class Solution {

public:

double findMedianSortedArrays(vector<int> &nums1, vector<int> &nums2) {

int n1 = nums1.size(), n2 = nums2.size();

// Ensure nums1 is the smaller array for simplicity

if (n1 > n2)

return findMedianSortedArrays(nums2, nums1);

int n = n1 + n2;

int left = (n1 + n2 + 1) / 2; // Calculate the left partition size

int low = 0, high = n1;

while (low <= high) {

int mid1 = (low + high) >> 1; // Calculate mid index for nums1

int mid2 = left - mid1; // Calculate mid index for nums2

int l1 = INT\_MIN, l2 = INT\_MIN, r1 = INT\_MAX, r2 = INT\_MAX;

// Determine values of l1, l2, r1, and r2

if (mid1 < n1)

r1 = nums1[mid1];

if (mid2 < n2)

r2 = nums2[mid2];

if (mid1 - 1 >= 0)

l1 = nums1[mid1 - 1];

if (mid2 - 1 >= 0)

l2 = nums2[mid2 - 1];

if (l1 <= r2 && l2 <= r1) {

// The partition is correct, we found the median

if (n % 2 == 1)

return max(l1, l2);

else

return ((double)(max(l1, l2) + min(r1, r2))) / 2.0;

}

else if (l1 > r2) {

// Move towards the left side of nums1

high = mid1 - 1;

}

else {

// Move towards the right side of nums1

low = mid1 + 1;

}

}

return 0; // If the code reaches here, the input arrays were not sorted.

}

};