Problem 2855: Minimum Right Shifts to Sort the Array

Problem Information

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

Problem Description

You are given a

0-indexed

array

nums

of length

n

containing

distinct

positive integers. Return

the

minimum

number of

right shifts

required to sort

nums

and

-1

if this is not possible.

A

right shift

is defined as shifting the element at index

i

to index

(i + 1) % n

, for all indices.

Example 1:

Input:

nums = [3,4,5,1,2]

Output:

2

Explanation:

After the first right shift, nums = [2,3,4,5,1]. After the second right shift, nums = [1,2,3,4,5]. Now nums is sorted; therefore the answer is 2.

Example 2:

Input:

nums = [1,3,5]

Output:

0

Explanation:

nums is already sorted therefore, the answer is 0.

Example 3:

Input:

nums = [2,1,4]

Output:

-1

Explanation:

It's impossible to sort the array using right shifts.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

nums

contains distinct integers.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int minimumRightShifts(List<Integer> nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumRightShifts(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int MinimumRightShifts(IList<int> nums) {

}
}
```

C:

```
int minimumRightShifts(int* nums, int numsSize) {

}
```

Go:

```
func minimumRightShifts(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun minimumRightShifts(nums: List<Int>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int minimumRightShifts(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def minimumRightShifts(nums: List[Int]): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-right-shifts nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Right Shifts to Sort the Array
* Difficulty: Easy
* Tags: array, 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 minimumRightShifts(vector<int>& nums) {
}
};
```

Java Solution:

```
/**
* Problem: Minimum Right Shifts to Sort the Array
* Difficulty: Easy
* Tags: array, 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 minimumRightShifts(List<Integer> nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Right Shifts to Sort the Array
Difficulty: Easy
Tags: array, 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 minimumRightShifts(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Right Shifts to Sort the Array
* Difficulty: Easy
* Tags: array, 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[]} nums
* @return {number}
*/
var minimumRightShifts = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Right Shifts to Sort the Array
* Difficulty: Easy
* Tags: array, 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 minimumRightShifts(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Right Shifts to Sort the Array
* Difficulty: Easy
* Tags: array, 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 MinimumRightShifts(IList<int> nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Right Shifts to Sort the Array
* Difficulty: Easy
* Tags: array, 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
*/

int minimumRightShifts(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Right Shifts to Sort the Array
// Difficulty: Easy
// Tags: array, 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 minimumRightShifts(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumRightShifts(nums: List<Int>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Right Shifts to Sort the Array
// Difficulty: Easy
// Tags: array, 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 minimum_right_shifts(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int minimumRightShifts(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def minimumRightShifts(nums: List[Int]): Int = {

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-right-shifts nums)
(-> (listof exact-integer?) exact-integer?)
)
```