Problem 665: Non-decreasing Array

Problem Information

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

Problem Description

Given an array

nums

with

n

integers, your task is to check if it could become non-decreasing by modifying

at most one element

.

We define an array is non-decreasing if

nums[i] <= nums[i + 1]

holds for every

i

(

0-based

) such that (

0 <= i <= n - 2

).

Example 1:

Input:

nums = [4,2,3]

Output:

true

Explanation:

You could modify the first 4 to 1 to get a non-decreasing array.

Example 2:

Input:

nums = [4,2,1]

Output:

false

Explanation:

You cannot get a non-decreasing array by modifying at most one element.

Constraints:

n == nums.length

1 <= n <= 10

4

-10

5

<= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (check-possibility nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Non-decreasing Array
* Difficulty: Medium
* 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 checkPossibility(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Non-decreasing Array
* Difficulty: Medium
* 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 checkPossibility(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Non-decreasing Array
Difficulty: Medium
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 checkPossibility(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Non-decreasing Array
* Difficulty: Medium
* 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 checkPossibility = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Non-decreasing Array
* Difficulty: Medium
* 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 checkPossibility(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Non-decreasing Array
* Difficulty: Medium
* 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 CheckPossibility(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Non-decreasing Array
* Difficulty: Medium
* 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 checkPossibility(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Non-decreasing Array
// Difficulty: Medium
// 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 checkPossibility(nums []int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Non-decreasing Array
// Difficulty: Medium
// 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 check_possibility(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (check-possibility nums)
(-> (listof exact-integer?) boolean?)
)
```