# Problem 2771: Longest Non-decreasing Subarray From Two Arrays

## Problem Information

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

## Problem Description

You are given two

0-indexed

integer arrays

nums1

and

nums2

of length

n

.

Let's define another

0-indexed

integer array,

nums3

, of length

n

. For each index

i

in the range

[0, n - 1]

, you can assign either

nums1[i]

or

nums2[i]

to

nums3[i]

.

Your task is to maximize the length of the

longest non-decreasing subarray

in

nums3

by choosing its values optimally.

Return

an integer representing the length of the

longest non-decreasing

subarray in

nums3

.

Note:

A

subarray

is a contiguous

non-empty

sequence of elements within an array.

Example 1:

Input:

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

Output:

2

Explanation:

One way to construct nums3 is: nums3 = [nums1[0], nums2[1], nums2[2]] => [2,2,1]. The subarray starting from index 0 and ending at index 1, [2,2], forms a non-decreasing subarray of length 2. We can show that 2 is the maximum achievable length.

Example 2:

Input:

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

Output:

4

Explanation:

One way to construct nums3 is: nums3 = [nums1[0], nums2[1], nums2[2], nums2[3]] => [1,2,3,4]. The entire array forms a non-decreasing subarray of length 4, making it the maximum achievable length.

Example 3:

Input:

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

Output:

2

Explanation:

One way to construct nums3 is: nums3 = [nums1[0], nums1[1]] => [1,1]. The entire array forms a non-decreasing subarray of length 2, making it the maximum achievable length.

Constraints:

1 <= nums1.length == nums2.length == n <= 10

5

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

9

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

```
};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

```
}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

### Scala:

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

}
}
```

### Elixir:

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

end
end
```

### Erlang:

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

### Racket:

```
(define/contract (max-non-decreasing-length nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Longest Non-decreasing Subarray From Two Arrays
* 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 maxNonDecreasingLength(vector<int>& nums1, vector<int>& nums2) {

}
};
```

### Java Solution:

```
/**
* Problem: Longest Non-decreasing Subarray From Two Arrays
* 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 maxNonDecreasingLength(int[] nums1, int[] nums2) {

}
}
```

### Python3 Solution:

```
"""
Problem: Longest Non-decreasing Subarray From Two Arrays
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 maxNonDecreasingLength(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Longest Non-decreasing Subarray From Two Arrays
* 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 maxNonDecreasingLength = function(nums1, nums2) {

};
```

### TypeScript Solution:

```
/**
* Problem: Longest Non-decreasing Subarray From Two Arrays
* 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 maxNonDecreasingLength(nums1: number[], nums2: number[]): number {

};
```

### C# Solution:

```
/*
* Problem: Longest Non-decreasing Subarray From Two Arrays
* 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 MaxNonDecreasingLength(int[] nums1, int[] nums2) {

}
}
```

### C Solution:

```
/*
* Problem: Longest Non-decreasing Subarray From Two Arrays
* 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 maxNonDecreasingLength(int* nums1, int nums1Size, int* nums2, int
nums2Size) {

}
```

### Go Solution:

```
// Problem: Longest Non-decreasing Subarray From Two Arrays
// 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 maxNonDecreasingLength(nums1 []int, nums2 []int) int {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Longest Non-decreasing Subarray From Two Arrays
// 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_non_decreasing_length(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

}
}
```

### Dart Solution:

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

}
}
```

### Scala Solution:

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

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (max-non-decreasing-length nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```