Problem 2811: Check if it is Possible to Split Array

Problem Information

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

Problem Description

You are given an array

nums

of length

n

and an integer

m

. You need to determine if it is possible to split the array into

n

arrays of size 1 by performing a series of steps.

An array is called

good

if:

The length of the array is

one

, or

The sum of the elements of the array is

greater than or equal

to

m

.

In each step, you can select an existing array (which may be the result of previous steps) with a length of

at least two

and split it into

two

arrays, if both resulting arrays are good.

Return true if you can split the given array into

n

arrays, otherwise return false.

Example 1:

Input:

nums = [2, 2, 1], m = 4

Output:

true

Explanation:

Split

[2, 2, 1]

to

[2, 2]

and

[1]

. The array

[1]

has a length of one, and the array

[2, 2]

has the sum of its elements equal to

4 >= m

, so both are good arrays.

Split

[2, 2]

to

[2]

and

[2]

. both arrays have the length of one, so both are good arrays.

Example 2:

Input:

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

Output:

false

Explanation:

The first move has to be either of the following:

Split

[2, 1, 3]

to

[2, 1]

and

[3]

. The array

[2, 1]

has neither length of one nor sum of elements greater than or equal to

m

.

Split

[2, 1, 3]

to

[2]

and

[1, 3]

. The array

[1, 3]

has neither length of one nor sum of elements greater than or equal to

m

.

So as both moves are invalid (they do not divide the array into two good arrays), we are unable to split

nums

into

n

arrays of size 1.

Example 3:

Input:

nums = [2, 3, 3, 2, 3], m = 6

Output:

true

Explanation:

Split

[2, 3, 3, 2, 3]

to

[2]

and

[3, 3, 2, 3]

.

Split

[3, 3, 2, 3]

to

[3, 3, 2]

and

[3]

.

Split

[3, 3, 2]

to

[3, 3]

and

[2]

.

Split

[3, 3]

to

[3]

and

[3]

.

Constraints:

1 <= n == nums.length <= 100

1 <= nums[i] <= 100

1 <= m <= 200

Code Snippets

C++:

```
class Solution {
public:
bool canSplitArray(vector<int>& nums, int m) {

}
};
```

Java:

```
class Solution {
public boolean canSplitArray(List<Integer> nums, int m) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function canSplitArray(nums: number[], m: number): boolean {

};
```

C#:

```
public class Solution {
public bool CanSplitArray(IList<int> nums, int m) {

}
}
```

C:

```
bool canSplitArray(int* nums, int numsSize, int m) {

}
```

Go:

```
func canSplitArray(nums []int, m int) bool {

}
```

Kotlin:

```
class Solution {
fun canSplitArray(nums: List<Int>, m: Int): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool canSplitArray(List<int> nums, int m) {

}
}
```

Scala:

```
object Solution {
def canSplitArray(nums: List[Int], m: Int): Boolean = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec can_split_array(Nums :: [integer()], M :: integer()) -> boolean().
can_split_array(Nums, M) ->
.
```

Racket:

```
(define/contract (can-split-array nums m)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if it is Possible to Split Array
* Difficulty: Medium
* Tags: array, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
bool canSplitArray(vector<int>& nums, int m) {
}
};
```

Java Solution:

```
/**
* Problem: Check if it is Possible to Split Array
* Difficulty: Medium
* Tags: array, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public boolean canSplitArray(List<Integer> nums, int m) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if it is Possible to Split Array
Difficulty: Medium
Tags: array, dp, greedy

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check if it is Possible to Split Array
* Difficulty: Medium
* Tags: array, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} nums
* @param {number} m
* @return {boolean}
*/
var canSplitArray = function(nums, m) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if it is Possible to Split Array
* Difficulty: Medium
* Tags: array, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function canSplitArray(nums: number[], m: number): boolean {
};
```

C# Solution:

```
/*
* Problem: Check if it is Possible to Split Array
* Difficulty: Medium
* Tags: array, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public bool CanSplitArray(IList<int> nums, int m) {

}
}
```

C Solution:

```
/*
* Problem: Check if it is Possible to Split Array
* Difficulty: Medium
* Tags: array, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

bool canSplitArray(int* nums, int numsSize, int m) {

}
```

Go Solution:

```
// Problem: Check if it is Possible to Split Array
// Difficulty: Medium
// Tags: array, dp, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func canSplitArray(nums []int, m int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canSplitArray(nums: List<Int>, m: Int): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Check if it is Possible to Split Array
// Difficulty: Medium
// Tags: array, dp, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn can_split_array(nums: Vec<i32>, m: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool canSplitArray(List<int> nums, int m) {

}
}
```

Scala Solution:

```
object Solution {
def canSplitArray(nums: List[Int], m: Int): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec can_split_array(Nums :: [integer()], M :: integer()) -> boolean().
can_split_array(Nums, M) ->
.
```

Racket Solution:

```
(define/contract (can-split-array nums m)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```