# Problem 1855: Maximum Distance Between a Pair of Values

## Problem Information

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

## Problem Description

You are given two

non-increasing 0-indexed

integer arrays

nums1

and

nums2

.

A pair of indices

(i, j)

, where

0 <= i < nums1.length

and

0 <= j < nums2.length

, is

valid

if both

i <= j

and

nums1[i] <= nums2[j]

. The

distance

of the pair is

j - i

.

Return

the

maximum distance

of any

valid

pair

(i, j)

. If there are no valid pairs, return

0

.

An array

arr

is

non-increasing

if

arr[i-1] >= arr[i]

for every

1 <= i < arr.length

.

Example 1:

Input:

nums1 = [55,30,5,4,2], nums2 = [100,20,10,10,5]

Output:

2

Explanation:

The valid pairs are (0,0), (2,2), (2,3), (2,4), (3,3), (3,4), and (4,4). The maximum distance is 2 with pair (2,4).

Example 2:

Input:

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

Output:

1

Explanation:

The valid pairs are (0,0), (0,1), and (1,1). The maximum distance is 1 with pair (0,1).

Example 3:

Input:

nums1 = [30,29,19,5], nums2 = [25,25,25,25,25]

Output:

2

Explanation:

The valid pairs are (2,2), (2,3), (2,4), (3,3), and (3,4). The maximum distance is 2 with pair (2,4).

Constraints:

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

5

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

5

Both

nums1

and

nums2

are

non-increasing

.

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

```
class Solution(object):
def maxDistance(self, nums1, nums2):
```

```
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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


}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

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

## Solutions

### C++ Solution:

```
/*
* Problem: Maximum Distance Between a Pair of Values
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

### Java Solution:

```
/**
* Problem: Maximum Distance Between a Pair of Values
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

### Python3 Solution:

```
"""
Problem: Maximum Distance Between a Pair of Values
```

```
Difficulty: Medium
Tags: array, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Maximum Distance Between a Pair of Values
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

**TypeScript Solution:**

```
/**
* Problem: Maximum Distance Between a Pair of Values
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

**C# Solution:**

```
/*
* Problem: Maximum Distance Between a Pair of Values
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

**C Solution:**

```
/*
* Problem: Maximum Distance Between a Pair of Values
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
```

**Go Solution:**

```
// Problem: Maximum Distance Between a Pair of Values
// Difficulty: Medium
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
```

**Kotlin Solution:**

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

}
}
```

**Swift Solution:**

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

}
}
```

## Rust Solution:

```
// Problem: Maximum Distance Between a Pair of Values
// Difficulty: Medium
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

## Ruby Solution:

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

end
```

## PHP Solution:

```
class Solution {

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

}
}
```

## Dart Solution:

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

}
}
```

## Scala Solution:

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

}
}
```

## Elixir Solution:

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

end
end
```

## Erlang Solution:

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

## Racket Solution:

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