Problem 1818: Minimum Absolute Sum Difference

Problem Information

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

Problem Description

You are given two positive integer arrays

nums1

and

nums2

, both of length

n

.

The

absolute sum difference

of arrays

nums1

and

nums2

is defined as the

sum

of

|nums1[i] - nums2[i]|

for each

0 <= i < n

(

0-indexed

).

You can replace

at most one

element of

nums1

with

any

other element in

nums1

to

minimize

the absolute sum difference.

Return the

minimum absolute sum difference

after

replacing at most one

element in the array

nums1

.

Since the answer may be large, return it

modulo

10

9

+ 7

.

|x|

is defined as:

x

if

x >= 0

, or

-x

if

x < 0

.

Example 1:

Input:

nums1 = [1,7,5], nums2 = [2,3,5]

Output:

3

Explanation:

There are two possible optimal solutions: - Replace the second element with the first: [1,

7

,5] => [1,

1

,5], or - Replace the second element with the third: [1,

7

,5] => [1,

5

,5]. Both will yield an absolute sum difference of

|1-2| + (|1-3| or |5-3|) + |5-5| =

3.

Example 2:

Input:

nums1 = [2,4,6,8,10], nums2 = [2,4,6,8,10]

Output:

0

Explanation:

nums1 is equal to nums2 so no replacement is needed. This will result in an absolute sum difference of 0.

Example 3:

Input:

nums1 = [1,10,4,4,2,7], nums2 = [9,3,5,1,7,4]

Output:

20

Explanation:

Replace the first element with the second: [

1

,10,4,4,2,7] => [

10

,10,4,4,2,7]. This yields an absolute sum difference of

|10-9| + |10-3| + |4-5| + |4-1| + |2-7| + |7-4| = 20

Constraints:

n == nums1.length

n == nums2.length

1 <= n <= 10

5

1 <= nums1[i], nums2[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minAbsoluteSumDiff(int* nums1, int nums1Size, int* nums2, int nums2Size)
{
```

```
}
```

Go:

```
func minAbsoluteSumDiff(nums1 []int, nums2 []int) int {

}
```

Kotlin:

```
class Solution {
fun minAbsoluteSumDiff(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

```
.
```

Racket:

```
(define/contract (min-absolute-sum-diff nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Absolute Sum Difference
* Difficulty: Medium
* 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:
int minAbsoluteSumDiff(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Absolute Sum Difference
* Difficulty: Medium
* 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 int minAbsoluteSumDiff(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Absolute Sum Difference
Difficulty: Medium
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 minAbsoluteSumDiff(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Absolute Sum Difference
* Difficulty: Medium
* 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 minAbsoluteSumDiff = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Absolute Sum Difference
* Difficulty: Medium
* 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 minAbsoluteSumDiff(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Absolute Sum Difference
* Difficulty: Medium
* 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 int MinAbsoluteSumDiff(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Absolute Sum Difference
* Difficulty: Medium
* 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
*/

int minAbsoluteSumDiff(int* nums1, int nums1Size, int* nums2, int nums2Size)
{

}
```

Go Solution:

```
// Problem: Minimum Absolute Sum Difference
// Difficulty: Medium
// 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 minAbsoluteSumDiff(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minAbsoluteSumDiff(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Absolute Sum Difference
// Difficulty: Medium
// 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 min_absolute_sum_diff(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

```
min_absolute_sum_diff(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (min-absolute-sum-diff nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```