Median of two sorted arrays of same size

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)).

```
Input : ar1[] = {1, 12, 15, 26, 38}
        ar2[] = {2, 13, 17, 30, 45}
Output : 16

Explanation :
After merging two arrays, we get
{1, 2, 12, 13, 15. 17, 26, 30, 38, 45}
Middle two elements are 15 and 17
Average of middle elements is (15 + 17)/2
which is equal to 16
```

Recommended: Please solve it on "*PRACTICE* " first, before moving on to the solution.

Note : Since size of the set for which we are looking for median is even (2n), we need take average of middle two numbers and return floor of the average.

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.

```
// A Simple Merge based O(n) solution
// to find median of two sorted arrays

class Main
{
    // function to calculate median
    static int getMedian(int ar1[], int ar2[], int n)
    {
        int i = 0;
        int j = 0;
        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])
            {
                /* Store the prev median */
                m1 = m2;
                m2 = ar1[i];
                i++;
            }
            else
            {
                /* Store the prev median */
                m1 = m2;
                m2 = ar2[j];
                j++;
            }
        }

        return (m1 + m2)/2;
    }


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

        int n1 = ar1.length;
        int n2 = ar2.length;
        if (n1 == n2)
            System.out.println("Median is " +
                        getMedian(ar1, ar2, n1));
        else
            System.out.println("arrays are of unequal size");
    }
}
```

Output :

```
Median is 16
```

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
```

Examples :

```
   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 :

```
// A Java program to divide and conquer based
// efficient solution to find
// median of two sorted arrays
// of same size.
import java.util.*;
class GfG {

    /* This function returns median
    of ar1[] and ar2[].
    Assumptions in this function:
        Both ar1[] and ar2[] are
        sorted arrays
        Both have n elements */

    static int getMedian(

        int[] a, int[] b, int startA,
        int startB, int endA, int endB)
    {
        if (endA - startA == 1) {
            return (
                        Math.max(a[startA],
                                 b[startB])
                        + Math.min(a[endA], b[endB]))
                / 2;
        }
        /* get the median of
    the first array */
        int m1 = median(a, startA, endA);

        /* get the median of
    the second array */
        int m2 = median(b, startB, endB);

        /* 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] */
```

```
else if (m1 < m2) {
            return getMedian(
                a, b, (endA + startA + 1) / 2,
                startB, endA,
                (endB + startB + 1) / 2);
        }

        /* if m1 > m2 then median must
    exist in ar1[....m1] and
    ar2[m2...] */
        else {
            return getMedian(
                a, b, startA,
                (endB + startB + 1) / 2,
                (endA + startA + 1) / 2, endB);
        }
    }

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

    // Driver code
    public static void main(String[] args)
    {
        int ar1[] = { 1, 2, 3, 6 };
        int ar2[] = { 4, 6, 8, 10 };
        int n1 = ar1.length;
        int n2 = ar2.length;
        if (n1 != n2) {
            System.out.println(
                "Doesn't work for arrays "
                + "of unequal size");
        }
        else if (n1 == 0) {
            System.out.println("Arrays are empty.");
        }
        else if (n1 == 1) {
            System.out.println((ar1[0] + ar2[0]) / 2);
        }
        else {
            System.out.println(
                "Median is "
                + getMedian(
                      ar1, ar2, 0, 0,
                      ar1.length - 1, ar2.length - 1));
```

```
        }
    }
}
```

Output :

```
Median is 5
```

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