Problem 209: Minimum Size Subarray Sum

Problem Information

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

Problem Description

Given an array of positive integers

nums

and a positive integer

target

, return

the

minimal length

of a

subarray

whose sum is greater than or equal to

target

. If there is no such subarray, return

0

instead.

Example 1:

Input:

target = 7, nums = [2,3,1,2,4,3]

Output:

2

Explanation:

The subarray [4,3] has the minimal length under the problem constraint.

Example 2:

Input:

target = 4, nums = [1,4,4]

Output:

1

Example 3:

Input:

target = 11, nums = [1,1,1,1,1,1,1,1]

Output:

0

Constraints:

1 <= target <= 10

9

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

4

Follow up:

If you have figured out the

O(n)

solution, try coding another solution of which the time complexity is

O(n log(n))

.

Code Snippets

C++:

```
class Solution {
public:
int minSubArrayLen(int target, vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int minSubArrayLen(int target, int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} target
* @param {number[]} nums
* @return {number}
*/
var minSubArrayLen = function(target, nums) {

};
```

TypeScript:

```
function minSubArrayLen(target: number, nums: number[]): number {

};
```

C#:

```
public class Solution {
public int MinSubArrayLen(int target, int[] nums) {

}
}
```

C:

```
int minSubArrayLen(int target, int* nums, int numsSize) {

}
```

Go:

```
func minSubArrayLen(target int, nums []int) int {

}
```

Kotlin:

```
class Solution {
fun minSubArrayLen(target: Int, nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minSubArrayLen(_ target: Int, _ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_sub_array_len(target: i32, nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} target
# @param {Integer[]} nums
# @return {Integer}
def min_sub_array_len(target, nums)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $target
* @param Integer[] $nums
* @return Integer
*/
function minSubArrayLen($target, $nums) {

}
}
```

Dart:

```
class Solution {
int minSubArrayLen(int target, List<int> nums) {

}
}
```

Scala:

```
object Solution {
def minSubArrayLen(target: Int, nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_sub_array_len(target :: integer, nums :: [integer]) :: integer
def min_sub_array_len(target, nums) do

end
end
```

Erlang:

```
-spec min_sub_array_len(Target :: integer(), Nums :: [integer()]) ->
integer().
min_sub_array_len(Target, Nums) ->
.
```

Racket:

```
(define/contract (min-sub-array-len target nums)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Size Subarray Sum
* Difficulty: Medium
* Tags: array, search
*
* 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:
int minSubArrayLen(int target, vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Size Subarray Sum
* Difficulty: Medium
* Tags: array, search
*
* 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 int minSubArrayLen(int target, int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Size Subarray Sum
Difficulty: Medium
Tags: array, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Size Subarray Sum
* Difficulty: Medium
* Tags: array, search
*
* 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} target
* @param {number[]} nums
* @return {number}
*/
var minSubArrayLen = function(target, nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Size Subarray Sum
* Difficulty: Medium
* Tags: array, search
*
* 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 minSubArrayLen(target: number, nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Size Subarray Sum
* Difficulty: Medium
* Tags: array, search
*
* 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 int MinSubArrayLen(int target, int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Size Subarray Sum
* Difficulty: Medium
* Tags: array, search
*
* 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
*/

int minSubArrayLen(int target, int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Size Subarray Sum
// Difficulty: Medium
// Tags: array, search
//
// 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 minSubArrayLen(target int, nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minSubArrayLen(target: Int, nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minSubArrayLen(_ target: Int, _ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Size Subarray Sum
// Difficulty: Medium
// Tags: array, search
//
// 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 min_sub_array_len(target: i32, nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} target
# @param {Integer[]} nums
# @return {Integer}
def min_sub_array_len(target, nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $target
* @param Integer[] $nums
* @return Integer
*/
function minSubArrayLen($target, $nums) {

}
}
```

Dart Solution:

```
class Solution {
int minSubArrayLen(int target, List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def minSubArrayLen(target: Int, nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_sub_array_len(target :: integer, nums :: [integer]) :: integer
def min_sub_array_len(target, nums) do

end
end
```

Erlang Solution:

```
-spec min_sub_array_len(Target :: integer(), Nums :: [integer()]) ->
integer().
min_sub_array_len(Target, Nums) ->
```

```
.
```

Racket Solution:

```
(define/contract (min-sub-array-len target nums)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```