

# Median of two sorted arrays - GeeksforGeeks

*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  $O(\log(n))$

[We strongly recommend that you click here and practice it, before moving on to the solution.](#)

*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.  

$$\text{Median} = (\max(\text{ar1}[0], \text{ar2}[0]) + \min(\text{ar1}[1], \text{ar2}[1]))/2$$

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  $\text{median} = (\max(\text{ar1}[0], \text{ar2}[0]) + \min(\text{ar1}[1], \text{ar2}[1]))/2$   
 $= (\max(15, 13) + \min(26, 17))/2$   
 $= (15 + 17)/2$   
 $= 16$

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;  
}
```

[Run on IDE](#)

Time Complexity:  $O(\log n)$

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  $ar2[j] \leq ar1[i] \leq ar2[j + 1]$ . 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  $ar1[i] \geq ar2[j]$  and  $ar1[i] \leq ar2[j+1]$  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.,  $arr[left \dots i-1]$ )
- 5) If  $ar1[i]$  is smaller than both  $ar2[j]$  and  $ar2[j+1]$  then do binary search in right half (i.e.,  $arr[i+1 \dots right]$ )
- 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:  $O(\log n)$

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>

Asked by Snehal

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