Problem 3555: Smallest Subarray to Sort in Every Sliding Window

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

.

For each contiguous

subarray

of length

k

, determine the

minimum

length of a continuous segment that must be sorted so that the entire window becomes

non■decreasing

; if the window is already sorted, its required length is zero.

Return an array of length

n – k + 1

where each element corresponds to the answer for its window.

Example 1:

Input:

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

Output:

[2,2,0]

Explanation:

nums[0...2] = [1, 3, 2]

. Sort

[3, 2]

to get

[1, 2, 3]

, the answer is 2.

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

. Sort

[3, 2]

to get

[2, 3, 4]

, the answer is 2.

nums[2...4] = [2, 4, 5]

is already sorted, so the answer is 0.

Example 2:

Input:

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

Output:

[4,4]

Explanation:

nums[0...3] = [5, 4, 3, 2]

. The whole subarray must be sorted, so the answer is 4.

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

. The whole subarray must be sorted, so the answer is 4.

Constraints:

1 <= nums.length <= 1000

1 <= k <= nums.length

1 <= nums[i] <= 10

Code Snippets

C++:

```
class Solution {
public:
vector<int> minSubarraySort(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int[] minSubarraySort(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number[]}
*/
```

```
var minSubarraySort = function(nums, k) {

};
```

TypeScript:

```
function minSubarraySort(nums: number[], k: number): number[] {

};
```

C#:

```
public class Solution {
public int[] MinSubarraySort(int[] nums, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* minSubarraySort(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go:

```
func minSubarraySort(nums []int, k int) []int {

}
```

Kotlin:

```
class Solution {
fun minSubarraySort(nums: IntArray, k: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func minSubarraySort(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn min_subarray_sort(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer[]}
def min_subarray_sort(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function minSubarraySort($nums, $k) {

}
}
```

Dart:

```
class Solution {
List<int> minSubarraySort(List<int> nums, int k) {

}
```

```
}
```

Scala:

```
object Solution {
def minSubarraySort(nums: Array[Int], k: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_subarray_sort(nums :: [integer], k :: integer) :: [integer]
def min_subarray_sort(nums, k) do

end
end
```

Erlang:

```
-spec min_subarray_sort(Nums :: [integer()], K :: integer()) -> [integer()].
min_subarray_sort(Nums, K) ->
.
```

Racket:

```
(define/contract (min-subarray-sort nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Subarray to Sort in Every Sliding Window
* 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:
vector<int> minSubarraySort(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Subarray to Sort in Every Sliding Window
* 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[] minSubarraySort(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Subarray to Sort in Every Sliding Window
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 minSubarraySort(self, nums: List[int], k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Smallest Subarray to Sort in Every Sliding Window
* 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
* @param {number} k
* @return {number[]}
*/
var minSubarraySort = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Subarray to Sort in Every Sliding Window
```

```
* 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 minSubarraySort(nums: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Smallest Subarray to Sort in Every Sliding Window
* 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[] MinSubarraySort(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Subarray to Sort in Every Sliding Window
* 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
*/
```

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* minSubarraySort(int* nums, int numsSize, int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: Smallest Subarray to Sort in Every Sliding Window
// 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 minSubarraySort(nums []int, k int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun minSubarraySort(nums: IntArray, k: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func minSubarraySort(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Smallest Subarray to Sort in Every Sliding Window
// 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 min_subarray_sort(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer[]}
def min_subarray_sort(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function minSubarraySort($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<int> minSubarraySort(List<int> nums, int k) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def minSubarraySort(nums: Array[Int], k: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_subarray_sort(nums :: [integer], k :: integer) :: [integer]
def min_subarray_sort(nums, k) do

end
end
```

Erlang Solution:

```
-spec min_subarray_sort(Nums :: [integer()], K :: integer()) -> [integer()].
min_subarray_sort(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (min-subarray-sort nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```