Problem 659: Split Array into Consecutive Subsequences

Problem Information

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

Problem Description

You are given an integer array

nums

that is

sorted in non-decreasing order

.

Determine if it is possible to split

nums

into

one or more subsequences

such that

both

of the following conditions are true:

Each subsequence is a

consecutive increasing sequence

(i.e. each integer is

exactly one

more than the previous integer).

All subsequences have a length of

3

or more

.

Return

true

if you can split

nums

according to the above conditions, or

false

otherwise

.

A

subsequence

of an array is a new array that is formed from the original array by deleting some (can be none) of the elements without disturbing the relative positions of the remaining elements. (i.e.,

[1,3,5]

is a subsequence of

[

1

,2,

3

,4,

5

]

while

[1,3,2]

is not).

Example 1:

Input:

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

Output:

true

Explanation:

nums can be split into the following subsequences: [

1

,

2

,

3

,3,4,5] --> 1, 2, 3 [1,2,3,

3

,

4

,

5

] --> 3, 4, 5

Example 2:

Input:

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

Output:

true

Explanation:

nums can be split into the following subsequences: [

1

,

2

,

3

,3,

4

,4,

5

,5] --> 1, 2, 3, 4, 5 [1,2,3,

3

,4,

4

,5,

5

] --> 3, 4, 5

Example 3:

Input:

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

Output:

false

Explanation:

It is impossible to split nums into consecutive increasing subsequences of length 3 or more.

Constraints:

1 <= nums.length <= 10

4

-1000 <= nums[i] <= 1000

nums

is sorted in

non-decreasing

order.

Code Snippets

C++:

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

}
};
```

Java:

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

}


}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Split Array into Consecutive Subsequences
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Split Array into Consecutive Subsequences
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean isPossible(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Split Array into Consecutive Subsequences
Difficulty: Medium
Tags: array, greedy, hash, sort, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Split Array into Consecutive Subsequences
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Split Array into Consecutive Subsequences
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function isPossible(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Split Array into Consecutive Subsequences
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool IsPossible(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Split Array into Consecutive Subsequences
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool isPossible(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Split Array into Consecutive Subsequences
// Difficulty: Medium
// Tags: array, greedy, hash, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func isPossible(nums []int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Split Array into Consecutive Subsequences
// Difficulty: Medium
// Tags: array, greedy, hash, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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