# Problem 1925: Count Square Sum Triples

## Problem Information

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

## Problem Description

A

square triple

(a,b,c)

is a triple where

a

,

b

, and

c

are

integers

and

a

2

+ b

2

= c

2

.

Given an integer

n

, return

the number of

square triples

such that

1 <= a, b, c <= n

.

Example 1:

Input:

n = 5

Output:

2

Explanation

: The square triples are (3,4,5) and (4,3,5).

Example 2:

Input:

n = 10

Output:

4

Explanation

: The square triples are (3,4,5), (4,3,5), (6,8,10), and (8,6,10).

Constraints:

1 <= n <= 250

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function countTriples(n: number): number {

};
```

**C#:**

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

}
}
```

**C:**

```
int countTriples(int n) {

}
```

**Go:**

```
func countTriples(n int) int {

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

```
(define/contract (count-triples n)
(-> exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Count Square Sum Triples
* Difficulty: Easy
* 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 countTriples(int n) {

}
};
```

### Java Solution:

```
/**
* Problem: Count Square Sum Triples
* Difficulty: Easy
* 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 countTriples(int n) {

}
}
```

**Python3 Solution:**

```
"""
Problem: Count Square Sum Triples
Difficulty: Easy
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 countTriples(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

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

**JavaScript Solution:**

```
/**
* Problem: Count Square Sum Triples
* Difficulty: Easy
* 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 countTriples = function(n) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Count Square Sum Triples
* Difficulty: Easy
* 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 countTriples(n: number): number {

};
```

**C# Solution:**

```
/*
* Problem: Count Square Sum Triples
* Difficulty: Easy
* 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 CountTriples(int n) {

}
}
```

**C Solution:**

```
/*
* Problem: Count Square Sum Triples
* Difficulty: Easy
* 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 countTriples(int n) {

}
```

**Go Solution:**

```
// Problem: Count Square Sum Triples
// Difficulty: Easy
// 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 countTriples(n int) int {

}
```

**Kotlin Solution:**

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

}
}
```

**Swift Solution:**

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

**Rust Solution:**

```
// Problem: Count Square Sum Triples
// Difficulty: Easy
// 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 count_triples(n: i32) -> i32 {

}
}
```

**Ruby Solution:**

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

end
```

**PHP Solution:**

```
class Solution {

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

}
}
```

**Dart Solution:**

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

}
}
```

**Scala Solution:**

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

}
}
```

**Elixir Solution:**

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

end
end
```

**Erlang Solution:**

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

**Racket Solution:**

```
(define/contract (count-triples n)
(-> exact-integer? exact-integer?)
)
```