PROBLEM-2

Median of Two Sorted Arrays

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be `O(log (m+n))`.

Example 1:

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

Output: 2.00000

Explanation: merged array = [1,2,3] and median is 2.
```

Example 2:

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

Output: 2.50000

Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.
```

Example 3:

```
Input: nums1 = [0,0], nums2 = [0,0]

Output: 0.00000
```

Example 4:

```
Input: nums1 = [], nums2 = [1]

Output: 1.00000
```

Example 5:

```
Input: nums1 = [2], nums2 = []

Output: 2.00000
```

Constraints:

- `nums1.length == m`
- `nums2.length == n`
- `0 <= m <= 1000`
- `0 <= n <= 1000`
- `1 <= m + n <= 2000`
- `-10^6 <= nums1[i], nums2[i] <= 10^6`

PROBLEM-2

SOLUTION

```
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        if (nums1.length > nums2.length) {
            return findMedianSortedArrays(nums2, nums1);
        }

        int m = nums1.length;
        int n = nums2.length;

        int start = 0;
        int end = m;

        while (start <= end) {

            int partitionNums1 = (start + end) / 2;
            int partitionNums2 = (m + n + 1) / 2 - partitionNums1;

            int maxLeftNums1 = partitionNums1 == 0 ? Integer.MIN_VALUE : nums1[partitionNums1 - 1];

            int minRightNums1 = partitionNums1 == m ? Integer.MAX_VALUE : nums1[partitionNums1];

            int maxLeftNums2 = partitionNums2 == 0 ? Integer.MIN_VALUE : nums2[partitionNums2 - 1];
            int minRightNums2 = partitionNums2 == n ? Integer.MAX_VALUE : nums2[partitionNums2];

            if (maxLeftNums1 <= minRightNums2 && maxLeftNums2 <= minRightNums1) {
```

PROBLEM-2

```
            if ((m + n) % 2 == 0) {
                return (Math.max(maxLeftNums1, maxLeftNums2) + Math.min(minRightNums1,
minRightNums2)) / 2.0;
            } else {
                return Math.max(maxLeftNums1, maxLeftNums2);
            }
        }

        else if (maxLeftNums1 > minRightNums2) {
            end = partitionNums1 - 1;
        }

        else {
            start = partitionNums1 + 1;
        }
    }

    throw new IllegalArgumentException();
  }

}
```

PROBLEM-2

OUTPUT