Problem 1909: Remove One Element to Make the Array Strictly Increasing

Problem Information

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

Problem Description

Given a

0-indexed

integer array

nums

, return

true

if it can be made

strictly increasing

after removing

exactly one

element, or

false

otherwise. If the array is already strictly increasing, return

true

.

The array

nums

is

strictly increasing

if

nums[i - 1] < nums[i]

for each index

(1 <= i < nums.length).

Example 1:

Input:

nums = [1,2,

10

,5,7]

Output:

true

Explanation:

By removing 10 at index 2 from nums, it becomes [1,2,5,7]. [1,2,5,7] is strictly increasing, so return true.

Example 2:

Input:

nums = [2,3,1,2]

Output:

false

Explanation:

[3,1,2] is the result of removing the element at index 0. [2,1,2] is the result of removing the element at index 1. [2,3,2] is the result of removing the element at index 2. [2,3,1] is the result of removing the element at index 3. No resulting array is strictly increasing, so return false.

Example 3:

Input:

nums = [1,1,1]

Output:

false

Explanation:

The result of removing any element is [1,1]. [1,1] is not strictly increasing, so return false.

Constraints:

2 <= nums.length <= 1000

1 <= nums[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (can-be-increasing nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove One Element to Make the Array Strictly Increasing
* 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 canBeIncreasing(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Remove One Element to Make the Array Strictly Increasing
* 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 canBeIncreasing(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove One Element to Make the Array Strictly Increasing
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 canBeIncreasing(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Remove One Element to Make the Array Strictly Increasing
* 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 canBeIncreasing = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove One Element to Make the Array Strictly Increasing
* 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 canBeIncreasing(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Remove One Element to Make the Array Strictly Increasing
* 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 CanBeIncreasing(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Remove One Element to Make the Array Strictly Increasing
* 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 canBeIncreasing(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Remove One Element to Make the Array Strictly Increasing
// 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 canBeIncreasing(nums []int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Remove One Element to Make the Array Strictly Increasing
// 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 can_be_increasing(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (can-be-increasing nums)
(-> (listof exact-integer?) boolean?)
)
```