Median of two sorted arrays of different sizes

This is an extension of median of two sorted arrays of equal size problem. Here we handle arrays of unequal size also.

The approach discussed in this post is similar to method 2 of equal size post. The basic idea is same, we find the median of two arrays and compare the medians to discard almost half of the elements in both arrays. Since the number of elements may differ here, there are many base cases that need to be handled separately. Before we proceed to complete solution, let us first talk about all base cases.

Let the two arrays be A[N] and B[M]. In the following explanation, it is assumed that N is smaller than or equal to M.

Base cases:
The smaller array has only one element
Case 1: N = 1, M = 1.
Case 2: N = 1, M is odd
Case 3: N = 1, M is even
The smaller array has only two elements
Case 4: N = 2, M = 2
Case 5: N = 2, M is odd
Case 6: N = 2, M is even

Case 1: There is only one element in both arrays, so output the average of A[0] and B[0].

Case 2: N = 1, M is odd
Let B[5] = {5, 10, 12, 15, 20}
First find the middle element of B[], which is 12 for above array. There are following 4 sub-cases.

…**2.1** If A[0] is smaller than 10, the median is average of 10 and 12.
…**2.2** If A[0] lies between 10 and 12, the median is average of A[0] and 12.
…**2.3** If A[0] lies between 12 and 15, the median is average of 12 and A[0].
…**2.4** If A[0] is greater than 15, the median is average of 12 and 15.
In all the sub-cases, we find that 12 is fixed. So, we need to find the median of B[M / 2 - 1], B[M / 2 + 1], A[0] and take its average with B[M / 2].

Case 3: N = 1, M is even
Let B[4] = {5, 10, 12, 15}
First find the middle items in B[], which are 10 and 12 in above example. There are following 3 sub-cases.
…**3.1** If A[0] is smaller than 10, the median is 10.
…**3.2** If A[0] lies between 10 and 12, the median is A[0].
…**3.3** If A[0] is greater than 10, the median is 12.
So, in this case, find the median of three elements B[M / 2 - 1], B[M / 2] and A[0].

Case 4: N = 2, M = 2
There are four elements in total. So we find the median of 4 elements.

Case 5: N = 2, M is odd
Let B[5] = {5, 10, 12, 15, 20}
The median is given by median of following three elements: B[M/2], max(A[0], B[M/2 - 1]), min(A[1], B[M/2 + 1]).

Case 6: N = 2, M is even
Let B[4] = {5, 10, 12, 15}
The median is given by median of following four elements: B[M/2], B[M/2 - 1], max(A[0], B[M/2 - 2]), min(A[1], B[M/2 + 1])

Remaining Cases:
Once we have handled the above base cases, following is the remaining process.
1) Find the middle item of A[] and middle item of B[].
…..**1.1)** If the middle item of A[] is greater than middle item of B[], ignore the last half of A[], let length of ignored part is idx. Also, cut down B[] by idx from the start.
…..**1.2)** else, ignore the first half of A[], let length of ignored part is idx. Also, cut down B[] by idx from the last.

Popular Posts

Following is C implementation of the above approach.

```
// A C program to find median of two sorted arrays of unequal size
#include <stdio.h>
#include <stdlib.h>

// A utility function to find maximum of two integers
int max( int a, int b )
{ return a > b ? a : b; }

// A utility function to find minimum of two integers
int min( int a, int b )
{ return a < b ? a : b; }

// A utility function to find median of two integers
float MO2( int a, int b )
{ return ( a + b ) / 2.0; }

// A utility function to find median of three integers
float MO3( int a, int b, int c )
{
    return a + b + c - max( a, max( b, c ) )
                     - min( a, min( b, c ) );
}

// A utility function to find median of four integers
float MO4( int a, int b, int c, int d )
{
    int Max = max( a, max( b, max( c, d ) ) );
    int Min = min( a, min( b, min( c, d ) ) );
    return ( a + b + c + d - Max - Min ) / 2.0;
}

// This function assumes that N is smaller than or equal to M
float findMedianUtil( int A[], int N, int B[], int M )
{
    // If the smaller array has only one element
    if( N == 1 )
    {
        // Case 1: If the larger array also has one element, simply ca
        if( M == 1 )
            return MO2( A[0], B[0] );

        // Case 2: If the larger array has odd number of elements, the
        // the middle 3 elements of larger array and the only element
        // smaller array. Take few examples like following
```

```
        // A = {9}, B[] = {5, 8, 10, 20, 30} and
        // A[] = {1}, B[] = {5, 8, 10, 20, 30}
        if( M & 1 )
            return MO2( B[M/2], MO3(A[0], B[M/2 - 1], B[M/2 + 1]) );

        // Case 3: If the larger array has even number of element, the
        // will be one of the following 3 elements
        // ... The middle two elements of larger array
        // ... The only element of smaller array
        return MO3( B[M/2], B[M/2 - 1], A[0] );
    }

    // If the smaller array has two elements
    else if( N == 2 )
    {
        // Case 4: If the larger array also has two elements, simply c
        if( M == 2 )
            return MO4( A[0], A[1], B[0], B[1] );

        // Case 5: If the larger array has odd number of elements, the
        // will be one of the following 3 elements
        // 1. Middle element of larger array
        // 2. Max of first element of smaller array and element just
        //    before the middle in bigger array
        // 3. Min of second element of smaller array and element just
        //    after the middle in bigger array
        if( M & 1 )
            return MO3 ( B[M/2],
                         max( A[0], B[M/2 - 1] ),
                         min( A[1], B[M/2 + 1] )
                       );

        // Case 6: If the larger array has even number of elements, th
        // median will be one of the following 4 elements
        // 1) & 2) The middle two elements of larger array
        // 3) Max of first element of smaller array and element
        //    just before the first middle element in bigger array
        // 4. Min of second element of smaller array and element
        //    just after the second middle in bigger array
        return MO4 ( B[M/2],
                     B[M/2 - 1],
                     max( A[0], B[M/2 - 2] ),
                     min( A[1], B[M/2 + 1] )
                   );
    }

    int idxA = ( N - 1 ) / 2;
```


Recent Comments

```
    int idxB = ( M - 1 ) / 2;

     /* if A[idxA] <= B[idxB], then median must exist in
        A[idxA....] and B[....idxB] */
    if( A[idxA] <= B[idxB] )
        return findMedianUtil( A + idxA, N / 2 + 1, B, M - idxA );

    /* if A[idxA] > B[idxB], then median must exist in
       A[...idxA] and B[idxB....] */
    return findMedianUtil( A, N / 2 + 1, B + idxA, M - idxA );
}

// A wrapper function around findMedianUtil(). This function makes
// sure that smaller array is passed as first argument to findMedianUt
float findMedian( int A[], int N, int B[], int M )
{
    if ( N > M )
       return findMedianUtil( B, M, A, N );

    return findMedianUtil( A, N, B, M );
}

// Driver program to test above functions
int main()
{
    int A[] = {900};
    int B[] = {5, 8, 10, 20};

    int N = sizeof(A) / sizeof(A[0]);
    int M = sizeof(B) / sizeof(B[0]);

    printf( "%f", findMedian( A, N, B, M ) );
    return 0;
}
```

Output:

```
10
```

Time Complexity: O(LogM + LogN)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Tpoics:

- Remove minimum elements from either side such that 2*min becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3

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

23 Comments GeeksforGeeks

Sort by Newest

prashant jha • 13 hours ago

u can also do it with tournament sort in lesser complexity

  • Reply • Share ›

Abhinay Madhasu • 9 days ago

this is a great explanation.. in many sources it was wrongly solved. thank you.

  • Reply • Share ›

lol • 24 days ago

The time complexity is not O(m+n) but O(smaller of m,n). The operations afte
size 2 is of constant time.

  • Reply • Share ›

Mangat Rai • 3 months ago

Better approach to get 1 base case only:-

take two mid elements in each array. if odd then,
mid1 = A[len]/2
mid2 = a{len]/2
else
mid1 = A[len]/2
mid2 = A[len]/2 -1

now compare these 4th mids of 2 arrays then, get smallest and greatest. In si
between these elements. Discard only even number elements both the sides.
i.e. one side you will have 3 or 4 elements other side as m.

  • Reply • Share ›

coder • 4 months ago

Can somebody please explain Case 5 and Case 6 in more detail?

• Reply • Share ›

Guest • 7 months ago

There is a typo. Instead of -
if(M & 1)
return MO2(B[M/2], MO3(A[0], B[M/2 - 1], B[M/2 + 1]));

should be -
if(M & 1)
return MO4(B[M/2], MO3(A[0], B[M/2 - 1], B[M/2 + 1]));

• Reply • Share ›

wasseypuriyan ➚ Guest • 7 months ago

Read it carefully.
we first find the median of MO3(A[0], B[M/2 - 1], B[M/2 + 1]) call this as average with B[M/2].
Hence MO2(B[M/2], 'New_Median')

2 • Reply • Share ›

Guest ➚ wasseypuriyan • 7 months ago

Oh.. my blunder :P

• Reply • Share ›

raghvendra • 9 months ago

[sourcecode language="C++"]
#include <iostream>

using namespace std;
int median(int a[],int b[],int n1,int n2)
{
int low=0,high=n1-1,mid,j,n=(n1+n2)/2;
while(low<=high)

```
{
mid=low+(high-low)/2;
//cout<<low<<" "<<high<<" "<<mid<<endl;
j=n-mid-1;
if(a[mid]>=b[j]&&a[mid]<=b[j+1])
return a[mid];
else if(a[mid]>b[j]&&a[mid]>b[j+1])
high=mid-1;
else low=mid+1;
}
```

see more

• Reply • Share ›

bateesh • 10 months ago

@Geeksforgeeks..

In case N is even say 10.
Then N-1 will be 9/2=4
so we will index 4th location.
If element at idx[a] is greater then we consider its first part.
total elements as per code will be N/2+1
i.e 10/2=5+1=6
But if we are at 4th location then we need to consider 5 elements only and not
fine but in case of even we are taking one extra element .dont we need to hanc
comment if m getting it wrng.

• Reply • Share ›

pritybhudolia • 11 months ago

This is very simple logic. Lets assume length of ar1[] is n and length of ar2[] is
be at location =[(n+m) / 2] + 1. Hope this is helpful. Works for all cases accorc

```
#include <iostream>
#include <cstdio>
using namespace std;

int getMedian(int ar1[], int ar2[], int n, int m)
{
    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;
        for (count = 0; count <=(m+n)/2; count++)
        {

            if (ar1[i] < ar2[j])
```

see more

1 ▲ ▼ • Reply • Share ›

Alien ➚ pritybhudolia • 8 months ago

This solution is similar to finding median using merge routine. the solut by Geeksforgeeks is o(log m + log n)

▲ ▼ • Reply • Share ›

shek8034 ➚ pritybhudolia • 11 months ago

Your approach is similar to the one used when the two arrays are of ec Nice modification. It works :)

▲ ▼ • Reply • Share ›

Prem Nirmal • 11 months ago

Very nice Prateek! Can you please show me the pseudo code? it is tedious to the algorithm.

Prateek Sharma • a year ago

No need to consider special cases.just simple median formula and that's it..Py
+logm)complexity...
[sourcecode language="Python"]
def medianOfTwoSortedArraysOfUnequalSize(a1,a2):
len1 = len(a1)
len2 = len(a2)
totalLength = len1+len2
if totalLength %2 != 0:
j = totalLength/2
else:
j = totalLength/2-1
u = 0
v = 0
mergeArray =[]
for i in range(j+1):
if u<len(a1) and v<len(a2):
if a1[u]< a2[v]:
mergeArray.append(a1[u])

see more

2 • Reply • Share ›

abhishek08aug • a year ago

Intelligent :D

```
/* Paste your code here (You may delete these lines if not writing c
```

• Reply • Share ›

abhishek08aug

Tumse na ho paega !

1 ^ | v • Reply • Share ›

Ali Fard • a year ago

how do you calculate time complexity as O(logn+logm)?

1 ^ | v • Reply • Share ›

pankaj • a year ago

Why can't u do same algo which is used to find kth smallest element in two sc in the case of madian. we need to find (n+m)/2 +1 th element in the array complexity will be O(log(N+M))

as follow!

```
 /* Paste your code here (You may delete these lines if not writing c
int k = (N+M)/2 +1;
int N = min(N, k);
int M = min(M, k);
if(N < M){
 swap N, M;
 swap A[], B[]
}
int step = k/4;
int i = min(N, k/2);
int j = k - i;
while(step){
 if(i>=0 &&i<=n && i>=0 && i<=n&& A[i] < B[i]){
```

see more

^ | v • Reply • Share ›

v • 2 years ago

Now we know that median withing elements of two sorted array will be at local

we can traverse on two array to have that middle no.

```
int nMiddle = (nCount1 + nCount2) / 2 + 1;  // not handling case //o

            int i=0;
            int j=0;
            int nMedianElement = 0;
            for (int k = 0; k < nMiddle; k++)
            {
                if (i == nCount1)
                {
                    nMedianElement = arr2[j];
                    j++;
                }
                else if (j == nCount2)
```

see more

• Reply • Share ›

PsychoCoder • 2 years ago

```
      /* if A[idxA] <= B[idxB], then median must exist in
        A[idxA....] and B[....idxB] */
    if( A[idxA] <= B[idxB] )
        return findMedianUtil( A + idxA, N / 2 + 1, B, M - idxA );

    /* if A[idxA] > B[idxB], then median must exist in
       A[...idxA] and B[idxB....] */
    return findMedianUtil( A, N / 2 + 1, B + idxA, M - idxA );
```

Is this portion is correct? I mean the index!

Reply • Share ›

Naruto PsychoCoder • a year ago

I think this is a typo ... the correct code should be

/* if A[idxA] <= B[idxB], then median must exist in
A[idxA....] and B[....idxB] */
if(A[idxA] B[idxB], then median must exist in
A[...idxA] and B[idxB....] */
return findMedianUtil(A, N / 2 + 1, B + idxB, M - idxB);

Reply • Share ›

asad Naruto • 3 months ago

It's not a typo error. But this is the most tricky part..!!
The explanation given in comments is wrong while the code is
median of subarray same as that of before, it's important to dis
from both the array. Since, size of A is smaller, idxA will always
same amount is discarded from both the array.
Try dry running on [1,5,6], [2,3,4,7,8] and you'll get my point (Co

Reply • Share ›

Subscribe Add Disqus to your site