Problem 2321: Maximum Score Of Spliced Array

Problem Information

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

Problem Description

You are given two

0-indexed

integer arrays

nums1

and

nums2

, both of length

n

.

You can choose two integers

left

and

right

where

0 <= left <= right < n

and

swap

the subarray

nums1[left...right]

with the subarray

nums2[left...right]

.

For example, if

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

and

nums2 = [11,12,13,14,15]

and you choose

left = 1

and

right = 2

,

nums1

becomes

[1,

12,13

,4,5]

and

nums2

becomes

[11,

2,3

,14,15]

.

You may choose to apply the mentioned operation

once

or not do anything.

The

score

of the arrays is the

maximum

of

sum(nums1)

and

sum(nums2)

, where

sum(arr)

is the sum of all the elements in the array

arr

.

Return

the

maximum possible score

.

A

subarray

is a contiguous sequence of elements within an array.

arr[left...right]

denotes the subarray that contains the elements of

nums

between indices

left

and

right

(

inclusive

).

Example 1:

Input:

nums1 = [60,60,60], nums2 = [10,90,10]

Output:

210

Explanation:

Choosing left = 1 and right = 1, we have nums1 = [60,

90

,60] and nums2 = [10,

60

,10]. The score is max(sum(nums1), sum(nums2)) = max(210, 80) = 210.

Example 2:

Input:

nums1 = [20,40,20,70,30], nums2 = [50,20,50,40,20]

Output:

220

Explanation:

Choosing left = 3, right = 4, we have nums1 = [20,40,20,

40,20

] and nums2 = [50,20,50,

70,30

]. The score is max(sum(nums1), sum(nums2)) = max(140, 220) = 220.

Example 3:

Input:

nums1 = [7,11,13], nums2 = [1,1,1]

Output:

31

Explanation:

We choose not to swap any subarray. The score is max(sum(nums1), sum(nums2)) = max(31, 3) = 31.

Constraints:

n == nums1.length == nums2.length

1 <= n <= 10

5

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

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
 * @return {number}
 */
var maximumsSplicedArray = function(nums1, nums2) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximums-spliced-array nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Score Of Spliced Array
* 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 maximumsSplicedArray(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Score Of Spliced Array
* 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 maximumsSplicedArray(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Score Of Spliced Array
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 maximumsSplicedArray(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Score Of Spliced Array
* 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 maximumsSplicedArray = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Score Of Spliced Array
* 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 maximumsSplicedArray(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Score Of Spliced Array
* 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 MaximumsSplicedArray(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Score Of Spliced Array
* 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 maximumsSplicedArray(int* nums1, int nums1Size, int* nums2, int
nums2Size) {

}
```

Go Solution:

```
// Problem: Maximum Score Of Spliced Array
// 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 maximumsSplicedArray(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Score Of Spliced Array
// 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 maximums_spliced_array(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
```

```
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximums-spliced-array nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```