Problem 870: Advantage Shuffle

Problem Information

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

Problem Description

You are given two integer arrays

nums1

and

nums2

both of the same length. The

advantage

of

nums1

with respect to

nums2

is the number of indices

i

for which

nums1[i] > nums2[i]

.

Return

any permutation of

nums1

that maximizes its

advantage

with respect to

nums2

.

Example 1:

Input:

nums1 = [2,7,11,15], nums2 = [1,10,4,11]

Output:

[2,11,7,15]

Example 2:

Input:

nums1 = [12,24,8,32], nums2 = [13,25,32,11]

Output:

[24,32,8,12]

Constraints:

1 <= nums1.length <= 10

5

nums2.length == nums1.length

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

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* advantageCount(int* nums1, int nums1Size, int* nums2, int nums2Size,
int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (advantage-count nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Advantage Shuffle
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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:
vector<int> advantageCount(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Advantage Shuffle
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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[] advantageCount(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Advantage Shuffle
Difficulty: Medium
Tags: array, greedy, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Advantage Shuffle
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 advantageCount = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Advantage Shuffle
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 advantageCount(nums1: number[], nums2: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Advantage Shuffle
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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[] AdvantageCount(int[] nums1, int[] nums2) {

}

}
```

C Solution:

```
/*
* Problem: Advantage Shuffle
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* advantageCount(int* nums1, int nums1Size, int* nums2, int nums2Size,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Advantage Shuffle
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 advantageCount(nums1 []int, nums2 []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Advantage Shuffle
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 advantage_count(nums1: Vec<i32>, nums2: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (advantage-count nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```