Problem 2772: Apply Operations to Make All Array Elements Equal to Zero

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

and a positive integer

k

.

You can apply the following operation on the array

any

number of times:

Choose

any

subarray of size

k

from the array and

decrease

all its elements by

1

.

Return

true

if you can make all the array elements equal to

0

, or

false

otherwise

.

A

subarray

is a contiguous non-empty part of an array.

Example 1:

Input:

nums = [2,2,3,1,1,0], k = 3

Output:

true

Explanation:

We can do the following operations: - Choose the subarray [2,2,3]. The resulting array will be nums = [

1

,

1

,

2

,1,1,0]. - Choose the subarray [2,1,1]. The resulting array will be nums = [1,1,

1

,

0

,

0

,0]. - Choose the subarray [1,1,1]. The resulting array will be nums = [

0

,

0

,

0

,0,0,0].

Example 2:

Input:

nums = [1,3,1,1], k = 2

Output:

false

Explanation:

It is not possible to make all the array elements equal to 0.

Constraints:

1 <= k <= nums.length <= 10

5

0 <= nums[i] <= 10

6

Code Snippets

C++:

```
class Solution {
public:
```

```
bool checkArray(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public boolean checkArray(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public bool CheckArray(int[] nums, int k) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun checkArray(nums: IntArray, k: Int): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Apply Operations to Make All Array Elements Equal to Zero
* 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 checkArray(vector<int>& nums, int k) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Apply Operations to Make All Array Elements Equal to Zero
* 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 checkArray(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Apply Operations to Make All Array Elements Equal to Zero
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 checkArray(self, nums: List[int], k: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Apply Operations to Make All Array Elements Equal to Zero
* 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
* @param {number} k
* @return {boolean}
*/
var checkArray = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Apply Operations to Make All Array Elements Equal to Zero
* 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 checkArray(nums: number[], k: number): boolean {
```

```
};
```

C# Solution:

```
/*
* Problem: Apply Operations to Make All Array Elements Equal to Zero
* 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 CheckArray(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Apply Operations to Make All Array Elements Equal to Zero
* 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 checkArray(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Apply Operations to Make All Array Elements Equal to Zero
// 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 checkArray(nums []int, k int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun checkArray(nums: IntArray, k: Int): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Apply Operations to Make All Array Elements Equal to Zero
// 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_array(nums: Vec<i32>, k: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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