Problem 1035: Uncrossed Lines

Problem Information

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

Problem Description

You are given two integer arrays

nums1

and

nums2

. We write the integers of

nums1

and

nums2

(in the order they are given) on two separate horizontal lines.

We may draw connecting lines: a straight line connecting two numbers

nums1[i]

and

nums2[j]

such that:

nums1[i] == nums2[j]

, and

the line we draw does not intersect any other connecting (non-horizontal) line.

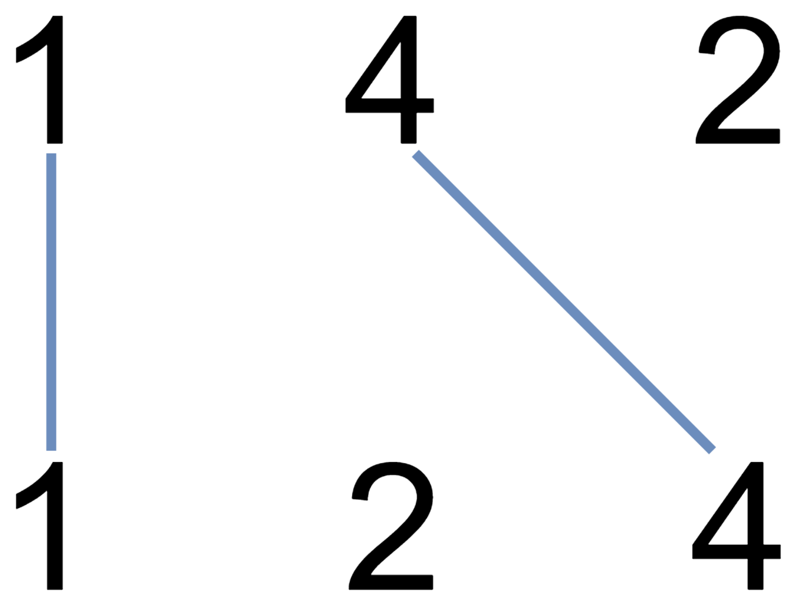
Note that a connecting line cannot intersect even at the endpoints (i.e., each number can only belong to one connecting line).

Return

the maximum number of connecting lines we can draw in this way

.

Example 1:



Input:

nums1 = [1,4,2], nums2 = [1,2,4]

Output:

2

Explanation:

We can draw 2 uncrossed lines as in the diagram. We cannot draw 3 uncrossed lines, because the line from nums1[1] = 4 to nums2[2] = 4 will intersect the line from nums1[2]=2 to nums2[1]=2.

Example 2:

Input:

nums1 = [2,5,1,2,5], nums2 = [10,5,2,1,5,2]

Output:

3

Example 3:

Input:

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

Output:

2

Constraints:

1 <= nums1.length, nums2.length <= 500

1 <= nums1[i], nums2[j] <= 2000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

}

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-uncrossed-lines nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Uncrossed Lines
* Difficulty: Medium
* 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 maxUncrossedLines(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Uncrossed Lines
* Difficulty: Medium
* 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 maxUncrossedLines(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Uncrossed Lines
Difficulty: Medium
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 maxUncrossedLines(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Uncrossed Lines
* Difficulty: Medium
* 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 maxUncrossedLines = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Uncrossed Lines
* Difficulty: Medium
* 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 maxUncrossedLines(nums1: number[], nums2: number[]): number {
};
```

C# Solution:

```
/*
* Problem: Uncrossed Lines
* Difficulty: Medium
* 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 MaxUncrossedLines(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Uncrossed Lines
* Difficulty: Medium
* 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 maxUncrossedLines(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go Solution:

```
// Problem: Uncrossed Lines
// Difficulty: Medium
// 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 maxUncrossedLines(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Uncrossed Lines
// Difficulty: Medium
// 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 max_uncrossed_lines(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-uncrossed-lines nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```