Problem 2582: Pass the Pillow

Problem Information

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

Problem Description

There are

n

people standing in a line labeled from

1

to

n

. The first person in the line is holding a pillow initially. Every second, the person holding the pillow passes it to the next person standing in the line. Once the pillow reaches the end of the line, the direction changes, and people continue passing the pillow in the opposite direction.

For example, once the pillow reaches the

n

th

person they pass it to the

n - 1

th

person, then to the

n - 2

th

person and so on.

Given the two positive integers

n

and

time

, return

the index of the person holding the pillow after

time

seconds

.

Example 1:

Input:

n = 4, time = 5

Output:

2

Explanation:

People pass the pillow in the following way: 1 -> 2 -> 3 -> 4 -> 3 -> 2. After five seconds, the 2

nd

person is holding the pillow.

Example 2:

Input:

n = 3, time = 2

Output:

3

Explanation:

People pass the pillow in the following way: 1 -> 2 -> 3. After two seconds, the 3

r

d

person is holding the pillow.

Constraints:

2 <= n <= 1000

1 <= time <= 1000

Note:

This question is the same as

3178: Find the Child Who Has the Ball After K Seconds.

Code Snippets

C++:

```
class Solution {
public:
int passThePillow(int n, int time) {

}
};
```

Java:

```
class Solution {
public int passThePillow(int n, int time) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} n
* @param {number} time
* @return {number}
*/
var passThePillow = function(n, time) {
};
```

TypeScript:

```
function passThePillow(n: number, time: number): number {

};
```

C#:

```
public class Solution {
public int PassThePillow(int n, int time) {

}
}
```

C:

```
int passThePillow(int n, int time) {

}
```

Go:

```
func passThePillow(n int, time int) int {

}
```

Kotlin:

```
class Solution {
fun passThePillow(n: Int, time: Int): Int {

}
}
```

Swift:

```
class Solution {
func passThePillow(_ n: Int, _ time: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn pass_the_pillow(n: i32, time: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} time
# @return {Integer}
def pass_the_pillow(n, time)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $time
* @return Integer
*/
function passThePillow($n, $time) {

}
}
```

Dart:

```
class Solution {
int passThePillow(int n, int time) {

}
}
```

Scala:

```
object Solution {
def passThePillow(n: Int, time: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec pass_the_pillow(n :: integer, time :: integer) :: integer
def pass_the_pillow(n, time) do

end
end
```

Erlang:

```
-spec pass_the_pillow(N :: integer(), Time :: integer()) -> integer().
pass_the_pillow(N, Time) ->
.
```

Racket:

```
(define/contract (pass-the-pillow n time)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Pass the Pillow
* 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 passThePillow(int n, int time) {

}
};
```

Java Solution:

```
/**
* Problem: Pass the Pillow
* 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 passThePillow(int n, int time) {

}
}
```

Python3 Solution:

```
"""
Problem: Pass the Pillow
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 passThePillow(self, n: int, time: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Pass the Pillow
* 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
* @param {number} time
* @return {number}
*/
var passThePillow = function(n, time) {

};
```

TypeScript Solution:

```
/**
* Problem: Pass the Pillow
* 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 passThePillow(n: number, time: number): number {

};
```

C# Solution:

```
/*
* Problem: Pass the Pillow
* 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 PassThePillow(int n, int time) {

}
}
```

C Solution:

```
/*
* Problem: Pass the Pillow
* 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 passThePillow(int n, int time) {

}
```

Go Solution:

```
// Problem: Pass the Pillow
// 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 passThePillow(n int, time int) int {

}
```

Kotlin Solution:

```
class Solution {
fun passThePillow(n: Int, time: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func passThePillow(_ n: Int, _ time: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Pass the Pillow
// 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 pass_the_pillow(n: i32, time: i32) -> i32 {
}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} time
# @return {Integer}
def pass_the_pillow(n, time)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $time
* @return Integer
*/
function passThePillow($n, $time) {

}
}
```

Dart Solution:

```
class Solution {
int passThePillow(int n, int time) {

}
}
```

Scala Solution:

```
object Solution {
def passThePillow(n: Int, time: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec pass_the_pillow(n :: integer, time :: integer) :: integer
def pass_the_pillow(n, time) do

end
end
```

Erlang Solution:

```
-spec pass_the_pillow(N :: integer(), Time :: integer()) -> integer().
pass_the_pillow(N, Time) ->
.
```

Racket Solution:

```
(define/contract (pass-the-pillow n time)
(-> exact-integer? exact-integer? exact-integer?)
)
```