Problem 4: Median of Two Sorted Arrays

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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

Code Snippets

C++:

```
class Solution {
public:
double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java:

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

}
}
```

Python3:

```
class Solution:
def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) ->
float:
```

Python:

```
class Solution(object):
def findMedianSortedArrays(self, nums1, nums2):
"""
```

```
:type nums1: List[int]
:type nums2: List[int]
:rtype: float
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var findMedianSortedArrays = function(nums1, nums2) {

};
```

TypeScript:

```
function findMedianSortedArrays(nums1: number[], nums2: number[]): number {

};
```

C#:

```
public class Solution {
public double FindMedianSortedArrays(int[] nums1, int[] nums2) {

}
}
```

C:

```
double findMedianSortedArrays(int* nums1, int nums1Size, int* nums2, int
nums2Size) {

}
```

Go:

```
func findMedianSortedArrays(nums1 []int, nums2 []int) float64 {

}
```

Kotlin:

```
class Solution {
fun findMedianSortedArrays(nums1: IntArray, nums2: IntArray): Double {

}
}
```

Swift:

```
class Solution {
func findMedianSortedArrays(_ nums1: [Int], _ nums2: [Int]) -> Double {

}
}
```

Rust:

```
impl Solution {
pub fn find_median_sorted_arrays(nums1: Vec<i32>, nums2: Vec<i32>) -> f64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Float}
def find_median_sorted_arrays(nums1, nums2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Float
*/
function findMedianSortedArrays($nums1, $nums2) {
```

```
}
}
```

Dart:

```
class Solution {
double findMedianSortedArrays(List<int> nums1, List<int> nums2) {

}
}
```

Scala:

```
object Solution {
def findMedianSortedArrays(nums1: Array[Int], nums2: Array[Int]): Double = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_median_sorted_arrays(nums1 :: [integer], nums2 :: [integer]) ::
float
def find_median_sorted_arrays(nums1, nums2) do

end
end
```

Erlang:

```
-spec find_median_sorted_arrays(Nums1 :: [integer()], Nums2 :: [integer()])
-> float().
find_median_sorted_arrays(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (find-median-sorted-arrays nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Median of Two Sorted Arrays
* Difficulty: Hard
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Median of Two Sorted Arrays
* Difficulty: Hard
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Median of Two Sorted Arrays
Difficulty: Hard
Tags: array, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) ->
float:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findMedianSortedArrays(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: float
"""
```

JavaScript Solution:

```
/**
* Problem: Median of Two Sorted Arrays
* Difficulty: Hard
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
```

```
*/
var findMedianSortedArrays = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Median of Two Sorted Arrays
* Difficulty: Hard
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function findMedianSortedArrays(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Median of Two Sorted Arrays
* Difficulty: Hard
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public double FindMedianSortedArrays(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Median of Two Sorted Arrays
* Difficulty: Hard
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

double findMedianSortedArrays(int* nums1, int nums1Size, int* nums2, int
nums2Size) {

}
```

Go Solution:

```
// Problem: Median of Two Sorted Arrays
// Difficulty: Hard
// Tags: array, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func findMedianSortedArrays(nums1 []int, nums2 []int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun findMedianSortedArrays(nums1: IntArray, nums2: IntArray): Double {

}
}
```

Swift Solution:

```
class Solution {
func findMedianSortedArrays(_ nums1: [Int], _ nums2: [Int]) -> Double {
```

```
}
}
```

Rust Solution:

```
// Problem: Median of Two Sorted Arrays
// Difficulty: Hard
// Tags: array, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn find_median_sorted_arrays(nums1: Vec<i32>, nums2: Vec<i32>) -> f64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Float}
def find_median_sorted_arrays(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Float
*/
function findMedianSortedArrays($nums1, $nums2) {

}
}
```

Dart Solution:

```
class Solution {
double findMedianSortedArrays(List<int> nums1, List<int> nums2) {

}
}
```

Scala Solution:

```
object Solution {
def findMedianSortedArrays(nums1: Array[Int], nums2: Array[Int]): Double = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_median_sorted_arrays(nums1 :: [integer], nums2 :: [integer]) ::
float
def find_median_sorted_arrays(nums1, nums2) do

end
end
```

Erlang Solution:

```
-spec find_median_sorted_arrays(Nums1 :: [integer()], Nums2 :: [integer()])
-> float().
find_median_sorted_arrays(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (find-median-sorted-arrays nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) flonum?)
)
```