Problem 3637: Trionic Array I

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

.

An array is

trionic

if there exist indices

0 < p < q < n – 1

such that:

nums[0...p]

is

strictly

increasing,

nums[p...q]

is

strictly

decreasing,

nums[q...n – 1]

is

strictly

increasing.

Return

true

if

nums

is trionic, otherwise return

false

.

Example 1:

Input:

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

Output:

true

Explanation:

Pick

p = 2

,

q = 4

:

nums[0...2] = [1, 3, 5]

is strictly increasing (

1 < 3 < 5

).

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

is strictly decreasing (

5 > 4 > 2

).

nums[4...5] = [2, 6]

is strictly increasing (

2 < 6

).

Example 2:

Input:

nums = [2,1,3]

Output:

false

Explanation:

There is no way to pick

p

and

q

to form the required three segments.

Constraints:

3 <= n <= 100

-1000 <= nums[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean isTrionic(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function isTrionic(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool IsTrionic(int[] nums) {
```

```
}
}
```

C:

```
bool isTrionic(int* nums, int numsSize) {

}
```

Go:

```
func isTrionic(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun isTrionic(nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool isTrionic(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (is-trionic nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Trionic Array 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 isTrionic(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Trionic Array 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 isTrionic(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Trionic Array 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 isTrionic(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Trionic Array 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
* @return {boolean}
*/
var isTrionic = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Trionic Array 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 isTrionic(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Trionic Array 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 IsTrionic(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Trionic Array 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 isTrionic(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Trionic Array 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 isTrionic(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isTrionic(nums: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Trionic Array 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 is_trionic(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $nums
* @return Boolean
*/
function isTrionic($nums) {

}
}
```

Dart Solution:

```
class Solution {
bool isTrionic(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (is-trionic nums)
(-> (listof exact-integer?) boolean?)
)
```