Problem 775: Global and Local Inversions

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

which represents a permutation of all the integers in the range

[0, n - 1]

.

The number of

global inversions

is the number of the different pairs

(i, j)

where:

0 <= i < j < n

nums[i] > nums[j]

The number of

local inversions

is the number of indices

i

where:

0 <= i < n - 1

nums[i] > nums[i + 1]

Return

true

if the number of

global inversions

is equal to the number of

local inversions

.

Example 1:

Input:

nums = [1,0,2]

Output:

true

Explanation:

There is 1 global inversion and 1 local inversion.

Example 2:

Input:

nums = [1,2,0]

Output:

false

Explanation:

There are 2 global inversions and 1 local inversion.

Constraints:

n == nums.length

1 <= n <= 10

5

0 <= nums[i] < n

All the integers of

nums

are

unique

.

nums

is a permutation of all the numbers in the range

[0, n - 1]

.

Code Snippets

C++:

```
class Solution {
public:
bool isIdealPermutation(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public boolean isIdealPermutation(int[] nums) {

}
}
```

Python3:

```
class Solution:
def isIdealPermutation(self, nums: List[int]) -> bool:
```

Python:

```
class Solution(object):
def isIdealPermutation(self, nums):
"""
:type nums: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {boolean}
*/
var isIdealPermutation = function(nums) {

};
```

TypeScript:

```
function isIdealPermutation(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool IsIdealPermutation(int[] nums) {

}
}
```

C:

```
bool isIdealPermutation(int* nums, int numsSize) {

}
```

Go:

```
func isIdealPermutation(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun isIdealPermutation(nums: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func isIdealPermutation(_ nums: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_ideal_permutation(nums: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Boolean}
def is_ideal_permutation(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Boolean
*/
function isIdealPermutation($nums) {

}
}
```

Dart:

```
class Solution {
bool isIdealPermutation(List<int> nums) {

}
```

```
}
```

Scala:

```
object Solution {
def isIdealPermutation(nums: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_ideal_permutation(nums :: [integer]) :: boolean
def is_ideal_permutation(nums) do

end
end
```

Erlang:

```
-spec is_ideal_permutation(Nums :: [integer()]) -> boolean().
is_ideal_permutation(Nums) ->
.
```

Racket:

```
(define/contract (is-ideal-permutation nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Global and Local Inversions
* Difficulty: Medium
* Tags: array, math
*
```

```
* 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:
bool isIdealPermutation(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Global and Local Inversions
* Difficulty: Medium
* Tags: array, math
*
* 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 boolean isIdealPermutation(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Global and Local Inversions
Difficulty: Medium
Tags: array, math

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 isIdealPermutation(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isIdealPermutation(self, nums):
"""
:type nums: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Global and Local Inversions
* Difficulty: Medium
* Tags: array, math
*
* 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[]} nums
* @return {boolean}
*/
var isIdealPermutation = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Global and Local Inversions
* Difficulty: Medium
* Tags: array, math
```

```
*
* 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 isIdealPermutation(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Global and Local Inversions
* Difficulty: Medium
* Tags: array, math
*
* 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 bool IsIdealPermutation(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Global and Local Inversions
* Difficulty: Medium
* Tags: array, math
*
* 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
*/

bool isIdealPermutation(int* nums, int numsSize) {
```

```
}
```

Go Solution:

```
// Problem: Global and Local Inversions
// Difficulty: Medium
// Tags: array, math
//
// 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 isIdealPermutation(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isIdealPermutation(nums: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isIdealPermutation(_ nums: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Global and Local Inversions
// Difficulty: Medium
// Tags: array, math
//
// 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 is_ideal_permutation(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Boolean}
def is_ideal_permutation(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Boolean
*/
function isIdealPermutation($nums) {

}
}
```

Dart Solution:

```
class Solution {
bool isIdealPermutation(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def isIdealPermutation(nums: Array[Int]): Boolean = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_ideal_permutation(nums :: [integer]) :: boolean
def is_ideal_permutation(nums) do

end
end
```

Erlang Solution:

```
-spec is_ideal_permutation(Nums :: [integer()]) -> boolean().
is_ideal_permutation(Nums) ->
.
```

Racket Solution:

```
(define/contract (is-ideal-permutation nums)
(-> (listof exact-integer?) boolean?)
)
```