10. 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)).

```
def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    m, n = len(nums1), len(nums2)
    total_length = m + n
    is_even = total_length % 2 == 0
    low, high = 0, m
    while low <= high:
        partition_nums1 = (low + high) // 2
        partition_nums2 = (total_length + 1) // 2 - partition_nums1
        max_left_nums1 = float('-inf') if partition_nums1 == 0 else nums1[partition_nums1 - 1]
        min_right_nums1 = float('inf') if partition_nums1 == m else nums1[partition_nums1]

        max_left_nums2 = float('-inf') if partition_nums2 == 0 else nums2[partition_nums2 - 1]
        min_right_nums2 = float('inf') if partition_nums2 == n else nums2[partition_nums2]
        if max_left_nums1 <= min_right_nums2 and max_left_nums2 <= min_right_nums1:
            if is_even:
                return (max(max_left_nums1, max_left_nums2) + min(min_right_nums1,
min_right_nums2)) / 2
            else:
                return max(max_left_nums1, max_left_nums2)
        elif max_left_nums1 > min_right_nums2:
            high = partition_nums1 - 1
        else:
            low = partition_nums1 + 1

    raise ValueError("Input arrays are not sorted or invalid.")


# Example usage:
nums1 = [1, 3]
```

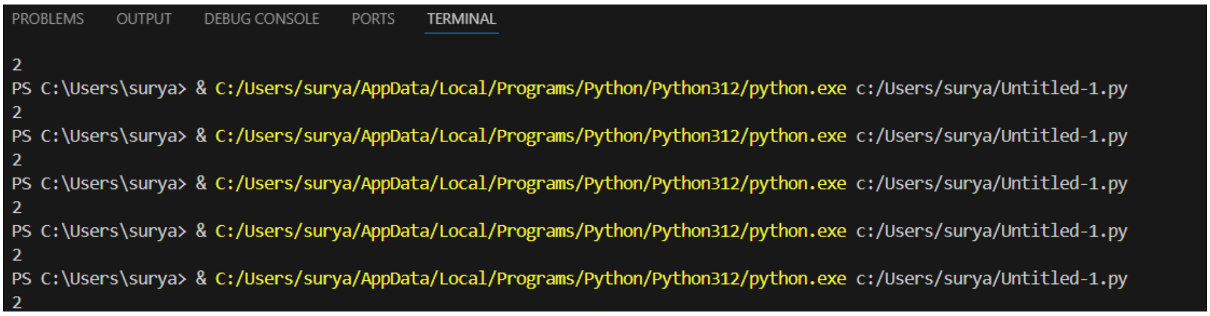
nums2 = [2]

print(findMedianSortedArrays(nums1, nums2)) # Output: 2.0

INPUT:

[1,3],[2]

OUTPUT:



TIME COMPLEXITY:

O(log (m+n))