

■ Median of Two Sorted Arrays

The hardest binary search problem — broken down step by step

● Binary Search

■ HARD

■ What You Will Learn

- What a 'partition' of two arrays means
- How to binary search on the partition index
- The cross-validation condition for correct partition
- How to compute median from the partition
- Edge case handling: empty halves, odd/even total

■ Intuition First

You have two sorted arrays. To find the median of their combined elements, you need to find the 'middle split'. Imagine merging them (don't actually merge — that's $O(n)$). If total length is 10, the median is between element 5 and 6. We want to partition BOTH arrays such that exactly 5 elements are on the left total.

Binary search on the partition of the SMALLER array. Once you know how many elements to take from array 1, you automatically know how many from array 2. Then validate if this partition is correct.

■ Core Idea

Partition array1 at index i , array2 at index $j = (m+n+1)/2 - i$. This ensures left side has exactly half the elements. Check if $\text{left1} \leq \text{right2}$ AND $\text{left2} \leq \text{right1}$. If yes, perfect partition found. If not, adjust i .

■ Visual — The Partition

```

nums1 = [1, 3, 8]      nums2 = [2, 4, 6, 7]
Total = 7 elements → left half has 4 elements

Try partition i=2 in nums1:
nums1: | 1  3 | 8 |      left1=3, right1=8
nums2: | 2  4 | 6  7 |    left2=4, right2=6
(j = 4-2 = 2 elements from nums2 on left)

Check: left1(3) <= right2(6) ✓
      left2(4) <= right1(8) ✓
→ VALID PARTITION!

Total=7 (odd) → median = max(left1, left2) = max(3,4) = 4 ✓

If total=8 (even):
median = (max(left1, left2) + min(right1, right2)) / 2

```

■ Step-by-Step Algorithm

1. Ensure nums1 is the smaller array (swap if needed). This keeps binary search efficient.
2. Set left=0, right=m (partition index range for nums1)
3. Binary search: compute partition i in nums1, j in nums2
 $j = (m + n + 1) / 2 - i$
4. Get boundary values: left1=nums1[i-1], right1=nums1[i], left2=nums2[j-1], right2=nums2[j]
 (Use -INF for left when i=0, +INF for right when i=m)
5. If left1 <= right2 AND left2 <= right1 → found! Compute median.
6. Else if left1 > right2 → i too large → right = i-1
7. Else (left2 > right1) → i too small → left = i+1

■ Java Code

```

1  class Solution {
2  public double findMedianSortedArrays(int[] nums1, int[] nums2) {
3      // Ensure nums1 is smaller
4      if (nums1.length > nums2.length) {
5          return findMedianSortedArrays(nums2, nums1);
6      }
7      int m = nums1.length, n = nums2.length;
8      int left = 0, right = m;
9
10     while (left <= right) {
11         int i = left + (right - left) / 2; // partition in nums1
12         int j = (m + n + 1) / 2 - i; // partition in nums2
13
14         int left1 = (i == 0) ? Integer.MIN_VALUE : nums1[i-1];

```

```

15     int right1 = (i == m) ? Integer.MAX_VALUE : nums1[i];
16     int left2 = (j == 0) ? Integer.MIN_VALUE : nums2[j-1];
17     int right2 = (j == n) ? Integer.MAX_VALUE : nums2[j];
18
19     if (left1 <= right2 && left2 <= right1) {
20         // Perfect partition!
21         if ((m + n) % 2 == 1) {
22             return Math.max(left1, left2);
23         } else {
24             return (Math.max(left1, left2) + Math.min(right1, right2)) / 2.0;
25         }
26     } else if (left1 > right2) {
27         right = i - 1; // Move i left
28     } else {
29         left = i + 1; // Move i right
30     }
31 }
32 return 0.0; // Should never reach
33 }
34 }

```

■ ■ Complexity

Complexity	Value	What it means
Time	$O(\log(\min(m,n)))$	Binary search only on smaller array. With $m=1000$, $n=1000000$: ~10 steps not 1000000!
Space	$O(1)$	No extra arrays, only boundary variables

■ ■ Common Mistakes

■ ■ Mistake 1

Not handling empty halves with INF sentinels. When $i=0$, $left1$ has no element — use `Integer.MIN_VALUE`. When $i=m$, $right1$ has none — use `Integer.MAX_VALUE`.

■ ■ Mistake 2

Not swapping to make `nums1` smaller. j could become negative if `nums1` is larger, causing `ArrayIndexOutOfBoundsException`.

■ ■ Mistake 3

Integer division in final step. Use `2.0` not `2` to get the decimal median for even-length cases.

■ Pro Tips

■ Pro Tip

Spend time deeply understanding the partition visualization before coding. Draw it on paper with 2-3 examples. The code is short — it's the mental model that takes time.

■ Think Like This

The key is: 'I want to split both arrays so that ALL elements on the left are smaller than ALL elements on the right.' Binary search finds the unique i that achieves this.

■ Practice Problems

#	Problem	Difficulty
1	Median of Two Sorted Arrays (LeetCode 4)	Hard
2	Kth Smallest Element in a Sorted Matrix (LeetCode 378)	Medium
3	Find K-th Smallest Pair Distance (LeetCode 719)	Hard

■ Quick Revision

✓	Binary search on partition index i of smaller array
✓	$j = (m+n+1)/2 - i$ ensures equal split
✓	Valid partition: $\text{left1} \leq \text{right2}$ AND $\text{left2} \leq \text{right1}$
✓	Handle edges: $i=0 \rightarrow \text{left1} = -\text{INF}$, $i=m \rightarrow \text{right1} = +\text{INF}$ (same for j)
✓	Odd total: $\text{median} = \max(\text{left1}, \text{left2})$. Even: $(\max(\text{lefts}) + \min(\text{rights})) / 2.0$
✓	Time $O(\log \min(m, n))$, Space $O(1)$

■ Visual Thinking

Imagine two conveyor belts of sorted candies. You want to grab exactly half the total candies, taking some from each belt. The constraint: every candy you leave on the left side of both belts must be smaller than every candy on the right side. Binary search finds the exact cut point on belt 1; belt 2's cut follows automatically.