Problem 829: Consecutive Numbers Sum

Problem Information

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

Problem Description

Given an integer

n

, return

the number of ways you can write

n

as the sum of consecutive positive integers.

Example 1:

Input:

n = 5

Output:

2

Explanation:

5 = 2 + 3

Example 2:

Input:

n = 9

Output:

3

Explanation:

9 = 4 + 5 = 2 + 3 + 4

Example 3:

Input:

n = 15

Output:

4

Explanation:

15 = 8 + 7 = 4 + 5 + 6 = 1 + 2 + 3 + 4 + 5

Constraints:

1 <= n <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int consecutiveNumbersSum(int n) {

}
};
```

Java:

```
class Solution {
public int consecutiveNumbersSum(int n) {

}
}
```

Python3:

```
class Solution:
def consecutiveNumbersSum(self, n: int) -> int:
```

Python:

```
class Solution(object):
def consecutiveNumbersSum(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var consecutiveNumbersSum = function(n) {

};
```

TypeScript:

```
function consecutiveNumbersSum(n: number): number {
};
```

C#:

```
public class Solution {
public int ConsecutiveNumbersSum(int n) {

}
}
```

C:

```
int consecutiveNumbersSum(int n) {

}
```

Go:

```
func consecutiveNumbersSum(n int) int {

}
```

Kotlin:

```
class Solution {
fun consecutiveNumbersSum(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func consecutiveNumbersSum(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn consecutive_numbers_sum(n: i32) -> i32 {
}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def consecutive_numbers_sum(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function consecutiveNumbersSum($n) {

}
}
```

Dart:

```
class Solution {
int consecutiveNumbersSum(int n) {

}
}
```

Scala:

```
object Solution {
def consecutiveNumbersSum(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec consecutive_numbers_sum(n :: integer) :: integer
def consecutive_numbers_sum(n) do

end
end
```

Erlang:

```
-spec consecutive_numbers_sum(N :: integer()) -> integer().
consecutive_numbers_sum(N) ->
.
```

Racket:

```
(define/contract (consecutive-numbers-sum n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Consecutive Numbers Sum
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int consecutiveNumbersSum(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Consecutive Numbers Sum
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int consecutiveNumbersSum(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Consecutive Numbers Sum
Difficulty: Hard
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def consecutiveNumbersSum(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def consecutiveNumbersSum(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Consecutive Numbers Sum
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @return {number}
*/
var consecutiveNumbersSum = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Consecutive Numbers Sum
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function consecutiveNumbersSum(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Consecutive Numbers Sum
* Difficulty: Hard
* Tags: math
*
```

```
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int ConsecutiveNumbersSum(int n) {

}
}
```

C Solution:

```
/*
* Problem: Consecutive Numbers Sum
* Difficulty: Hard
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int consecutiveNumbersSum(int n) {

}
```

Go Solution:

```
// Problem: Consecutive Numbers Sum
// Difficulty: Hard
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func consecutiveNumbersSum(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun consecutiveNumbersSum(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func consecutiveNumbersSum(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Consecutive Numbers Sum
// Difficulty: Hard
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn consecutive_numbers_sum(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def consecutive_numbers_sum(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function consecutiveNumbersSum($n) {

}
}
```

Dart Solution:

```
class Solution {
int consecutiveNumbersSum(int n) {

}
}
```

Scala Solution:

```
object Solution {
def consecutiveNumbersSum(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec consecutive_numbers_sum(n :: integer) :: integer
def consecutive_numbers_sum(n) do

end
end
```

Erlang Solution:

```
-spec consecutive_numbers_sum(N :: integer()) -> integer().
consecutive_numbers_sum(N) ->
.
```

Racket Solution:

```
(define/contract (consecutive-numbers-sum n)
(-> exact-integer? exact-integer?)
)
```