GeeksforGeeks

A computer science portal for geeks

GeeksQuiz

Median of two sorted arrays

Question: There are 2 sorted arrays A and B of size n each. Write an algorithm to find the median of the array obtained after merging the above 2 arrays(i.e. array of length 2n). The complexity should be O(log(n))

Median: In probability theory and statistics, a median is described as the number separating the higher half of a sample, a population, or a probability distribution, from the lower half.
The median of a finite list of numbers can be found by arranging all the numbers from lowest value to highest value and picking the middle one.

For getting the median of input array { 12, 11, 15, 10, 20 }, first sort the array. We get { 10, 11, 12, 15, 20 } after sorting. Median is the middle element of the sorted array which is 12.

There are different conventions to take median of an array with even number of elements, one can take the mean of the two middle values, or first middle value, or second middle value.

Let us see different methods to get the median of two sorted arrays of size n each. Since size of the set for which we are looking for median is even (2n), we are taking average of middle two numbers in all below solutions.

Method 1 (Simply count while Merging)
Use merge procedure of merge sort. Keep track of count while comparing elements of two arrays. If count becomes n(For 2n elements), we have reached the median. Take the average of the elements at indexes n-1 and n in the merged array. See the below implementation.

Implementation:

```
#include <stdio.h>

/* This function returns median of ar1[] and ar2[].
   Assumptions in this function:
   Both ar1[] and ar2[] are sorted arrays
   Both have n elements */
int getMedian(int ar1[], int ar2[], int n)
{
    int i = 0;  /* Current index of i/p array ar1[] */
    int j = 0; /* Current index of i/p array ar2[] */
    int count;
    int m1 = -1, m2 = -1;

    /* Since there are 2n elements, median will be average
     of elements at index n-1 and n in the array obtained after
     merging ar1 and ar2 */
    for (count = 0; count <= n; count++)
    {
        /*Below is to handle case where all elements of ar1[] are
          smaller than smallest(or first) element of ar2[]*/
        if (i == n)
        {
            m1 = m2;
            m2 = ar2[0];
            break;
        }

        /*Below is to handle case where all elements of ar2[] are
          smaller than smallest(or first) element of ar1[]*/
        else if (j == n)
        {
            m1 = m2;
            m2 = ar1[0];
            break;
        }

        if (ar1[i] < ar2[j])
        {
            m1 = m2;  /* Store the prev median */
            m2 = ar1[i];
            i++;
        }
        else
        {
```

```
            m1 = m2;  /* Store the prev median */
            m2 = ar2[j];
            j++;
        }
    }

    return (m1 + m2)/2;
}

/* Driver program to test above function */
int main()
{
    int ar1[] = {1, 12, 15, 26, 38};
    int ar2[] = {2, 13, 17, 30, 45};

    int n1 = sizeof(ar1)/sizeof(ar1[0]);
    int n2 = sizeof(ar2)/sizeof(ar2[0]);
    if (n1 == n2)
        printf("Median is %d", getMedian(ar1, ar2, n1));
    else
        printf("Doesn't work for arrays of unequal size");
    getchar();
    return 0;
}
```

Time Complexity: O(n)

Method 2 (By comparing the medians of two arrays)
This method works by first getting medians of the two sorted arrays and then comparing them.

Let ar1 and ar2 be the input arrays.

Algorithm:

```
1) Calculate the medians m1 and m2 of the input arrays ar1[]
   and ar2[] respectively.
2) If m1 and m2 both are equal then we are done.
     return m1 (or m2)
3) If m1 is greater than m2, then median is present in one
   of the below two subarrays.
    a)  From first element of ar1 to m1 (ar1[0...|_n/2_|])
    b)  From m2 to last element of ar2  (ar2[|_n/2_|...n-1])
4) If m2 is greater than m1, then median is present in one
   of the below two subarrays.
   a)  From m1 to last element of ar1  (ar1[|_n/2_|...n-1])
   b)  From first element of ar2 to m2 (ar2[0...|_n/2_|])
5) Repeat the above process until size of both the subarrays
   becomes 2.
6) If size of the two arrays is 2 then use below formula to get
  the median.
    Median = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2
```

Example:

```
   ar1[] = {1, 12, 15, 26, 38}
```

```
ar2[] = {2, 13, 17, 30, 45}
```

For above two arrays m1 = 15 and m2 = 17

For the above ar1[] and ar2[], m1 is smaller than m2. So median is present in one of the following two subarrays.

```
[15, 26, 38] and [2, 13, 17]
```

Let us repeat the process for above two subarrays:

```
m1 = 26 m2 = 13.
```

m1 is greater than m2. So the subarrays become

```
  [15, 26] and [13, 17]
Now size is 2, so median = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2
                       = (max(15, 13) + min(26, 17))/2
                       = (15 + 17)/2
                       = 16
```

Implementation:

```
#include<stdio.h>

int max(int, int); /* to get maximum of two integers */
int min(int, int); /* to get minimum of two integeres */
int median(int [], int); /* to get median of a sorted array */

/* This function returns median of ar1[] and ar2[].
   Assumptions in this function:
   Both ar1[] and ar2[] are sorted arrays
   Both have n elements */
int getMedian(int ar1[], int ar2[], int n)
{
    int m1; /* For median of ar1 */
    int m2; /* For median of ar2 */

    /* return -1  for invalid input */
    if (n <= 0)
        return -1;

    if (n == 1)
        return (ar1[0] + ar2[0])/2;

    if (n == 2)
        return (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1])) / 2;

    m1 = median(ar1, n); /* get the median of the first array */
    m2 = median(ar2, n); /* get the median of the second array */

    /* If medians are equal then return either m1 or m2 */
    if (m1 == m2)
        return m1;

     /* if m1 < m2 then median must exist in ar1[m1....] and ar2[....m2] */
    if (m1 < m2)
    {
        if (n % 2 == 0)
```

```
            return getMedian(ar1 + n/2 - 1, ar2, n - n/2 +1);
        else
            return getMedian(ar1 + n/2, ar2, n - n/2);
    }

    /* if m1 > m2 then median must exist in ar1[....m1] and ar2[m2...] */
    else
    {
        if (n % 2 == 0)
            return getMedian(ar2 + n/2 - 1, ar1, n - n/2 + 1);
        else
            return getMedian(ar2 + n/2, ar1, n - n/2);
    }
}

/* Function to get median of a sorted array */
int median(int arr[], int n)
{
    if (n%2 == 0)
        return (arr[n/2] + arr[n/2-1])/2;
    else
        return arr[n/2];
}

/* Driver program to test above function */
int main()
{
    int ar1[] = {1, 2, 3, 6};
    int ar2[] = {4, 6, 8, 10};
    int n1 = sizeof(ar1)/sizeof(ar1[0]);
    int n2 = sizeof(ar2)/sizeof(ar2[0]);
    if (n1 == n2)
      printf("Median is %d", getMedian(ar1, ar2, n1));
    else
     printf("Doesn't work for arrays of unequal size");

    getchar();
    return 0;
}

/* Utility functions */
int max(int x, int y)
{
    return x > y? x : y;
}

int min(int x, int y)
{
    return x > y? y : x;
}
```

Time Complexity: O(logn)
Algorithmic Paradigm: Divide and Conquer

Method 3 (By doing binary search for the median):
The basic idea is that if you are given two arrays ar1[] and ar2[] and know the length of each, you can check whether an element ar1[i] is the median in constant time. Suppose that the median is ar1[i]. Since

the array is sorted, it is greater than exactly i values in array ar1[]. Then if it is the median, it is also greater than exactly j = n – i – 1 elements in ar2[].
It requires constant time to check if ar2[j] <= ar1[i] <= ar2[j + 1]. If ar1[i] is not the median, then depending on whether ar1[i] is greater or less than ar2[j] and ar2[j + 1], you know that ar1[i] is either greater than or less than the median. Thus you can binary search for median in O(lg n) worst-case time.

For two arrays ar1 and ar2, first do binary search in ar1[]. If you reach at the end (left or right) of the first array and don't find median, start searching in the second array ar2[].

```
1) Get the middle element of ar1[] using array indexes left and right.
   Let index of the middle element be i.
2) Calculate the corresponding index j of ar2[]
     j = n - i - 1
3) If ar1[i] >= ar2[j] and ar1[i] <= ar2[j+1] then ar1[i] and ar2[j]
   are the middle elements.
     return average of ar2[j] and ar1[i]
4) If ar1[i] is greater than both ar2[j] and ar2[j+1] then
     do binary search in left half  (i.e., arr[left ... i-1])
5) If ar1[i] is smaller than both ar2[j] and ar2[j+1] then
     do binary search in right half (i.e., arr[i+1....right])
6) If you reach at any corner of ar1[] then do binary search in ar2[]
```

Example:

```
   ar1[] = {1, 5, 7, 10, 13}
   ar2[] = {11, 15, 23, 30, 45}
```

Middle element of ar1[] is 7. Let us compare 7 with 23 and 30, since 7 smaller than both 23 and 30, move to right in ar1[]. Do binary search in {10, 13}, this step will pick 10. Now compare 10 with 15 and 23. Since 10 is smaller than both 15 and 23, again move to right. Only 13 is there in right side now. Since 13 is greater than 11 and smaller than 15, terminate here. We have got the median as 12 (average of 11 and 13)

Implementation:

```
#include<stdio.h>

int getMedianRec(int ar1[], int ar2[], int left, int right, int n);

/* This function returns median of ar1[] and ar2[].
   Assumptions in this function:
   Both ar1[] and ar2[] are sorted arrays
   Both have n elements */
int getMedian(int ar1[], int ar2[], int n)
{
    return getMedianRec(ar1, ar2, 0, n-1, n);
}

/* A recursive function to get the median of ar1[] and ar2[]
   using binary search */
int getMedianRec(int ar1[], int ar2[], int left, int right, int n)
{
    int i, j;

    /* We have reached at the end (left or right) of ar1[] */
    if (left > right)
        return getMedianRec(ar2, ar1, 0, n-1, n);
```

```
    i = (left + right)/2;
    j = n - i - 1;  /* Index of ar2[] */

    /* Recursion terminates here.*/
    if (ar1[i] > ar2[j] && (j == n-1 || ar1[i] <= ar2[j+1]))
    {
        /* ar1[i] is decided as median 2, now select the median 1
           (element just before ar1[i] in merged array) to get the
           average of both*/
        if (i == 0 || ar2[j] > ar1[i-1])
            return (ar1[i] + ar2[j])/2;
        else
            return (ar1[i] + ar1[i-1])/2;
    }

    /*Search in left half of ar1[]*/
    else if (ar1[i] > ar2[j] && j != n-1 && ar1[i] > ar2[j+1])
        return getMedianRec(ar1, ar2, left, i-1, n);

    /*Search in right half of ar1[]*/
    else /* ar1[i] is smaller than both ar2[j] and ar2[j+1]*/
        return getMedianRec(ar1, ar2, i+1, right, n);
}

/* Driver program to test above function */
int main()
{
    int ar1[] = {1, 12, 15, 26, 38};
    int ar2[] = {2, 13, 17, 30, 45};
    int n1 = sizeof(ar1)/sizeof(ar1[0]);
    int n2 = sizeof(ar2)/sizeof(ar2[0]);
    if (n1 == n2)
        printf("Median is %d", getMedian(ar1, ar2, n1));
    else
        printf("Doesn't work for arrays of unequal size");

    getchar();
    return 0;
}
```

Time Complexity: O(logn)
Algorithmic Paradigm: Divide and Conquer

The above solutions can be optimized for the cases when all elements of one array are smaller than all elements of other array. For example, in method 3, we can change the getMedian() function to following so that these cases can be handled in O(1) time. Thanks to nutcracker for suggesting this optimization.

```
/* This function returns median of ar1[] and ar2[].
   Assumptions in this function:
   Both ar1[] and ar2[] are sorted arrays
   Both have n elements */
int getMedian(int ar1[], int ar2[], int n)
{
   // If all elements of array 1 are smaller then
   // median is average of last element of ar1 and
   // first element of ar2
   if (ar1[n-1] < ar2[0])
     return (ar1[n-1]+ar2[0])/2;
```

```
    // If all elements of array 1 are smaller then
    // median is average of first element of ar1 and
    // last element of ar2
    if (ar2[n-1] < ar1[0])
      return (ar2[n-1]+ar1[0])/2;

    return getMedianRec(ar1, ar2, 0, n-1, n);
}
```

References:
http://en.wikipedia.org/wiki/Median

http://ocw.alfaisal.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/30C68118-E436-4FE3-8C79-6BAFBB07D935/0/ps9sol.pdf ds3etph5wn

Asked by Snehal

Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.

Related Topics:

- Find Union and Intersection of two unsorted arrays
- Pythagorean Triplet in an array
- Maximum profit by buying and selling a share at most twice
- Design a data structure that supports insert, delete, search and getRandom in constant time
- Print missing elements that lie in range 0 – 99
- Iterative Merge Sort
- Group multiple occurrence of array elements ordered by first occurrence
- Given a sorted and rotated array, find if there is a pair with a given sum

Tags: Divide and Conquer

Writing code in comment? Please use **ideone.com** and share the link here.

c++ implementation of method 1 where the two arrays may have different size:
https://github.com/maheshwari-...

Reply • Share ›

Jassi • 2 months ago

Doesn't method 3 assume that sizes of array1 and array2 are same? In the recursion case when left > right, it is passing n as the size of array2.

Reply • Share ›

Guest • 2 months ago

An Iterative version: http://ideone.com/7duNj3

Reply • Share ›

archie • 2 months ago

can some one tell me wat will be the case in method second if 2 i/p of array are like this arr[]= {1,5,17,20}
arr2={4,8,13,19}

Reply • Share ›

Alla vamsi krishna • 2 months ago

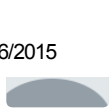
```
public class Program
{
private static void Main( string[] args )
{
int[] a1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};//{2, 4, 6, 8, 10};
int[] a2 = {12, 13, 14, 15, 16, 17, 18, 20};//{1, 3, 5, 7, 9, 11};
int median = findMedian( a1, a1.Length, a2, a2.Length, true, (a1.Length + a2.Length) / 2 );
Console.WriteLine( median );
}

private static int min( int a, int b )
{
return a < b ? a : b;
}

private static int max( int a, int b )
{
return a > b ? a : b;
```

see more

Reply • Share ›

Cracker • 2 months ago

http://algods-cracker.blogspot...



Reply • Share ›

NicolasR • 4 months ago

Hey Everyone,

In Method 2, it's not clear to me what definition of the median is used in the case where the median is not unique.

"If there is an even number of observations, then there is no single middle value;" from http://en.wikipedia.org/wiki/M...

Method 2 sometimes returns a value that is a correct median but which is not the the mean of the two middle values.

Anyone is aware of this ? It's not necessarily a big deal, I'm just wondering if it's true of if i'm missing something.

An example :

[371, 470, 750, 873] [70, 491, 535, 746]

m1 = 610.0 m2= 513.0

---

see more

7 Reply • Share ›

deeksha jain • 4 months ago

Isnt this just redundant in the first method, as the for loop will anyways exit even without this check? As count becomes n+1.

```
if (i == n)
{
m1 = m2;
m2 = ar2[0];
break;
}
else if (j == n)
{
m1 = m2;
m2 = ar1[0];
break;
}
```

1 Reply • Share ›

Ganesh Vijay ➚ deeksha jain • 4 months ago

No its not redundant. The loop for count runs through 0 to n not n-1. Basically the first two if statements are to handle corner cases. The program works well with the normal merge. But consider the below case
arr1= {1,2,3,4,5 } and arr2 = {6,7,8,9,10} So after n-1 iteration m1 will be 4 and m2 will be 5. But that will not give you median. The actual median is (5+6)/2. So we run the look till n (count <= n). so that we adjust the values of m1 and m2 to 5 and 6 which returns the perfect median.
The only redundancy is m1=m2 by program structure. If its not affecting the logc, you can take that out..!!

Reply • Share ›

The Internet • 5 months ago

I'm not sure if I've pointed this out before, but I have revisited this code and tried to write it in a more generic fashion. A lot of the arithmetic can be simplified and if-statements eliminated. Take a look at this implementation: http://ideone.com/WyL31e. I have only tested it with the example given and haven't been terribly thorough in testing. I have used this problem primarily as an exercise in using templates in C++11 and type traits, because I felt that integers unnecessarily constrain the solution and this algorithm could be applied to any arithmetic type, such as floats.

Reply • Share ›

Tanay • 5 months ago

Can't we find the median if the size of the two arrays are different???

Reply • Share ›

TheInternet ➚ Tanay • 5 months ago

Sure we can! I think this article just constrains the inputs in this manner, because it allows the improvement in efficiency shown by these methods to work. Try to go through these methods assuming that you have two sorted arrays of different size. You will see that it becomes a bit more tricky - at least it seems that way to me thinking about this on the spot, superficially, and not trying too hard. Have you made any attempt to see if these methods can be adapted to arrays of different sizes? If so, please let us know!

Reply • Share ›

guest_12345 ➚ TheInternet • a month ago

The method says when size of two arrays become 2, you have to apply some formula. If the sizes are different, one will reduce to length 2 before another. What will you do then.?

Reply • Share ›

ravi • 6 months ago

we are calculating mean here , but median has been asked in question.

• Reply • Share ›

Pooja ↗ ravi • 6 months ago

@ravi - We are calculating median. Mean would have been adding all the elements and dividing by the number of elements.

• Reply • Share ›

geosharath • 7 months ago

if the median element repeats then method 3 enters infinite loop.

1 • Reply • Share ›

purva • 7 months ago

I think 3rd method is wrong
if input is:
arr[1] ={1,12,15,26,38}
arr[2]= {2,4,8,9,11}

5 • Reply • Share ›

guest • 8 months ago

There should be equal sign to terminate recursion in method 3. (ar1[i] >= ar2[j])

For this case it will go in infinite loop :
int ar1[] = {-13, -11,3,40,50};
int ar2[] = {-20,-10,3,66,88};

```
if (ar1[i] > ar2[j] && (j == n-1 || ar1[i] <= ar2[j+1]))
{
/* ar1[i] is decided as median 2, now select the median 1
(element just before ar1[i] in merged array) to get the
average of both*/
if (i == 0 || ar2[j] > ar1[i-1])
return (ar1[i] + ar2[j])/2;
else
return (ar1[i] + ar1[i-1])/2;
}
```

1 • Reply • Share ›

mayank yadav ↗ guest • 8 months ago

They should have changed it by now.

• Reply • Share ›

Guest ↗ guest • 8 months ago

Correct :) I logged in just to write this as a comment.

• Reply • Share ›

guest ➚ Guest • 8 months ago

It still needs to be changed on page.

▲ | ▼ • Reply • Share ›

Siva Krishna • 8 months ago

Can any one explain why this is correct?

3) If m1 is greater than m2, then median is present in one
of the below two subarrays.
a) From first element of ar1 to m1 (ar1[0...|_n/2_|])
b) From m2 to last element of ar2 (ar2[|_n/2_|...n-1])
4) If m2 is greater than m1, then median is present in one
of the below two subarrays.
a) From m1 to last element of ar1 (ar1[|_n/2_|...n-1])
b) From first element of ar2 to m2 (ar2[0...|_n/2_|])

5 ▲ | ▼ • Reply • Share ›

Kim Jong-il ➚ Siva Krishna • 8 months ago

First of all consider first case : M1 == M2
==> It means that ((n/2)-1) elements of A1[] is less than M1 and same is true for A2[] also. It means total (n-2) elements are less than M1 or M2. But we all know that median is Nth element and we have 2 elements M1 and M2 which are equal. So it becomes (n-2+2)= N, hence M1 or M2 will be the answer.

Second Case : M1>M2
==> It means that M1 is greater than ((n/2)-1) elements of A2[], (Note that here i m talking about A2[], not A1[], Because its obvious that M1 is greater that ((n/2)-1) elements of A1[]) and we are not sure that how many element present in the A1[], which are less than M2. Hence up to this point we are not sure that which element is our Nth element. So to find Nth element we have to move First half of A1[] and second half of A2[] so that we can get a position which is Nth position.

Similar logic is involved in the third case to, M1<m2. hope="" it="" helps.="" all="" the="" best.="">

7 ▲ | ▼ • Reply • Share ›

prakhar • 9 months ago

I think second method is wrong
consider

input#1: {1,10,17,26} ,{2,13,15,30}
input#2: {1,10,15,26} ,{2,13,17,30}

Since if we merge them, arrays obtained will be same, then median of these two test cases

should be same..

But according to method two it is not.
output #1=12
output #2=14

• Reply • Share ›

falku ➚ prakhar • 7 months ago

He is right sir ! both inputs are giving the same output i.e. 14 :)

1 • Reply • Share ›

Achin Saxena ➚ prakhar • 9 months ago

Second method is correct.....both the inputs are giving the same output i.e., 14.

• Reply • Share ›

div1234522 • 9 months ago

Thanks! :D

• Reply • Share ›

Ankit Goyal • 9 months ago

What happens when the constraint that both the arrays are of same size is removed?

• Reply • Share ›

Vãíbhåv Joshî • 10 months ago

method 1 -> http://ideone.com/JYnVrl

http://ideone.com/s8GNTD

• Reply • Share ›

Vãíbhåv Joshî • 10 months ago

how do i pass a sub array of an array as a parameter in java method ????

1 • Reply • Share ›

kashish • 10 months ago

/* Here is working piece of code, along with some sample input*/

```
int findMedian(int a[], int alo, int ahi, int b[], int blo, int bhi)
{
int amid = alo + (ahi - alo)/2;
int amed = a[amid]; //median of array a
int bmid = blo + (bhi - blo)/2;
int bmed = b[bmid]; //median of array b
if((ahi - alo) <= 1 && (bhi - blo) <= 1)
{
//when size of both arrays become 2
```

```
//when size of both arrays becme 2
return (max(a[alo], b[blo]) + min(a[ahi], b[bhi]))/2;

}
else if(amed < bmed)
{
/*median lies from amid to ahi
* or from blo to bmid.
```

see more

• Reply • Share ›

Deepanshu Agrawal ➤ kashish • 8 months ago

Sorry... i did not see that it works for only the arrays with same lengths...

• Reply • Share ›

Deepanshu Agrawal ➤ kashish • 8 months ago

I don't think that the above code works...Try the following testcase:

A[] = 15,17,29,57,60,63

B[] = 1,7,19,37,40

Amed = 29

Bmed = 19

New arrays after discarding the elements:

A[] = 15,17,29

B[] = 19,37,40

Amed = 17

Bmed = 37

New arrays after discarding the elements:

A[] = 17,29

B[] = 19,37

Now med = (max(17,19) + min(29,37))/2 = (19 + 29)/2 = 24
Actual med is 29.

• Reply • Share ›

Klove • a year ago

I do not understand this particular piece of code

getMedian(ar1 + n/2 - 1, ar2, n - n/2 +1); .. why is it ar1+n/2-1 and n-n/2+1 ?

1 ^ | v • Reply • Share ›

GOPI GOPINATH ➦ Klove • 10 months ago

arr1+n/2-1 means we are moving to the middle of the array. Please remember that array can act as a pointer. so here we are passing only second half of the array.

2 ^ | v • Reply • Share ›

Rudra • a year ago

Method 3 gives TLE for the following case:

int ar1[] = {1, 11, 11};
int ar2[] = {1, 1, 1};

Here is the fix:

Change "ar1[i] > ar2[j]" to "ar1[i] >= ar2[j]" in both the places in "getMedianRec()".

^ | v • Reply • Share ›

Amarnath Raju Vysyaraju • a year ago

Will the above methods work for this case :{1,2,3,9,10} & {4,5,6,7,8}.

Why are we assuming both that the median should come from both the arrays? Isn't this assumption wrong ?

^ | v • Reply • Share ›

kinshuk chandra • a year ago

Begin with ar1 and ar2, and let m1 and m2 be the 2 medians respectively and let M be actual median.
M will lie between m1 and m2. If m1 & gt;
m2 => M lies between m2 to m1, so all elements less than m1 but
more than m2, hence first half of ar1 and 2nd half of ar2. Likewise for
m2>m1. and go on until number of elements are 2. Here is the post -
http://k2code.blogspot.in/2011...

^ | v • Reply • Share ›

guest • a year ago

we can simply first calculate the median of the two small arrays then take the average of the two medians to give the median of 2n array
arr1-> median m1
arr2->median m2
final arr3[arr1..ar2]->(m1+m2)/2
gives median of arr3

gives median of arr3

• Reply • Share ›

wliao ➚ guest • a year ago

wouldn't work for {1, 2, 15, 90, 100}, {4,5,6,7,8}

• Reply • Share ›

Guest • a year ago

Please explain how method 3 would work on following input:
Array1: 1 2 3 4 5
Array2: 1 2 4 5 6
Median should be (3+4)/2 but through method 3, it comes out to be (2+4)/2. Am I going wrong somewhere ?

• Reply • Share ›

newCoder • a year ago

```
/**
* There are 2 sorted arrays A and B of size n each. Write an algorithm to
* find the median of the array obtained after merging the above 2
* arrays(i.e. array of length 2n). The complexity should be O(log(n))
*
* @param a
* {1,3,5,7,9}
* @param b
* {2,4,6,8,10}
*
* @return average of the 2 medians from the merged array of length 2n.
*/
public static int findMedian(int a[], int b[]) {
assert a.length == b.length;

int n = a.length;

int low1 = 0;
```

see more

• Reply • Share ›

Allen • a year ago

How do we know the median of two smaller array is still the median of the two original array in method 3 ?

• Reply • Share ›

Rohit Sharma ➚ Allen • a year ago

if distribution is uniform i.e. elements of array have merged alternatively

if distribution is uniform i.e. elements of array have merged alternatively

.

• Reply • Share ›

james • a year ago

So what if i want to find the 5th largest number of the two array with out merging the two arrays.

• Reply • Share ›

Timothy • a year ago

What about two different sized lists?
If you have {1,2,3} and {4,5,6,7,8,9,10} the correct median should be 5 but when you remove {1,7,8,9,10} during the recursive algorithm, the algorithm will try to find the median for {2,3} and {5,6,7,8,9,10} and return that, which is 6.

1 • Reply • Share ›

krishna Timothy • 10 months ago

i think we can do this problem in O(M+N),or is there any optimized algorithm,if exist can you give reference ?

• Reply • Share ›

Guest • a year ago

How would this work on input
Array 1: 1, 2, 7, 8
Array 2: 3, 4, 5, 6

Median of the two arrays is (4 + 5) / 2, but the algorithm would get rid of either 4 or 5 in first run.
Or am i missing something?

• Reply • Share ›

gourav pathak Guest • a year ago

Here repeating elements are also counted...So the median is (3+4)/2
as merged array is {1,1,2,2,3,4,5,6,7,8} and not {1,2,3,4,5,6,7,8}(which you are probably referring to in your comment)

• Reply • Share ›

Load more comments

- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

- # Recent Comments

  - Ashish Aggarwal

    Try Data Structures and Algorithms Made Easy -...

    Algorithms · 17 minutes ago

  - Vlad

    Thanks. Very interesting lectures.

    Expected Number of Trials until Success · 1 hour ago

  - cfh

    My implementation which prints the index of the...

    Longest Even Length Substring such that Sum of First and Second Half is same · 1 hour ago

  - Gaurav pruthi

    forgot to see that part ;)

    Bloomberg Interview | Set 1 (Phone Interview) · 1 hour ago

  - saeid aslami

    thanks

    Greedy Algorithms | Set 7 (Dijkstra's shortest path algorithm) · 1 hour ago

  - Cracker

    Implementation:...

    Implement Stack using Queues · 2 hours ago