Median of Two Sorted Arrays

Given two sorted arrays, **a[]** and **b[]**, the task is to find the median of these sorted arrays, where **N** is the number of elements in the first array, and **M** is the number of elements in the second array.

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

Examples:

> ***Input:*** *a[] = {-5, 3, 6, 12, 15}, b[] = {-12, -10, -6, -3, 4, 10}*
> ***Output:*** *The median is 3.*
> ***Explanation:*** *The merged array is: ar3[] = {-12, -10, -6, -5 , -3, 3, 4, 6, 10, 12, 15}.*
> *So the median of the merged array is 3*
>
> ***Input:*** *a[] = {2, 3, 5, 8}, b[] = {10, 12, 14, 16, 18, 20}*
> ***Output:*** *The median is 11.*
> ***Explanation :*** *The merged array is: ar3[] = {2, 3, 5, 8, 10, 12, 14, 16, 18, 20}*
> *If the number of the elements are even. So there are two middle elements.*
> *Take the average between the two: (10 + 12) / 2 = 11.*

Median of two sorted arrays of different sizes using Binary Search:

> *The given two arrays are sorted, so we can utilize the ability of Binary Search to divide the array and find the median. Median means the point at which the whole array is divided into two parts. Hence since the two arrays are not merged so to get the median we require merging which is costly. Hence instead of merging, we will use a modified binary search algorithm to efficiently find the median.*

Illustration:

- ***realmidinmergedarray = 6.***

  - *leftB = -3*
  - *rightA = 6*
  - *rightB = 4*
- ***A[ ]** = { -5, **3**, 6, 12, 15 } & **B[ ]** = { -12, -10, -6, **-3**, 4, 10}*
- *As **leftA <= rightB** and **leftB <= rightA**, so the condition holds and **3** is returned as the **median***

Follow the steps below to solve the problem:

- If we would have merged the two arrays, the median is the point that will divide the sorted merged array into two equal parts. So the actual median point in the merged array would have been (M+N+1)/2;
- We divide A[] and B[] into two parts. We will find the mid value and divide the

of both A[] and B[] will result in the left part of the merged array.

- Now we have 4 variables indicating four values two from array A[] and two from array B[].
  - leftA -> Rightmost element in left part of A.
  - leftb -> Rightmost element in left part of B
  - rightA -> Leftmost element in right part of A
  - rightB -> Leftmost element in right part of B
- Hence to confirm that the partition was correct we have to check if leftA<=rightB and leftB<=rightA. This is the case when the sum of two parts of A and B results in the left part of the merged array.
  - If the condition fails we have to find another midpoint in A and then left part in B[].
  - If we find **leftA > rightB**. means we have to decrease the size of A's partition and shift to lesser value in A[].
  - So update the right pointer of to mid-1 else we will increase the left pointer to mid+1.
  - Repeat the above steps with new partitions till we get the answers.
- If **leftA ≤ rightB** and **leftB ≤ rightA**, then we get the correct partition and our answer depends on the total size of the merged array (i.e. M+N). If (M+N) is even we take max(leftA, leftB) and min(rightA, rightB), add them and divide

C++ Java

Dash

All

Articles

Videos

Problems

Quiz

Contest

```
// Method to find median
static double Median(int[] A, int[] B)
{
    int n = A.length;
    int m = B.length;
    if (n > m)
        return Median(B,
                      A); // Swapping to make A smaller

    int start = 0;
    int end = n;
    int realmidinmergedarray = (n + m + 1) / 2;

    while (start <= end) {
        int mid = (start + end) / 2;
        int leftAsize = mid;
        int leftBsize = realmidinmergedarray - mid;
        int leftA
            = (leftAsize > 0)
                ? A[leftAsize - 1]
                : Integer
                        .MIN_VALUE; // checking overflow
                                    // of indices
        int leftB = (leftBsize > 0) ? B[leftBsize - 1]
                                    : Integer.MIN_VALUE;
        int rightA = (leftAsize < n)
                         ? A[leftAsize]
                         : Integer.MAX_VALUE;
        int rightB = (leftBsize < m)
                         ? B[leftBsize]
                         : Integer.MAX_VALUE;

        // if correct partition is done
        if (leftA <= rightB && leftB <= rightA) {
            if ((m + n) % 2 == 0)
                return (Math.max(leftA, leftB)
```

Menu

Track Progress

53 of 60 Complete. (89%)

```
        return Math.max(leftA, leftB);
    }
    else
        start = mid + 1;
}
return 0.0;
}

// Driver code
public static void main(String[] args)
{
    int[] arr1 = { -5, 3, 6, 12, 15 };
    int[] arr2 = { -12, -10, -6, -3, 4, 10 };
    System.out.println("Median of the two arrays are");
    System.out.println(Median(arr1, arr2));
}
}
```


Output

```
Median of the two arrays are
3
```

Time Complexity: O(min(log M, log N)): Since binary search is being applied on the smaller of the 2 arrays
Auxiliary Space: O(1)

Dash

All

Articles

Videos

Problems

Quiz

Contest

Menu

Track Progress

53 of **60** Complete. (89%)