int arr [], int n) { int res = arr [0]; // Traverse over arr[] to find minimum element for (int i = 1 ; i < n ; i ++) { if (arr [i] < res) { res = arr [i]; } } return res ; } int main () { int arr [] = { 5 , 6 , 1 , 2 , 3 , 4 }; int n = sizeof (arr) / sizeof (arr [0]); printf ("%d \n " , findMin (arr , n)); return 0 ; } Java // Java program to find minimum element in a // sorted rotated array using linear search import java.util.* ; class GfG { static int findMin (int [] arr) { int res = arr [0] ; // Traverse over arr[] to find minimum element for (int i = 1 ; i < arr . length ; i ++) { res = Math . min (res , arr [i]); } return res ; } public static void main (String [] args) { int [] arr = { 5 , 6 , 1 , 2 , 3 , 4 }; System . out . println (findMin (arr)); } } Python # Python program to find minimum element in a # sorted rotated array using linear search def findMin (arr): res = arr [0] # Traverse over arr[] to find minimum element for i in range (1 , len (arr)): res = min (res , arr [i]) return res if __name__ == "__main__" : arr = [5 , 6 , 1 , 2 , 3 , 4] print (findMin (arr)) C# // C# program to find minimum element in a // sorted rotated array using linear search using System ; using System.Collections.Generic ; class GfG { static int FindMin (int [] arr) { int res = arr [0]; // Traverse over arr[] to find minimum element for (int i = 1 ; i < arr . Length ; i ++) { res = Math . Min (res , arr [i]); } return res ; } static void Main () { int [] arr = { 5 , 6 , 1 , 2 , 3 , 4 }; Console . WriteLine (FindMin (arr)); } } JavaScript // JavaScript program to find minimum element in a // sorted rotated array using linear search function findMin (arr) { let res = arr [0]; // Traverse over arr[] to find minimum element for (let i = 1 ; i < arr . length ; i ++) { res = Math . min (res , arr [i]); } return res ; } // Driver code const arr = [5 , 6 , 1 , 2 , 3 , 4]; console . log (findMin (arr)); Output 1 [Expected Approach] Binary Search - O(log n) Time and O(1) Space We can optimize the minimum element searching by using Binary Search where we find the mid element and then decide whether to stop or to go to left half or right half: If arr[mid] > arr[high] , it means arr[low ... mid] is sorted and we need to search in the right half. So we change low = mid + 1 . If arr[mid] <= arr[high] , it means arr[mid ... high] is sorted and we need to search in the left half. So we change high = mid. (Note : Current mid might be the minimum element). How do we terminate the search? One way could be to check if the mid is smaller than both of its adjacent, then we return mid . This would require a lot of condition checks like if adjacent indexes are valid or not and then comparing mid with both. We use an

interesting fact here: If arr[low] < arr[high] , then the current subarray is sorted, So we return arr[low] . C++ // C++ program to find minimum element in a // sorted and rotated array using binary search #include <iostream> #include <vector> using namespace std ; int findMin (vector < int > & arr) { int lo = 0 , hi = arr . size () - 1 ; while (lo < hi) { // The current subarray is already sorted, // the minimum is at the low index if (arr [lo] < arr [hi]) return arr [lo]; // We reach here when we have at least // two elements and the current subarray // is rotated 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. As the mid element // itself can be the smallest else hi = mid ; } return arr [lo]; } int main () { vector < int > arr = { 5 , 6 , 1 , 2 , 3 , 4 }; cout << findMin (arr) << endl ; return 0 ; } C // C program to find minimum element in a // sorted and rotated array using binary search #include <stdio.h> int findMin (int arr [], int n) { int lo = 0 , hi = n - 1 ; while (lo < hi) { // The current subarray is already sorted, // the minimum is at the low index if (arr [lo] < arr [hi]) return arr [lo]; // We reach here when we have at least // two elements and the current subarray // is rotated 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. As the mid element // itself can be the smallest else hi = mid ; } return arr [lo]; } int main () { int arr [] = { 5 , 6 , 1 , 2 , 3 , 4 }; int n = sizeof (arr) / sizeof (arr [0]); printf ("%d \n " , findMin (arr , n)); return 0 ; } Java // Java program to find minimum element in a // sorted and rotated array using binary search import java.util.* ; class GfG { static int findMin (int [] arr) { int lo = 0 , hi = arr . length - 1 ; while (lo < hi) { // The current subarray is already sorted, // the minimum is at the low index if (arr [lo] < arr [hi]) return arr [lo] ; // We reach here when we have at least // two elements and the current subarray // is rotated 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. As the mid element // itself can be the smallest else hi = mid ; } return arr [lo] ; } public static void main (String [] args) { int [] arr = { 5 , 6 , 1 , 2 , 3 , 4 }; System . out . println (findMin (arr)); } } Python # Python program to find minimum element in a # sorted and rotated array using binary search def findMin (arr): lo , hi = 0 , len (arr) - 1 while lo < hi : # The current subarray is already sorted, # the minimum is at the low index if arr [lo] < arr [hi]: return arr [lo] # We reach here when we have at least # two elements and the current subarray # is rotated 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. As the mid element # itself can be the smallest else : hi = mid return arr [lo] if __name__ == "__main__" : arr = [5 , 6 , 1 , 2 , 3 , 4] print (findMin (arr)) C# // C# program to find minimum element in a // sorted and rotated array using binary search using System ; class GfG { static int findMin (int [] arr) { int lo = 0 , hi = arr . Length - 1 ; while (lo < hi) { // The current subarray is already sorted, // the minimum is at the low index if (arr [lo] < arr [hi]) return arr [lo]; // We reach here when we have at least // two elements and the current subarray // is rotated 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. As the mid element // itself can be the smallest else hi = mid ; } return arr [lo]; } static void Main () { int [] arr = { 5 , 6 , 1 , 2 , 3 , 4 }; Console . WriteLine (findMin (arr)); } } JavaScript // JavaScript program to find minimum element in a // sorted and rotated array using binary search function findMin (arr) { let lo = 0 , hi = arr . length - 1 ; while (lo < hi) { // The current subarray is already sorted, // the minimum is at the low index if (arr [lo] < arr [hi]) return arr [lo]; // We reach here when we have at least // two elements and the current subarray // is rotated const 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. As the mid element // itself can be the smallest else hi = mid ; } return arr [lo]; } // Driver Code const arr = [5 , 6 , 1 , 2 , 3 , 4]; console . log (findMin (arr)); Output 1 Comment Article Tags: Article Tags: Divide and Conquer Searching DSA Arrays Microsoft Amazon Adobe Morgan Stanley Samsung Snapdeal Binary Search Times Internet + 8 More