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

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.

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 call MO2()
        if( M == 1 )
            return MO2( A[0], B[0] );

        // Case 2: If the larger array has odd number of elements, then consider
        // the middle 3 elements of larger array and the only element of
        // 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, then median
        // 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 call MO4()
        if( M == 2 )
            return MO4( A[0], A[1], B[0], B[1] );
```

```
    // Case 5: If the larger array has odd number of elements, then median
    // 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, then
    // 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;
    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 findMedianUtil
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.