Problem 801: Minimum Swaps To Make Sequences Increasing

Problem Information

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

Problem Description

You are given two integer arrays of the same length

nums1

and

nums2

. In one operation, you are allowed to swap

nums1[i]

with

nums2[i]

.

For example, if

nums1 = [1,2,3,

8

]

, and

nums2 = [5,6,7,

4

]

, you can swap the element at

i = 3

to obtain

nums1 = [1,2,3,4]

and

nums2 = [5,6,7,8]

.

Return

the minimum number of needed operations to make

nums1

and

nums2

strictly increasing

. The test cases are generated so that the given input always makes it possible.

An array

arr

is

strictly increasing

if and only if

arr[0] < arr[1] < arr[2] < ... < arr[arr.length - 1]

.

Example 1:

Input:

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

Output:

1

Explanation:

Swap nums1[3] and nums2[3]. Then the sequences are: nums1 = [1, 3, 5, 7] and nums2 = [1, 2, 3, 4] which are both strictly increasing.

Example 2:

Input:

nums1 = [0,3,5,8,9], nums2 = [2,1,4,6,9]

Output:

1

Constraints:

2 <= nums1.length <= 10

5

nums2.length == nums1.length

0 <= nums1[i], nums2[i] <= 2 * 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}

}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-swap nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Swaps To Make Sequences Increasing
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Swaps To Make Sequences Increasing
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Swaps To Make Sequences Increasing
Difficulty: Hard
Tags: array, dp
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Swaps To Make Sequences Increasing
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Swaps To Make Sequences Increasing
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Minimum Swaps To Make Sequences Increasing
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Minimum Swaps To Make Sequences Increasing
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Minimum Swaps To Make Sequences Increasing
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minSwap(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Swaps To Make Sequences Increasing
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-swap nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```