Problem 3364: Minimum Positive Sum Subarray

Problem Information

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

Problem Description

You are given an integer array

nums

and

two

integers

l

and

r

. Your task is to find the

minimum

sum of a

subarray

whose size is between

l

and

r

(inclusive) and whose sum is greater than 0.

Return the

minimum

sum of such a subarray. If no such subarray exists, return -1.

A

subarray

is a contiguous

non-empty

sequence of elements within an array.

Example 1:

Input:

nums = [3, -2, 1, 4], l = 2, r = 3

Output:

1

Explanation:

The subarrays of length between

l = 2

and

r = 3

where the sum is greater than 0 are:

[3, -2]

with a sum of 1

[1, 4]

with a sum of 5

[3, -2, 1]

with a sum of 2

[-2, 1, 4]

with a sum of 3

Out of these, the subarray

[3, -2]

has a sum of 1, which is the smallest positive sum. Hence, the answer is 1.

Example 2:

Input:

nums = [-2, 2, -3, 1], l = 2, r = 3

Output:

-1

Explanation:

There is no subarray of length between

l

and

r

that has a sum greater than 0. So, the answer is -1.

Example 3:

Input:

nums = [1, 2, 3, 4], l = 2, r = 4

Output:

3

Explanation:

The subarray

[1, 2]

has a length of 2 and the minimum sum greater than 0. So, the answer is 3.

Constraints:

1 <= nums.length <= 100

1 <= l <= r <= nums.length

-1000 <= nums[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int minimumSumSubarray(vector<int>& nums, int l, int r) {

}
};
```

Java:

```
class Solution {
public int minimumSumSubarray(List<Integer> nums, int l, int r) {

}
}
```

Python3:

```
class Solution:
def minimumSumSubarray(self, nums: List[int], l: int, r: int) -> int:
```

Python:

```
class Solution(object):
def minimumSumSubarray(self, nums, l, r):
"""
:type nums: List[int]
:type l: int
:type r: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} l
* @param {number} r
* @return {number}
*/
```

```
var minimumSumSubarray = function(nums, l, r) {

};
```

TypeScript:

```
function minimumSumSubarray(nums: number[], l: number, r: number): number {

};
```

C#:

```
public class Solution {
public int MinimumSumSubarray(IList<int> nums, int l, int r) {

}
}
```

C:

```
int minimumSumSubarray(int* nums, int numsSize, int l, int r) {

}
```

Go:

```
func minimumSumSubarray(nums []int, l int, r int) int {

}
```

Kotlin:

```
class Solution {
fun minimumSumSubarray(nums: List<Int>, l: Int, r: Int): Int {

}
}
```

Swift:

```
class Solution {
func minimumSumSubarray(_ nums: [Int], _ l: Int, _ r: Int) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn minimum_sum_subarray(nums: Vec<i32>, l: i32, r: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} l
# @param {Integer} r
# @return {Integer}
def minimum_sum_subarray(nums, l, r)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $l
* @param Integer $r
* @return Integer
*/
function minimumSumSubarray($nums, $l, $r) {

}
}
```

Dart:

```
class Solution {
int minimumSumSubarray(List<int> nums, int l, int r) {
```

```
}
}
```

Scala:

```
object Solution {
def minimumSumSubarray(nums: List[Int], l: Int, r: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_sum_subarray(nums :: [integer], l :: integer, r :: integer) ::
integer
def minimum_sum_subarray(nums, l, r) do

end
end
```

Erlang:

```
-spec minimum_sum_subarray(Nums :: [integer()], L :: integer(), R ::
integer()) -> integer().
minimum_sum_subarray(Nums, L, R) ->
.
```

Racket:

```
(define/contract (minimum-sum-subarray nums l r)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Positive Sum Subarray
```

```
* Difficulty: Easy
* 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:
int minimumSumSubarray(vector<int>& nums, int l, int r) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Positive Sum Subarray
* Difficulty: Easy
* 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 int minimumSumSubarray(List<Integer> nums, int l, int r) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Positive Sum Subarray
Difficulty: Easy
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 minimumSumSubarray(self, nums: List[int], l: int, r: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumSumSubarray(self, nums, l, r):
"""
:type nums: List[int]
:type l: int
:type r: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Positive Sum Subarray
* Difficulty: Easy
* 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} l
* @param {number} r
* @return {number}
*/
var minimumSumSubarray = function(nums, l, r) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Positive Sum Subarray
* Difficulty: Easy
* 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 minimumSumSubarray(nums: number[], l: number, r: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Positive Sum Subarray
* Difficulty: Easy
* 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 int MinimumSumSubarray(IList<int> nums, int l, int r) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Positive Sum Subarray
* Difficulty: Easy
* 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
*/

int minimumSumSubarray(int* nums, int numsSize, int l, int r) {

}
```

Go Solution:

```
// Problem: Minimum Positive Sum Subarray
// Difficulty: Easy
// 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 minimumSumSubarray(nums []int, l int, r int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumSumSubarray(nums: List<Int>, l: Int, r: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumSumSubarray(_ nums: [Int], _ l: Int, _ r: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Positive Sum Subarray
// Difficulty: Easy
// 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 minimum_sum_subarray(nums: Vec<i32>, l: i32, r: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} l
# @param {Integer} r
# @return {Integer}
def minimum_sum_subarray(nums, l, r)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $l
* @param Integer $r
* @return Integer
*/
function minimumSumSubarray($nums, $l, $r) {

}
}
```

Dart Solution:

```
class Solution {
int minimumSumSubarray(List<int> nums, int l, int r) {

}
}
```

Scala Solution:

```
object Solution {
def minimumSumSubarray(nums: List[Int], l: Int, r: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_sum_subarray(nums :: [integer], l :: integer, r :: integer) ::
integer
def minimum_sum_subarray(nums, l, r) do

end
end
```

Erlang Solution:

```
-spec minimum_sum_subarray(Nums :: [integer()], L :: integer(), R ::
integer()) -> integer().
minimum_sum_subarray(Nums, L, R) ->
.
```

Racket Solution:

```
(define/contract (minimum-sum-subarray nums l r)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```