Leetcode problem number :04

Statement:-

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.

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

Explanation

आपको **दो sorted arrays** दिए गए हैं:

- nums1 (आकार = m)
- nums2 (आकार = n)

िोनों arrays **पहले से बढ़ते क्रम (ascending order)** में हैं।

– **आपका काम:**

इन िोनों arrays को ममलाकर (बिना सच में merge ककए) उनका **median** ननकालना है।

⛔ ध्यान िें:

Time Complexity **O(log (m+n))** होनी चादहए।

Simple merge करना allowed नहkीं है।

🔷 **Median क्या होता है?**

- अगर कुल elements **odd** हैं → िीच वाला element
- अगर कुल elements **even** हैं → िीच के िो elements का औसत

उदाहरण:

- [1, 2, 3] → median = 2
- [1, 2, 3, 4] → median = (2 + 3) / 2 = 2.5

Real code for paste in Leetcode

java

```
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 low = 0, high = m;

        while (low <= high) {

            int cut1 = (low + high) / 2;
            int cut2 = (m + n + 1) / 2 - cut1;

            int left1 = (cut1 == 0) ? Integer.MIN_VALUE : nums1[cut1 - 1];
            int left2 = (cut2 == 0) ? Integer.MIN_VALUE : nums2[cut2 - 1];

            int right1 = (cut1 == m) ? Integer.MAX_VALUE : nums1[cut1];
            int right2 = (cut2 == n) ? Integer.MAX_VALUE : nums2[cut2];

            if (left1 <= right2 && left2 <= right1) {

                if ((m + n) % 2 == 0)
                    return (Math.max(left1, left2) + Math.min(right1, right2)) / 2.0;
                else
                    return Math.max(left1, left2);
```

```
            }

            else if (left1 > right2)
                high = cut1 - 1;
            else
                low = cut1 + 1;
        }

        return 0.0;
    }
}
```

C++

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        // Get the sizes of both input arrays.
        int n = nums1.size();
        int m = nums2.size();

        // Merge the arrays into a single sorted array.
        vector<int> merged;
        for (int i = 0; i < n; i++) {
            merged.push_back(nums1[i]);
        }
        for (int i = 0; i < m; i++) {
```

```
            merged.push_back(nums2[i]);
        }

        // Sort the merged array.
        sort(merged.begin(), merged.end());

        // Calculate the total number of elements in the merged array.
        int total = merged.size();

        if (total % 2 == 1) {
            // If the total number of elements is odd, return the middle element as the median.
            return static_cast<double>(merged[total / 2]);
        } else {
            // If the total number of elements is even, calculate the average of the two middle elements as the median.
            int middle1 = merged[total / 2 - 1];
            int middle2 = merged[total / 2];
            return (static_cast<double>(middle1) + static_cast<double>(middle2)) / 2.0;
        }
    }
};
```

Python

```
class Solution:
    def findMedianSortedArrays(self, nums1, nums2):
        # Merge the arrays into a single sorted array.
        merged = nums1 + nums2

        # Sort the merged array.
        merged.sort()
```

```
    # Calculate the total number of elements in the merged array.
    total = len(merged)


    if total % 2 == 1:
        # If the total number of elements is odd, return the middle element as the median.
        return float(merged[total // 2])
    else:
        # If the total number of elements is even, calculate the average of the two middle elements as the median.
        middle1 = merged[total // 2 - 1]
        middle2 = merged[total // 2]
        return (float(middle1) + float(middle2)) / 2.0
```