```
/**
 * 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)).
 *
 * Solution
 * Take minimum size of two array. Possible number of partitions are from 0 to m in m size array.
 * Try every cut in binary search way. When you cut first array at i then you cut second array at (m + n + 1)/2 - i
 * Now try to find the i where a[i-1] <= b[j] and b[j-1] <= a[i]. So this i is partition around which lies the median.
 *
 * Time complexity is O(log(min(x,y))
 * Space complexity is O(1)
 *
 * https://leetcode.com/problems/median-of-two-sorted-arrays/
 * https://discuss.leetcode.com/topic/4996/share-my-o-log-min-m-n-solution-with-explanation/4
 */
```

```
class Solution {
    public double findMedianSortedArrays(int[] a, int[] b) {
        if (a.length > b.length) {
            return findMedianSortedArrays(b, a);
        }
        int x = a.length, y = b.length;
        int l = 0, h = x;

        while (l <= h) {
            int partitionX = (l + h) / 2;
            int partitionY = (x + y + 1) / 2 - partitionX;

            int maxLeftX = (partitionX == 0) ? Integer.MIN_VALUE : a[partitionX - 1];
            int minRightX = (partitionX == x) ? Integer.MAX_VALUE : a[partitionX];

            int maxLeftY = (partitionY == 0) ? Integer.MIN_VALUE : b[partitionY - 1];
            int minRightY = (partitionY == y) ? Integer.MAX_VALUE : b[partitionY];

            if (maxLeftX <= minRightY && maxLeftY <= minRightX) {
                if ((x + y) % 2 == 0) {
                    return (Math.min(minRightY, minRightX) + Math.max(maxLeftX, maxLeftY)) * 0.5;
                } else {
                    return Math.max(maxLeftX, maxLeftY);
                }
            } else if (maxLeftX > minRightY) {
                h = partitionX - 1;
            } else {
                l = partitionX + 1;
            }
        }
        return -1;
    }
    /*

    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int i = 0, j = 0, k = 0;
        int LEN1 = nums1.length;
        int LEN2 = nums2.length;
        int MAX_LEN = LEN1 + LEN2;
        int [] list = new int [MAX_LEN];
        while(i < LEN1 || j < LEN2){
            if (i == LEN1){
                list[k++] = nums2[j++];
            }
            else if (j == LEN2){
                list[k++] = nums1[i++];
            }
            else if (nums1[i] <= nums2[j]){
                list[k++] = nums1[i++];
            } else {
                list[k++] = nums2[j++];
            }
            if (k > MAX_LEN / 2 + 1) {
                break;
            }
        }

        if (MAX_LEN % 2 == 1)
            return list[MAX_LEN / 2];
        else
            return 1.0 * (list[MAX_LEN / 2 - 1] + list[MAX_LEN / 2]) / 2;
    }
    */
}
```