Problem 581: Shortest Unsorted Continuous Subarray

Problem Information

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

Problem Description

Given an integer array

nums

, you need to find one

continuous subarray

such that if you only sort this subarray in non-decreasing order, then the whole array will be sorted in non-decreasing order.

Return

the shortest such subarray and output its length

.

Example 1:

Input:

nums = [2,6,4,8,10,9,15]

Output:

5

Explanation:

You need to sort [6, 4, 8, 10, 9] in ascending order to make the whole array sorted in ascending order.

Example 2:

Input:

nums = [1,2,3,4]

Output:

0

Example 3:

Input:

nums = [1]

Output:

0

Constraints:

1 <= nums.length <= 10

4

-10

5

<= nums[i] <= 10

5

Follow up:

Can you solve it in

O(n)

time complexity?

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int findUnsortedSubarray(int[] nums) {

}
}
```

Python3:

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

Python:

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

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int FindUnsortedSubarray(int[] nums) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun findUnsortedSubarray(nums: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-unsorted-subarray nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Unsorted Continuous Subarray
* 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 findUnsortedSubarray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Unsorted Continuous Subarray
* 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 findUnsortedSubarray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Unsorted Continuous Subarray
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 findUnsortedSubarray(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Unsorted Continuous Subarray
* 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[]} nums
* @return {number}
*/
var findUnsortedSubarray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Unsorted Continuous Subarray
* 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 findUnsortedSubarray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Shortest Unsorted Continuous Subarray
* 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 FindUnsortedSubarray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Unsorted Continuous Subarray
* 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 findUnsortedSubarray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Shortest Unsorted Continuous Subarray
// 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 findUnsortedSubarray(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findUnsortedSubarray(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shortest Unsorted Continuous Subarray
// 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 find_unsorted_subarray(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-unsorted-subarray nums)
(-> (listof exact-integer?) exact-integer?)
)
```