1) Same sizes (n, n)

1) Merge Sort

- Maintain the count while Merging
- If count becomes n, we have reached the median

```
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;
        }
         /* equals sign because if two
            arrays have some common elements */
        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;
}
```

Time Complexity ⇒ O(n)

2) Compare Medians of 2 Arrays O(logn)

1) Calculate median of both the arrays m_1 & m_2

2) If $m_1 == m_2$ return m_1

3) If $m_1 > m_2$, then search in

   a) array1[0] — m_1

   b) m_2 — array[n-1]

4) Similarly for $m_2 > m_1$

5) If size of Arrays is 2 then

$$\text{Median} = \frac{\max(a1_0, a2_0) + \min(a1_1, a2_1)}{2}$$

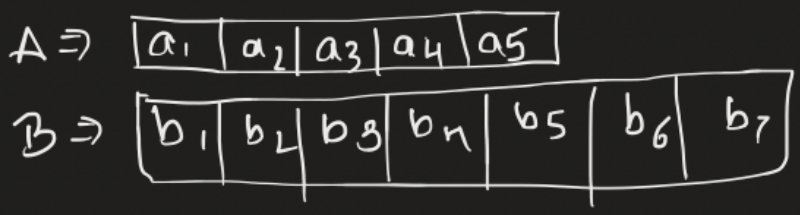
```
int getMedian(int ar1[], int ar2[], int n)
{
    /* return -1  for invalid input */
    if (n <= 0)
        return -1;
    if (n == 1)
        return (ar1[0] + ar2[0])/2;
    if (n == 2)
        return (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1])) / 2;

    int m1 = median(ar1, n); /* get the median of the first array */
    int 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);
        return getMedian(ar1 + n/2, ar2, n - n/2);
    }

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

2) Different Sizes (m, n)

1) Similar to method 1 in Same sizes Count in Merge Sort.

$O(n+m)$

2) Divide and Conquer $O(\log(m+n))$



total length ⇒ $m+n$

length of left half of sorted array = $\frac{m+n}{2}$

same for right half

⇒ Let's say we take x elements from A

$a_1 \dots a_x$

⇒ Now we have to take $y = \frac{m+n}{2} - x$ elements from B

⇒ Now, we have

$a_1 \dots a_x \mid a_{x+1} \dots a_n$

$b_1 \dots b_y \mid b_{y+1} \dots b_m$

→ Checking if split has been made at the correct point

Eg:→ 1 3 4 | 7 10 12

2 3 | 6 15

For correct split

$4 <= 6$ && $3 <= 7$

All elements in the left half should be smaller than all elements on the right

→ We know that $a_x < a_{x+1}$ & $b_y < b_{y+1}$

→ We only need to make sure

$a_x <= b_{y+1}$ & $b_y <= a_{x+1}$

Finally our median is

$$\Rightarrow \frac{\text{Max of left} + \text{Min of right}}{2}$$

$$\Rightarrow \frac{\text{Max}(a_x, b_y) + \text{Min}(a_{x+1}, b_{y+1})}{2}$$

→ We can use binary search to find the correct partion

→ We can use binary search to find the correct partion

→ Let's say $a_x <= b_{y+1}$, but

$$b_y > a_{x+1}$$

In this case, we need to decrease x and increase y.

– Same for other case

low ⇒ least no. of elements that can be taken from A to form left half

⇒ 0 if $l_2 >$ size of left half

⇒ leftHalf − l_2 if $l_2 <$ leftHalf

high ⇒ max no. of elements that can be taken from right half

⇒ l_1 if $l_1 <$ size of leftHalf

⇒ leftHalf if $l_1 >$ size of leftHalf

→ if $l_1 + l_2$ is odd, then either left or right half (which ever has more elements) contains the answer