Median of two sorted arrays of same size - GeeksforGeeks

Source: https://www.geeksforgeeks.org/median-of-two-sorted-arrays/

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 Median of two sorted arrays of same size Last Updated : 29 Oct, 2025 Given 2 sorted arrays a[] and b[] , each of size n , the task is to find the median of the array obtained after merging a[] and b[]. Note: Since the size of the merged array will always be even, the median will be the average of the middle two numbers. Input : a[] = [1, 12, 15, 26, 38], b[] = [2, 13, 17, 30, 45] Output : 16 Explanation : The merged sorted array is [1, 2, 12, 13, 15, 17, 26, 30, 38, 45]. The middle two elements are 15 and 17, so median = (15 + 17)/2 = 16 Input : a[] = [10], b[] = [21] Output : 15.5 Explanation : The merged sorted array is [10, 21]. The middle two elements are 10 and 21, so median = (10 + 21)/2 = 15.5 Try it on GfG Practice Table of Content [Naive Approach] Using Sorting - O(n * logn) Time and O(n) Space [Better Approach] Using Merge of Merge Sort - O(n) Time and O(1) Space [Expected Approach] Using Binary Search - O(log n) Time and O(1) Space [Naive Approach] Using Sorting - O(n * logn) Time and O(n) Space The idea is to concatenate both the arrays into a new array, sort the new array and return the middle of the new sorted array. Illustration: a[] = [1, 12, 15, 26, 38], b[] = [2, 13, 17, 30, 45] After concatenating them in a third array : c[] = [1, 12, 15, 26, 38, 2, 13, 17, 30, 45] Sort c[] = [1, 2, 12, 13, 15 , 17 , 26, 30, 38, 45] So the median is the average of two middle elements: (15 + 17) / 2 = 16 C++ // C++ Code to find Median of two Sorted Arrays of // Same Size using Sorting #include <bits/stdc++.h> using namespace std ; // Function to find the median of two sorted arrays of equal size double getMedian (vector < int >& a , vector < int >& b) { // Concatenate vector < int > c (a . begin (), a . end ()); c . insert (c . end (), b . begin (), b . end ()); // Sort the concatenated array sort (c . begin (), c . end ()); // Calculate and return the median int n = c . size (); int mid1 = n / 2 ; int mid2 = mid1 - 1 ; return (c [mid1] + c [mid2]) / 2.0 ; } int main () { vector < int > a = { 1 , 12 , 15 , 26 , 38 }; vector < int > b = { 2 , 13 , 17 , 30 , 45 }; cout << getMedian (a , b) << endl ; return 0 ; } C // C Code to find Median of two Sorted Arrays of // Same Size using Sorting #include <stdio.h> // Function to compare two integers for qsort int compare (const void * a , const void * b) { return (* (int *) a - * (int *) b); } // Function to find the median of two sorted arrays of equal size double getMedian (int a [], int size1 , int b [], int size2) { // Concatenate arrays int totalSize = size1 + size2 ; int c [size1 + size2]; // Copy elements from a and b to c for (int i = 0 ; i < size1 ; i ++) c [i] = a [i]; for (int i = 0 ; i < size2 ; i ++) c [size1 + i] = b [i]; // Sort the concatenated array qsort (c , totalSize , sizeof (int), compare); // Calculate and return the median int mid1 = totalSize / 2 ; int mid2 = mid1 - 1 ; return (c [mid1] + c [mid2]) / 2.0 ; } int main () { int a [] = { 1 , 12 , 15 , 26 , 38 }; int b [] = { 2 , 13 , 17 , 30 , 45 }; int size1 = sizeof (a) / sizeof (a [0]); int size2 = sizeof (b) / sizeof (b [0]); printf ("%f" , getMedian (a , size1 , b , size2)); return 0 ; } Java // Java Code to find Median of two Sorted Arrays of // Same Size using Sorting import java.util.Arrays ; class GfG { // Function to find the median of two sorted arrays of equal size static double getMedian (int [] a , int [] b) { // Concatenate the two arrays int [] c = new int [a . length + b . length] ; System . arraycopy (a , 0 , c , 0 , a . length); System . arraycopy (b , 0 , c , a . length , b . length); // Sort the concatenated array Arrays . sort (c); // Calculate and return the median int n = c . length ; int mid1 = n / 2 ; int mid2 = mid1 - 1 ; return (c [mid1] + c [mid2]) / 2.0 ; } public static void main (String [] args) { int [] a = { 1 , 12 , 15 , 26 , 38 }; int [] b = { 2 , 13 , 17 , 30 , 45 }; System . out . println (getMedian (a , b)); } } Python # Python Code to find Median of two Sorted Arrays of # Same Size using Sorting # Function to find the median of two sorted arrays of # equal size def getMedian (a , b): # Concatenate the two lists c = a + b

Sort the concatenated list c . sort () # Calculate and return the median n = len (c) mid1 = n // 2 mid2 = mid1 - 1 return (c [mid1] + c [mid2]) / 2.0 # Example usage a = [1 , 12 , 15 , 26 , 38] b = [2 , 13 , 17 , 30 , 45] print (getMedian (a , b)) C# // C# Code to find Median of two Sorted Arrays of // Same Size using Sorting using System ; using System.Linq ; class GfG { // Function to find the median of two sorted arrays // of equal size static double getMedian (int [] a , int [] b) { // Concatenate the two arrays int [] c = a . Concat (b). ToArray (); // Sort the concatenated array Array . Sort (c); // Calculate and return the median int n = c . Length ; int mid1 = n / 2 ; int mid2 = mid1 - 1 ; return (c [mid1] + c [mid2]) / 2.0 ; } static void Main () { int [] a = { 1 , 12 , 15 , 26 , 38 }; int [] b = { 2 , 13 , 17 , 30 , 45 }; Console . WriteLine (getMedian (a , b)); } } JavaScript // JavaScript Code to find Median of two Sorted Arrays of // Same Size using Sorting // Function to find the median of two sorted arrays of equal size function getMedian (a , b) { // Concatenate the two arrays let c = a . concat (b); // Sort the concatenated array c . sort ((a , b) => a - b); // Calculate and return the median let n = c . length ; let mid1 = Math . floor (n / 2); let mid2 = mid1 - 1 ; return (c [mid1] + c [mid2]) / 2 ; } // Example usage let a = [1 , 12 , 15 , 26 , 38]; let b = [2 , 13 , 17 , 30 , 45]; console . log (getMedian (a , b)); Output 16 Time Complexity : O((2n) * log(2n)), where n is the size of array a[] and b[]. Auxiliary Space : O(2n), because we are creating a new merged array of size 2n. [Better Approach] Using Merge of Merge Sort - O(n) Time and O(1) Space The given arrays are sorted, so merge the sorted arrays in an efficient way and keep the count of elements processed so far. So when we reach half of the total, print the median. The median will be the average of elements at index (n - 1) and n in the array obtained after merging both the arrays. C++ // C++ Code to find Median of two Sorted Arrays of // Same Size using Merge of Merge Sort #include <bits/stdc++.h> using namespace std ; // Function to find the median of two sorted arrays // of equal size float getMedian (vector < int >& a , vector < int >& b) { int n = a . size (); int i = 0 , j = 0 ; int count ; // m1 to store element at index n of merged array // m2 to store element at index (n - 1) of merged array int m1 = -1 , m2 = -1 ; // Loop till n for (count = 0 ; count <= n ; count ++) { m2 = m1 ; // If both the arrays have remaining elements if (i < n && j < n) { if (a [i] > b [j]) m1 = b [j ++]; else m1 = a [i ++]; } // If only a has remaining elements else if (i < n) m1 = a [i ++]; // If only b has remaining elements else m1 = b [j ++]; } return (m1 + m2) / 2.0 ; } int main () { vector < int > a = { 1 , 12 , 15 , 26 , 38 }; vector < int > b = { 2 , 13 , 17 , 30 , 45 }; cout << getMedian (a , b) << endl ; return 0 ; } C // C Code to find Median of two Sorted Arrays of // Same Size using Merge of Merge Sort #include <stdio.h> // Function to find the median of two sorted arrays of equal size float getMedian (int a [], int b [], int n) { int i = 0 , j = 0 ; int count ; int m1 = -1 , m2 = -1 ; // Loop till n for (count = 0 ; count <= n ; count ++) { m2 = m1 ; // If both the arrays have remaining elements if (i < n && j < n) { if (a [i] > b [j]) m1 = b [j ++]; else m1 = a [i ++]; } // If only a has remaining elements else if (i < n) m1 = a [i ++]; // If only b has remaining elements else m1 = b [j ++]; } return (m1 + m2) / 2.0 ; } int main () { int a [] = { 1 , 12 , 15 , 26 , 38 }; int b [] = { 2 , 13 , 17 , 30 , 45 }; int n = sizeof (a) / sizeof (a [0]); printf ("%f" , getMedian (a , b , n)); return 0 ; } Java // Java Code to find Median of two Sorted Arrays of // Same Size using Merge of Merge Sort import java.util.Arrays ; class GfG { // Function to find the median of two sorted arrays of equal size static double getMedian (int [] a , int [] b) { int n = a . length ; int i = 0 , j = 0 ; int count ; // m1 to store element at index n of merged array // m2 to store element at index (n - 1) of merged array int m1 = - 1 , m2 = - 1 ; // Loop till n for (count = 0 ; count <= n ; count ++) { m2 = m1 ; // If both the arrays have remaining elements if (i < n && j < n) { if (a [i] > b [j]) m1 = b [j ++] ; else m1 = a [i ++] ; } // If only a has remaining elements else if (i < n) m1 = a [i ++] ; // If only b has remaining elements else m1 = b [j ++] ; } return (m1 + m2) / 2.0 ; } public static void main (String [] args) { int [] a = { 1 , 12 , 15 , 26 , 38 }; int [] b = { 2 , 13 , 17 , 30 , 45 }; System . out . println (getMedian (a , b)); } } Python # Python Code to find Median of two Sorted Arrays of # Same Size using Merge of Merge Sort # Function to find the median of two sorted arrays # of equal size def getMedian (a , b): n = len (a) i , j = 0 , 0 count = 0 # m1 to store element at index n of merged array # m2 to store element at index (n - 1) of merged array m1 , m2 = - 1 , - 1 # Loop till n for count in range (n + 1): m2 = m1 # If both the arrays have remaining elements if i < n and j < n : if a [i] > b [j]: m1 = b [j] j += 1 else : m1 = a [i] i += 1 # If only a has remaining elements elif i < n : m1 = a [i] i += 1 # If only b has remaining elements else : m1 = b [j] j += 1 return (m1 + m2) / 2.0 a = [1 , 12 , 15 , 26 , 38] b = [2 , 13 , 17 , 30 , 45] print (getMedian (a , b)) C# // C# Code to find Median of two Sorted Arrays of // Same Size using Merge of Merge Sort using System ; using System.Linq ; class GfG { // Function to find the median of two sorted arrays // of equal size static float getMedian (int [] a , int [] b) { int n = a . Length ; int i = 0 , j = 0 ; int count ; // m1 to store element at index n of merged array // m2 to store element at index (n - 1) of merged array int m1 = - 1 , m2 = - 1 ; // Loop till n for (count = 0 ; count <= n ; count ++) { m2 = m1 ; // If both the arrays have remaining elements if (i < n && j < n) { if (a [i] > b [j]) m1 = b [j ++]; else

m1 = a [i ++]; } // If only a has remaining elements else if (i < n) m1 = a [i ++]; // If only b has remaining elements else m1 = b [j ++]; } return (m1 + m2) / 2.0f ; } static void Main () { int [] a = { 1 , 12 , 15 , 26 , 38 }; int [] b = { 2 , 13 , 17 , 30 , 45 }; Console . WriteLine (getMedian (a , b)); } } JavaScript // JavaScript Code to find Median of two Sorted Arrays of // Same Size using Merge of Merge Sort // Function to find the median of two sorted arrays // of equal size function getMedian (a , b) { let n = a . length ; let i = 0 , j = 0 ; let count ; // m1 to store element at index n of merged array // m2 to store element at index (n - 1) of merged array let m1 = - 1 , m2 = - 1 ; // Loop till n for (count = 0 ; count <= n ; count ++) { m2 = m1 ; // If both the arrays have remaining elements if (i < n && j < n) { if (a [i] > b [j]) m1 = b [j ++]; else m1 = a [i ++]; } // If only a has remaining elements else if (i < n) m1 = a [i ++]; // If only b has remaining elements else m1 = b [j ++]; } return (m1 + m2) / 2.0 ; } let a = [1 , 12 , 15 , 26 , 38]; let b = [2 , 13 , 17 , 30 , 45]; console . log (getMedian (a , b)); Output 16 Time Complexity: O(n), where n is the size of array a[] and b[] . Auxiliary Space: O(1) [Expected Approach] Using Binary Search - O(log n) Time and O(1) Space To find the median of the two sorted arrays, a[] and b[] of size n , we need the average of two middle elements of merged sorted array. So, if we divide the merged array into two halves, then the median will be (last element in first half + first element in second half) / 2 . The idea is to use Binary Search to find the valid partition in a[] say mid1 , such that all elements of a[0...mid1 - 1] will lie in the first half of the merged sorted array. Since, we know that first half of the merged sorted array will have total n elements, the remaining mid2 = (n - mid1) elements will be from b[]. In other words, the first half of the merged sorted array will have all the elements in a[0...mid1 - 1] and b[0...mid2 - 1] . How to check if the partition mid1 and mid2 is valid or not? For mid1 and mid2 to be valid, we need to check for the following conditions: All elements in a[0...mid1 - 1] should be less than or equal to all elements in b[mid2...n - 1]. Since both the subarrays are sorted individually, we can check a[mid1 - 1] should be less than or equal to b[mid2]. All elements in b[0...mid2 - 1] should be less than or equal to all elements in a[mid1...n - 1]. Since both the subarrays are sorted individually, we can check b[mid2 - 1] should be less than or equal to a[mid1]. For simplicity, take the element to the left of partition mid1 as l1, so l1 = a[mid1 - 1] and element to the right of partition mid1 as r1, so r1 = a[mid1] . Similarly, take the element to the left of mid2 as l2, so l2 = b[mid2 - 1] and element to the right of mid2 as r2, so r2 = b[mid2] . So, the above conditions can be simplified as l1 <= r2 and l2 <= r1 . If the partition is not valid, we can have two cases: If l1 > r2, this means that we have taken extra elements from a[], so take less elements by moving high = mid - 1 . If l2 > r1, this means that we have taken less elements from a[], so take more elements by moving low = mid + 1 . Illustration: Below is the implementation of the above approach: C++ // C++ Program to find the median of merged sorted // array using Binary Search #include <bits/stdc++.h> using namespace std ; // function to find median of merged sorted array double getMedian (vector < int > & a , vector < int > & b) { int n = a . size (); // We can take [0...n] number of elements from a[] int low = 0 , high = n ; while (low <= high) { // Take mid1 elements from a int mid1 = (low + high) / 2 ; // Take mid2 elements from b int mid2 = n - mid1 ; // Find elements to the left and right of partition in a int l1 = (mid1 == 0 ? INT_MIN : a [mid1 - 1]); int r1 = (mid1 == n ? INT_MAX : a [mid1]); // Find elements to the left and right of partition in b int l2 = (mid2 == 0 ? INT_MIN : b [mid2 - 1]); int r2 = (mid2 == n ? INT_MAX : b [mid2]); // If it is a valid partition if (l1 <= r2 && l2 <= r1) return (max (l1 , l2) + min (r1 , r2)) / 2.0 ; // If we need to take lesser elements from a if (l1 > r2) high = mid1 - 1 ; // If we need to take more elements from a else low = mid1 + 1 ; } return 0 ; } int main () { vector < int > a = { 1 , 12 , 15 , 26 , 38 }; vector < int > b = { 2 , 13 , 17 , 30 , 45 }; cout << getMedian (a , b) << endl ; return 0 ; } C // C Program to find the median of merged sorted // array using Binary Search #include <stdio.h> #include <limits.h> // Function to find median of merged sorted array double getMedian (int a [], int b [], int n) { int low = 0 , high = n ; while (low <= high) { // Take mid1 elements from a int mid1 = (low + high) / 2 ; // Take mid2 elements from b int mid2 = n - mid1 ; // Find elements to the left and right of partition in a int l1 = (mid1 == 0 ? INT_MIN : a [mid1 - 1]); int r1 = (mid1 == n ? INT_MAX : a [mid1]); // Find elements to the left and right of partition in b int l2 = (mid2 == 0 ? INT_MIN : b [mid2 - 1]); int r2 = (mid2 == n ? INT_MAX : b [mid2]); // If it is a valid partition if (l1 <= r2 && l2 <= r1) return (fmax (l1 , l2) + fmin (r1 , r2)) / 2.0 ; // If we need to take fewer elements from a if (l1 > r2) high = mid1 - 1 ; // If we need to take more elements from a else low = mid1 + 1 ; } return 0.0 ; } int main () { int a [] = { 1 , 12 , 15 , 26 , 38 }; int b [] = { 2 , 13 , 17 , 30 , 45 }; int n = sizeof (a) / sizeof (a [0]); printf ("%f" , getMedian (a , b , n)); return 0 ; } Java // Java Program to find the median of merged sorted // array using Binary Search import java.util.Arrays ; class GfG { // Function to find median of merged sorted array static double getMedian (int [] a , int [] b) { int n = a . length ; // We can take [0...n] number of elements from a[] int low = 0 , high = n ; while (low <= high) { // Take mid1 elements from a int mid1 = (low + high) / 2 ; // Take mid2 elements from b int mid2 = n -

mid1 ; // Find elements to the left and right of partition in a int l1 = (mid1 == 0 ? Integer . MIN_VALUE : a [mid1 - 1]); int r1 = (mid1 == n ? Integer . MAX_VALUE : a [mid1]); // Find elements to the left and right of partition in b int l2 = (mid2 == 0 ? Integer . MIN_VALUE : b [mid2 - 1]); int r2 = (mid2 == n ? Integer . MAX_VALUE : b [mid2]); // If it is a valid partition if (l1 <= r2 && l2 <= r1) return (Math . max (l1 , l2) + Math . min (r1 , r2)) / 2.0 ; // If we need to take fewer elements from a if (l1 > r2) high = mid1 - 1 ; // If we need to take more elements from a else low = mid1 + 1 ; } return 0 ; } public static void main (String [] args) { int [] a = { 1 , 12 , 15 , 26 , 38 }; int [] b = { 2 , 13 , 17 , 30 , 45 }; System . out . println (getMedian (a , b)); } } Python # Python Program to find the median of merged sorted # array using Binary Search # Function to find median of merged sorted array def getMedian (a , b): n = len (a) # We can take [0...n] number of elements from a[] low , high = 0 , n while low <= high : # Take mid1 elements from a mid1 = (low + high) // 2 # Take mid2 elements from b mid2 = n - mid1 # Find elements to the left and right of partition in a l1 = float ('-inf') if mid1 == 0 else a [mid1 - 1] r1 = float ('inf') if mid1 == n else a [mid1] # Find elements to the left and right of partition in b l2 = float ('-inf') if mid2 == 0 else b [mid2 - 1] r2 = float ('inf') if mid2 == n else b [mid2] # If it is a valid partition if l1 <= r2 and l2 <= r1 : return (max (l1 , l2) + min (r1 , r2)) / 2.0 # If we need to take fewer elements from a if l1 > r2 : high = mid1 - 1 # If we need to take more elements from a else : low = mid1 + 1 return 0 a = [1 , 12 , 15 , 26 , 38] b = [2 , 13 , 17 , 30 , 45] print (getMedian (a , b)) C# // C# Program to find the median of merged sorted // array using Binary Search using System ; class GfG { // Function to find median of merged sorted array static double getMedian (int [] a , int [] b) { int n = a . Length ; // We can take [0...n] number of elements from a[] int low = 0 , high = n ; while (low <= high) { // Take mid1 elements from a int mid1 = (low + high) / 2 ; // Take mid2 elements from b int mid2 = n - mid1 ; // Find elements to the left and right of partition in a int l1 = (mid1 == 0) ? int . MinValue : a [mid1 - 1]; int r1 = (mid1 == n) ? int . MaxValue : a [mid1]; // Find elements to the left and right of partition in b int l2 = (mid2 == 0) ? int . MinValue : b [mid2 - 1]; int r2 = (mid2 == n) ? int . MaxValue : b [mid2]; // If it is a valid partition if (l1 <= r2 && l2 <= r1) return (Math . Max (l1 , l2) + Math . Min (r1 , r2)) / 2.0 ; // If we need to take fewer elements from a if (l1 > r2) high = mid1 - 1 ; // If we need to take more elements from a else low = mid1 + 1 ; } return 0 ; } static void Main () { int [] a = { 1 , 12 , 15 , 26 , 38 }; int [] b = { 2 , 13 , 17 , 30 , 45 }; Console . WriteLine (getMedian (a , b)); } } JavaScript // JavaScript Program to find the median of merged sorted // array using Binary Search // Function to find median of merged sorted array function getMedian (a , b) { const n = a . length ; let low = 0 , high = n ; while (low <= high) { // Take mid1 elements from a const mid1 = Math . floor ((low + high) / 2); // Take mid2 elements from b const mid2 = n - mid1 ; // Find elements to the left and right of partition in a const l1 = (mid1 === 0) ? - Infinity : a [mid1 - 1]; const r1 = (mid1 === n) ? Infinity : a [mid1]; // Find elements to the left and right of partition in b const l2 = (mid2 === 0) ? - Infinity : b [mid2 - 1]; const r2 = (mid2 === n) ? Infinity : b [mid2]; // If it is a valid partition if (l1 <= r2 && l2 <= r1) { return (Math . max (l1 , l2) + Math . min (r1 , r2)) / 2.0 ; } // If we need to take fewer elements from a if (l1 > r2) { high = mid1 - 1 ; } // If we need to take more elements from a else { low = mid1 + 1 ; } } // Return 0 if no median found (this should not happen with valid input) return 0 ; } // Example usage const a = [1 , 12 , 15 , 26 , 38]; const b = [2 , 13 , 17 , 30 , 45]; console . log (getMedian (a , b)); Output 16 Time Complexity : O(log n), where n is the size of input array. Auxiliary Space : O(1) Related Articles: Median of two sorted arrays of different sizes K-th Element of Merged Two Sorted Arrays Comment Article Tags: Article Tags: Divide and Conquer Searching Mathematical DSA Arrays Amazon Samsung D-E-Shaw Accolite FactSet statistical-algorithms median-finding + 8 More