Problem 3427: Sum of Variable Length Subarrays

Problem Information

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

Problem Description

You are given an integer array

nums

of size

n

. For

each

index

i

where

0 <= i < n

, define a

subarray

nums[start ... i]

where

start = max(0, i - nums[i])

.

Return the total sum of all elements from the subarray defined for each index in the array.

Example 1:

Input:

nums = [2,3,1]

Output:

11

Explanation:

i

Subarray

Sum

0

nums[0] = [2]

2

1

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

5

2

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

4

Total Sum

11

The total sum is 11. Hence, 11 is the output.

Example 2:

Input:

nums = [3,1,1,2]

Output:

13

Explanation:

i

Subarray

Sum

0

nums[0] = [3]

3

1

nums[0 ... 1] = [3, 1]

4

2

nums[1 ... 2] = [1, 1]

2

3

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

4

Total Sum

13

The total sum is 13. Hence, 13 is the output.

Constraints:

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

1 <= nums[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int subarraySum(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function subarraySum(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int SubarraySum(int[] nums) {

}
}
```

C:

```
int subarraySum(int* nums, int numsSize) {

}
```

Go:

```
func subarraySum(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun subarraySum(nums: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int subarraySum(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Sum of Variable Length Subarrays
* 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 subarraySum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Variable Length Subarrays
* 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 subarraySum(int[] nums) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Sum of Variable Length Subarrays
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 subarraySum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Variable Length Subarrays
* 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
* @return {number}
*/
var subarraySum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Variable Length Subarrays
* 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 subarraySum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Variable Length Subarrays
* 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 SubarraySum(int[] nums) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Sum of Variable Length Subarrays
* 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 subarraySum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of Variable Length Subarrays
// 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 subarraySum(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun subarraySum(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Variable Length Subarrays
// 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 subarray_sum(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int subarraySum(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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