Median of Two Sorted Arrays

There are two sorted arrays nums1 and nums2 of size m and n respectively. Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

You may assume nums1 and nums2 cannot be both empty.

Example 1:

```
nums1 = [1, 3]
nums2 = [2]
```

The median is 2.0

Example 2:

```
nums1 = [1, 2]
nums2 = [3, 4]
```

The median is (2 + 3)/2 = 2.5

Solution 1 in C++

```
#include <iostream>
#include <algorithm>
#include <vector>


using namespace std;

class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        nums1.insert( nums1.end(), nums2.begin(), nums2.end());
        sort( nums1.begin(), nums1.end());
        int num_size = nums1.size();
        if (num_size)
        {
            int iMedian=(int)(nums1.size()/2);

            cout << "count of numbers:" << num_size << endl;
            for( int i=0; i<num_size; i++)
                cout << nums1[i] << " ";
            cout << endl;
            cout << "Median index:" << iMedian << endl;

            if (num_size%2==1) return (double)nums1[iMedian];
            else
            {
                return (double)(nums1[iMedian]+nums1[iMedian-1])/2;
            }
        }
        else return 0;
    }
};

int main(void)
{
/*
    vector<int> nums1{ 1, 3};
    vector<int> nums2{ 2};
*/

    vector<int> nums1{ 1, 2};
    vector<int> nums2{ 3, 4};

    Solution s;
    cout << s.findMedianSortedArrays( nums1, nums2) << endl;
}
```

Solution 2 in Java

```
class Solution {
    public double findMedianSortedArrays(int[] A, int[] B) {
        int m = A.length;
        int n = B.length;
        if (m > n) { // to ensure m<=n
            int[] temp = A; A = B; B = temp;
            int tmp = m; m = n; n = tmp;
        }
        int iMin = 0, iMax = m, halfLen = (m + n + 1) / 2;
        while (iMin <= iMax) {
            int i = (iMin + iMax) / 2;
            int j = halfLen - i;
            if (i < iMax && B[j-1] > A[i]){
                iMin = i + 1; // i is too small
            }
            else if (i > iMin && A[i-1] > B[j]) {
                iMax = i - 1; // i is too big
            }
            else { // i is perfect
                int maxLeft = 0;
                if (i == 0) { maxLeft = B[j-1]; }
                else if (j == 0) { maxLeft = A[i-1]; }
                else { maxLeft = Math.max(A[i-1], B[j-1]); }
                if ( (m + n) % 2 == 1 ) { return maxLeft; }

                int minRight = 0;
                if (i == m) { minRight = B[j]; }
                else if (j == n) { minRight = A[i]; }
                else { minRight = Math.min(B[j], A[i]); }

                return (maxLeft + minRight) / 2.0;
            }
        }
        return 0.0;
    }
}
```

Solution 3 in Python

```
def median(A, B):
    m, n = len(A), len(B)
    if m > n:
        A, B, m, n = B, A, n, m
    if n == 0:
        raise ValueError

    imin, imax, half_len = 0, m, (m + n + 1) / 2
    while imin <= imax:
        i = (imin + imax) / 2
        j = half_len - i
        if i < m and B[j-1] > A[i]:
            # i is too small, must increase it
            imin = i + 1
        elif i > 0 and A[i-1] > B[j]:
            # i is too big, must decrease it
            imax = i - 1
        else:
            # i is perfect

            if i == 0: max_of_left = B[j-1]
            elif j == 0: max_of_left = A[i-1]
            else: max_of_left = max(A[i-1], B[j-1])

            if (m + n) % 2 == 1:
                return max_of_left

            if i == m: min_of_right = B[j]
            elif j == n: min_of_right = A[i]
            else: min_of_right = min(A[i], B[j])

            return (max_of_left + min_of_right) / 2.0
```