Problem 769: Max Chunks To Make Sorted

Problem Information

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

Problem Description

You are given an integer array

arr

of length

n

that represents a permutation of the integers in the range

[0, n - 1]

.

We split

arr

into some number of

chunks

(i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.

Return

the largest number of chunks we can make to sort the array

.

Example 1:

Input:

arr = [4,3,2,1,0]

Output:

1

Explanation:

Splitting into two or more chunks will not return the required result. For example, splitting into [4, 3], [2, 1, 0] will result in [3, 4, 0, 1, 2], which isn't sorted.

Example 2:

Input:

arr = [1,0,2,3,4]

Output:

4

Explanation:

We can split into two chunks, such as [1, 0], [2, 3, 4]. However, splitting into [1, 0], [2], [3], [4] is the highest number of chunks possible.

Constraints:

n == arr.length

1 <= n <= 10

0 <= arr[i] < n

All the elements of

arr

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int maxChunksToSorted(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public int maxChunksToSorted(int[] arr) {

}
}
```

Python3:

```
class Solution:
def maxChunksToSorted(self, arr: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxChunksToSorted(self, arr):
```

```
"""
:type arr: List[int]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function maxChunksToSorted(arr: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxChunksToSorted(int[] arr) {

}
}
```

C:

```
int maxChunksToSorted(int* arr, int arrSize) {

}
```

Go:

```
func maxChunksToSorted(arr []int) int {

}
```

Kotlin:

```
class Solution {
fun maxChunksToSorted(arr: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxChunksToSorted(_ arr: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_chunks_to_sorted(arr: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer}
def max_chunks_to_sorted(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function maxChunksToSorted($arr) {

}
}
```

Dart:

```
class Solution {
int maxChunksToSorted(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def maxChunksToSorted(arr: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_chunks_to_sorted(arr :: [integer]) :: integer
def max_chunks_to_sorted(arr) do

end
end
```

Erlang:

```
-spec max_chunks_to_sorted(Arr :: [integer()]) -> integer().
max_chunks_to_sorted(Arr) ->
.
```

Racket:

```
(define/contract (max-chunks-to-sorted arr)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Max Chunks To Make Sorted
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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 maxChunksToSorted(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Max Chunks To Make Sorted
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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 maxChunksToSorted(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Max Chunks To Make Sorted
Difficulty: Medium
Tags: array, greedy, sort, stack
```

```
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 maxChunksToSorted(self, arr: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Max Chunks To Make Sorted
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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[]} arr
* @return {number}
*/
var maxChunksToSorted = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Max Chunks To Make Sorted
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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 maxChunksToSorted(arr: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Max Chunks To Make Sorted
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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 MaxChunksToSorted(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Max Chunks To Make Sorted
* Difficulty: Medium
* Tags: array, greedy, sort, stack
*
* 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 maxChunksToSorted(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Max Chunks To Make Sorted
// Difficulty: Medium
// Tags: array, greedy, sort, stack
//
// 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 maxChunksToSorted(arr []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxChunksToSorted(arr: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxChunksToSorted(_ arr: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Max Chunks To Make Sorted
// Difficulty: Medium
// Tags: array, greedy, sort, stack
```

```
//
// 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_chunks_to_sorted(arr: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer}
def max_chunks_to_sorted(arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function maxChunksToSorted($arr) {

}
}
```

Dart Solution:

```
class Solution {
int maxChunksToSorted(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def maxChunksToSorted(arr: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_chunks_to_sorted(arr :: [integer]) :: integer
def max_chunks_to_sorted(arr) do

end
end
```

Erlang Solution:

```
-spec max_chunks_to_sorted(Arr :: [integer()]) -> integer().
max_chunks_to_sorted(Arr) ->
.
```

Racket Solution:

```
(define/contract (max-chunks-to-sorted arr)
(-> (listof exact-integer?) exact-integer?)
)
```