Problem 1187: Make Array Strictly Increasing

Problem Information

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

Problem Description

Given two integer arrays

arr1

and

arr2

, return the minimum number of operations (possibly zero) needed to make

arr1

strictly increasing.

In one operation, you can choose two indices

0 <= i < arr1.length

and

0 <= j < arr2.length

and do the assignment

arr1[i] = arr2[j]

.

If there is no way to make

arr1

strictly increasing, return

-1

.

Example 1:

Input:

arr1 = [1,5,3,6,7], arr2 = [1,3,2,4]

Output:

1

Explanation:

Replace

5

with

2

, then

arr1 = [1, 2, 3, 6, 7]

.

Example 2:

Input:

arr1 = [1,5,3,6,7], arr2 = [4,3,1]

Output:

2

Explanation:

Replace

5

with

3

and then replace

3

with

4

.

arr1 = [1, 3, 4, 6, 7]

.

Example 3:

Input:

arr1 = [1,5,3,6,7], arr2 = [1,6,3,3]

Output:

-1

Explanation:

You can't make

arr1

strictly increasing.

Constraints:

1 <= arr1.length, arr2.length <= 2000

0 <= arr1[i], arr2[i] <= 10^9

Code Snippets

C++:

```
class Solution {
public:
int makeArrayIncreasing(vector<int>& arr1, vector<int>& arr2) {

}
};
```

Java:

```
class Solution {
public int makeArrayIncreasing(int[] arr1, int[] arr2) {

}
}
```

Python3:

```
class Solution:
def makeArrayIncreasing(self, arr1: List[int], arr2: List[int]) -> int:
```

Python:

```
class Solution(object):
def makeArrayIncreasing(self, arr1, arr2):
"""
:type arr1: List[int]
:type arr2: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} arr1
* @param {number[]} arr2
* @return {number}
*/
var makeArrayIncreasing = function(arr1, arr2) {

};
```

TypeScript:

```
function makeArrayIncreasing(arr1: number[], arr2: number[]): number {

};
```

C#:

```
public class Solution {
public int MakeArrayIncreasing(int[] arr1, int[] arr2) {

}
}
```

C:

```
int makeArrayIncreasing(int* arr1, int arr1Size, int* arr2, int arr2Size) {

}
```

Go:

```
func makeArrayIncreasing(arr1 []int, arr2 []int) int {

}
```

Kotlin:

```
class Solution {
fun makeArrayIncreasing(arr1: IntArray, arr2: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func makeArrayIncreasing(_ arr1: [Int], _ arr2: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn make_array_increasing(arr1: Vec<i32>, arr2: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr1
# @param {Integer[]} arr2
# @return {Integer}
def make_array_increasing(arr1, arr2)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer[] $arr1
* @param Integer[] $arr2
* @return Integer
*/
function makeArrayIncreasing($arr1, $arr2) {

}
}
```

Dart:

```
class Solution {
int makeArrayIncreasing(List<int> arr1, List<int> arr2) {

}
}
```

Scala:

```
object Solution {
def makeArrayIncreasing(arr1: Array[Int], arr2: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec make_array_increasing(arr1 :: [integer], arr2 :: [integer]) :: integer
def make_array_increasing(arr1, arr2) do

end
end
```

Erlang:

```
-spec make_array_increasing(Arr1 :: [integer()], Arr2 :: [integer()]) ->
integer().
make_array_increasing(Arr1, Arr2) ->
.
```

Racket:

```
(define/contract (make-array-increasing arr1 arr2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make Array Strictly Increasing
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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 makeArrayIncreasing(vector<int>& arr1, vector<int>& arr2) {

}
};
```

Java Solution:

```
/**
* Problem: Make Array Strictly Increasing
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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 makeArrayIncreasing(int[] arr1, int[] arr2) {
}
}
```

Python3 Solution:

```
"""
Problem: Make Array Strictly Increasing
Difficulty: Hard
Tags: array, dp, sort, search

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 makeArrayIncreasing(self, arr1: List[int], arr2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def makeArrayIncreasing(self, arr1, arr2):
"""
:type arr1: List[int]
:type arr2: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Make Array Strictly Increasing
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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[]} arr1
* @param {number[]} arr2
* @return {number}
*/
var makeArrayIncreasing = function(arr1, arr2) {

};
```

TypeScript Solution:

```
/**
* Problem: Make Array Strictly Increasing
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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 makeArrayIncreasing(arr1: number[], arr2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Make Array Strictly Increasing
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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 MakeArrayIncreasing(int[] arr1, int[] arr2) {
}
}
```

C Solution:

```
/*
* Problem: Make Array Strictly Increasing
* Difficulty: Hard
* Tags: array, dp, sort, search
*
* 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 makeArrayIncreasing(int* arr1, int arr1Size, int* arr2, int arr2Size) {

}
```

Go Solution:

```
// Problem: Make Array Strictly Increasing
// Difficulty: Hard
// Tags: array, dp, sort, search
//
// 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 makeArrayIncreasing(arr1 []int, arr2 []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun makeArrayIncreasing(arr1: IntArray, arr2: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func makeArrayIncreasing(_ arr1: [Int], _ arr2: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Make Array Strictly Increasing
// Difficulty: Hard
// Tags: array, dp, sort, search
//
// 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 make_array_increasing(arr1: Vec<i32>, arr2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr1
# @param {Integer[]} arr2
# @return {Integer}
def make_array_increasing(arr1, arr2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr1
* @param Integer[] $arr2
* @return Integer
*/
function makeArrayIncreasing($arr1, $arr2) {

}
}
```

Dart Solution:

```
class Solution {
int makeArrayIncreasing(List<int> arr1, List<int> arr2) {

}
}
```

Scala Solution:

```
object Solution {
def makeArrayIncreasing(arr1: Array[Int], arr2: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec make_array_increasing(arr1 :: [integer], arr2 :: [integer]) :: integer
def make_array_increasing(arr1, arr2) do

end
end
```

Erlang Solution:

```
-spec make_array_increasing(Arr1 :: [integer()], Arr2 :: [integer()]) ->
integer().
make_array_increasing(Arr1, Arr2) ->
.
```

Racket Solution:

```
(define/contract (make-array-increasing arr1 arr2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)

)
```