

Problem 441: Arranging Coins

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You have

n

coins and you want to build a staircase with these coins. The staircase consists of

k

rows where the

i

th

row has exactly

i

coins. The last row of the staircase

may be

incomplete.

Given the integer

n

, return

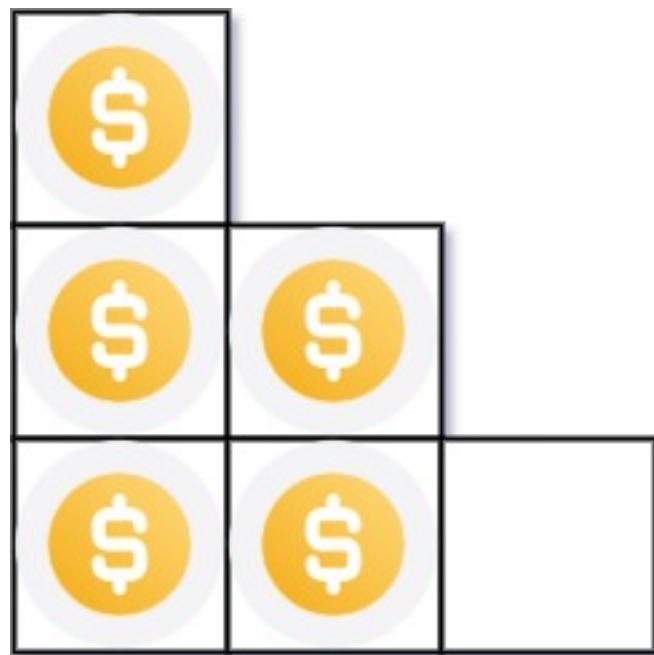
the number of

complete rows

of the staircase you will build

.

Example 1:



Input:

$n = 5$

Output:

2

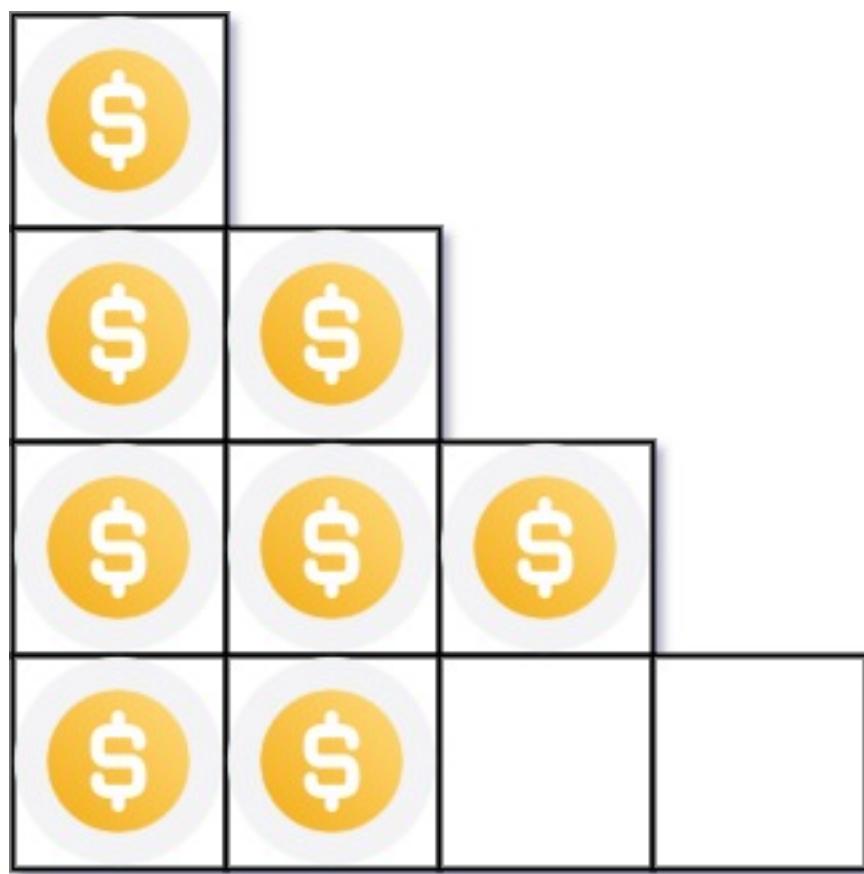
Explanation:

Because the 3

rd

row is incomplete, we return 2.

Example 2:



Input:

$n = 8$

Output:

3

Explanation:

Because the 4

th

row is incomplete, we return 3.

Constraints:

$1 \leq n \leq 2$

31

- 1

Code Snippets

C++:

```
class Solution {  
public:  
    int arrangeCoins(int n) {  
  
    }  
};
```

Java:

```
class Solution {  
public int arrangeCoins(int n) {  
  
}  
}
```

Python3:

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

Python:

```
class Solution(object):  
    def arrangeCoins(self, n):
```

```
"""
:type n: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
```

C:

```
int arrangeCoins(int n) {

}
```

Go:

```
func arrangeCoins(n int) int {

}
```

Kotlin:

```
class Solution {  
    fun arrangeCoins(n: Int): Int {  
        }  
        }  
}
```

Swift:

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

Rust:

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

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param Integer $n  
     * @return Integer  
     */  
    function arrangeCoins($n) {  
  
    }  
}
```

Dart:

```
class Solution {  
    int arrangeCoins(int n) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def arrangeCoins(n: Int): Int = {  
  
    }  
}
```

Elixir:

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

Erlang:

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

Racket:

```
(define/contract (arrange-coins n)  
  (-> exact-integer? exact-integer?)  
)
```

Solutions

C++ Solution:

```

/*
 * Problem: Arranging Coins
 * Difficulty: Easy
 * Tags: math, search
 *
 * 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 arrangeCoins(int n) {
        }
    };

```

Java Solution:

```

/**
 * Problem: Arranging Coins
 * Difficulty: Easy
 * Tags: math, search
 *
 * 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 arrangeCoins(int n) {
    }
}

```

Python3 Solution:

```

"""
Problem: Arranging Coins
Difficulty: Easy
Tags: math, search

```

```

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 arrangeCoins(self, n: int) -> int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def arrangeCoins(self, n):
"""

:type n: int
:rtype: int
"""

```

JavaScript Solution:

```

/**
 * Problem: Arranging Coins
 * Difficulty: Easy
 * Tags: math, search
 *
 * 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 arrangeCoins = function(n) {

};

```

TypeScript Solution:

```

/**
 * Problem: Arranging Coins
 * Difficulty: Easy
 * Tags: math, search
 *
 * 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 arrangeCoins(n: number): number {

};

```

C# Solution:

```

/*
 * Problem: Arranging Coins
 * Difficulty: Easy
 * Tags: math, search
 *
 * 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 ArrangeCoins(int n) {

    }
}

```

C Solution:

```

/*
 * Problem: Arranging Coins
 * Difficulty: Easy
 * Tags: math, search
 *
 * 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 arrangeCoins(int n) {  
  
}
```

Go Solution:

```
// Problem: Arranging Coins  
// Difficulty: Easy  
// Tags: math, search  
//  
// 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 arrangeCoins(n int) int {  
  
}
```

Kotlin Solution:

```
class Solution {  
    fun arrangeCoins(n: Int): Int {  
  
    }  
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Arranging Coins  
// Difficulty: Easy  
// Tags: math, search
```

```

// 
// 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 arrange_coins(n: i32) -> i32 {

}
}

```

Ruby Solution:

```

# @param {Integer} n
# @return {Integer}
def arrange_coins(n)

end

```

PHP Solution:

```

class Solution {

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

}
}

```

Dart Solution:

```

class Solution {
int arrangeCoins(int n) {

}
}

```

Scala Solution:

```
object Solution {  
    def arrangeCoins(n: Int): Int = {  
        }  
        }  
    }
```

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

```
(define/contract (arrange-coins n)  
  (-> exact-integer? exact-integer?)  
)
```