Problem 1752: Check if Array Is Sorted and Rotated

Problem Information

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

Problem Description

Given an array

nums

, return

true

if the array was originally sorted in non-decreasing order, then rotated

some

number of positions (including zero)

. Otherwise, return

false

.

There may be

duplicates

in the original array.

Note:

An array

A

rotated by

x

positions results in an array

B

of the same length such that

B[i] == A[(i+x) % A.length]

for every valid index

i

.

Example 1:

Input:

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

Output:

true

Explanation:

[1,2,3,4,5] is the original sorted array. You can rotate the array by x = 2 positions to begin on the element of value 3: [3,4,5,1,2].

Example 2:

Input:

nums = [2,1,3,4]

Output:

false

Explanation:

There is no sorted array once rotated that can make nums.

Example 3:

Input:

nums = [1,2,3]

Output:

true

Explanation:

[1,2,3] is the original sorted array. You can rotate the array by x = 0 positions (i.e. no rotation) to make nums.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Check if Array Is Sorted and Rotated
* Difficulty: Easy
* Tags: array, sort
*
* 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 check(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Check if Array Is Sorted and Rotated
* Difficulty: Easy
* Tags: array, sort
*
* 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 check(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if Array Is Sorted and Rotated
Difficulty: Easy
Tags: array, sort

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Check if Array Is Sorted and Rotated
* Difficulty: Easy
* Tags: array, sort
*
* 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 check = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if Array Is Sorted and Rotated
* Difficulty: Easy
* Tags: array, sort
*
* 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 check(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if Array Is Sorted and Rotated
* Difficulty: Easy
* Tags: array, sort
*
* 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 Check(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Check if Array Is Sorted and Rotated
* Difficulty: Easy
* Tags: array, sort
*
* 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 check(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Check if Array Is Sorted and Rotated
// Difficulty: Easy
// Tags: array, sort
//
// 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 check(nums []int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Check if Array Is Sorted and Rotated
// Difficulty: Easy
// Tags: array, sort
//
// 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(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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