Problem 3349: Adjacent Increasing Subarrays Detection I

Problem Information

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

Problem Description

Given an array

nums

of

n

integers and an integer

k

, determine whether there exist

two

adjacent

subarrays

of length

k

such that both subarrays are

strictly

increasing

. Specifically, check if there are

two

subarrays starting at indices

a

and

b

(

a < b

), where:

Both subarrays

nums[a..a + k - 1]

and

nums[b..b + k - 1]

are

strictly increasing

.

The subarrays must be

adjacent

, meaning

b = a + k

.

Return

true

if it is

possible

to find

two

such subarrays, and

false

otherwise.

Example 1:

Input:

nums = [2,5,7,8,9,2,3,4,3,1], k = 3

Output:

true

Explanation:

The subarray starting at index

2

is

[7, 8, 9]

, which is strictly increasing.

The subarray starting at index

5

is

[2, 3, 4]

, which is also strictly increasing.

These two subarrays are adjacent, so the result is

true

.

Example 2:

Input:

nums = [1,2,3,4,4,4,4,5,6,7], k = 5

Output:

false

Constraints:

2 <= nums.length <= 100

1 < 2 * k <= nums.length

-1000 <= nums[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean hasIncreasingSubarrays(List<Integer> nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public bool HasIncreasingSubarrays(IList<int> nums, int k) {

}
}
```

C:

```
bool hasIncreasingSubarrays(int* nums, int numsSize, int k) {

}
```

Go:

```
func hasIncreasingSubarrays(nums []int, k int) bool {

}
```

Kotlin:

```
class Solution {
fun hasIncreasingSubarrays(nums: List<Int>, k: Int): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

```
object Solution {
def hasIncreasingSubarrays(nums: List[Int], k: Int): Boolean = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (has-increasing-subarrays nums k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Adjacent Increasing Subarrays Detection I
```

```
* Difficulty: Easy
* Tags: array
*
* 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 hasIncreasingSubarrays(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Adjacent Increasing Subarrays Detection I
* Difficulty: Easy
* Tags: array
*
* 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 hasIncreasingSubarrays(List<Integer> nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Adjacent Increasing Subarrays Detection I
Difficulty: Easy
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Adjacent Increasing Subarrays Detection I
* Difficulty: Easy
* Tags: array
*
* 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 {boolean}
*/
var hasIncreasingSubarrays = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Adjacent Increasing Subarrays Detection I
* Difficulty: Easy
* Tags: array
*
* 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 hasIncreasingSubarrays(nums: number[], k: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Adjacent Increasing Subarrays Detection I
* Difficulty: Easy
* Tags: array
*
* 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 HasIncreasingSubarrays(IList<int> nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Adjacent Increasing Subarrays Detection I
* Difficulty: Easy
* Tags: array
*
* 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 hasIncreasingSubarrays(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Adjacent Increasing Subarrays Detection I
// Difficulty: Easy
// Tags: array
//
// 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 hasIncreasingSubarrays(nums []int, k int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun hasIncreasingSubarrays(nums: List<Int>, k: Int): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Adjacent Increasing Subarrays Detection I
// Difficulty: Easy
// Tags: array
```

```
//
// 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 has_increasing_subarrays(nums: Vec<i32>, k: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def hasIncreasingSubarrays(nums: List[Int], k: Int): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (has-increasing-subarrays nums k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```