# **Leetcode 4 – Median of Two Sorted Arrays**

## Problem Understanding

You are given **two sorted arrays** nums1 and nums2 of sizes m and n respectively.  
Return the **median** of the two sorted arrays.  
The overall **runtime complexity** should be **O(log(min(m, n)))**.

### Constraints:

* Both arrays are individually sorted.
* Need to find the **median** of the merged array **without actually merging** it.
* Must solve it in **logarithmic time** — so binary search is necessary.

## Optimized Java Solution (Binary Search on Partition)

public class Solution {

public double findMedianSortedArrays(int[] nums1, int[] nums2) {

// Ensure nums1 is smaller for simpler logic

if (nums1.length > nums2.length)

return findMedianSortedArrays(nums2, nums1);

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

int left = 0, right = m;

while (left <= right) {

int partition1 = (left + right) / 2;

int partition2 = (m + n + 1) / 2 - partition1;

int maxLeft1 = (partition1 == 0) ? Integer.MIN\_VALUE : nums1[partition1 - 1];

int minRight1 = (partition1 == m) ? Integer.MAX\_VALUE : nums1[partition1];

int maxLeft2 = (partition2 == 0) ? Integer.MIN\_VALUE : nums2[partition2 - 1];

int minRight2 = (partition2 == n) ? Integer.MAX\_VALUE : nums2[partition2];

if (maxLeft1 <= minRight2 && maxLeft2 <= minRight1) {

if ((m + n) % 2 == 0) {

return (Math.max(maxLeft1, maxLeft2) + Math.min(minRight1, minRight2)) / 2.0;

} else {

return Math.max(maxLeft1, maxLeft2);

}

} else if (maxLeft1 > minRight2) {

right = partition1 - 1;

} else {

left = partition1 + 1;

}

}

throw new IllegalArgumentException("Invalid input");

}

}

## Dry Run Example

**Input:**  
nums1 = [1, 3], nums2 = [2, 4, 5]

Merged would be: [1, 2, 3, 4, 5] → Median is 3

But we won't merge.

**m = 2, n = 3 → total = 5**  
→ (m + n + 1) / 2 = 3 → we want to partition both arrays such that left half has 3 elements.

### Step 1:

* left = 0, right = 2
* partition1 = 1, partition2 = 2
* maxLeft1 = 1, minRight1 = 3
* maxLeft2 = 2, minRight2 = 4
* ✅ maxLeft1 <= minRight2 and maxLeft2 <= minRight1 → correct partition

### Median:

* Total is odd → median = max(maxLeft1, maxLeft2) = max(1, 2) = **2**

Oops! But actually merged array is [1, 2, 3, 4, 5], so median = **3**

Let’s adjust the partition and retry:

* partition1 = 2, partition2 = 1
* maxLeft1 = 3, minRight1 = ∞
* maxLeft2 = 2, minRight2 = 4
* Now maxLeft1 > minRight2 → wrong → move right = 1

Eventually, you'll reach partition1 = 1, partition2 = 2 → correct.

Then,

* maxLeft = max(1, 2) = 2
* minRight = min(3, 4) = 3
* (m + n = 5 → odd) → median = **maxLeft** = 3

✔️ Matches expectation!

## Time / Space Complexity

|  |  |  |
| --- | --- | --- |
| Metric | Value | Why |
| ⏱ Time | O(log(min(m,n))) | Binary search on smaller array |
| 💾 Space | O(1) | No extra space used |

## Alternate Approaches

|  |  |
| --- | --- |
| Approach | Notes |
| ❌ Merge & Find Median | O(m + n), simple but **not allowed** due to time constraint |
| ✅ Binary Partition | Optimal, uses binary search on one array to find correct partition |
| ❌ Brute Search | Try all combinations — too slow |