**Arranging Coins - Detailed Documentation**

**Table of Contents**



## 1. Problem Statement

You are given n coins, and you want to build a staircase where the i-th row contains exactly i coins. The staircase stops when the remaining coins are not enough to complete the next row. Return the number of complete rows of the staircase that can be built.

**Example 1:**

Input: n = 5

Output: 2

Explanation:

```
*
**
***
```

Since the third row is incomplete, the output is 2.

**Example 2:**

Input: n = 8

Output: 3

Explanation:

*

**

***

****

Since the fourth row is incomplete, the output is 3.

## 2. Intuition

The number of coins required to form k complete rows is the sum of the first k natural numbers:

$S_k = k * (k + 1) / 2$

To find the largest k such that:

$k * (k + 1) / 2 \leq n$

We can solve for k using the quadratic equation.

## 3. Key Observations

- The sum of the first k numbers follows the arithmetic sum formula:

  $S_k = k * (k + 1) / 2$

- Given n, we need to find the largest integer k such that:

  $k * (k + 1) \leq 2 * n$

- This can be solved using the quadratic formula.

## 4. Approach

i. Use the quadratic formula: The equation

$k^2 + k - 2n = 0$

is a quadratic equation of the form:

$ax^2 + bx + c = 0$

Here,

- $a = 1$
- $b = 1$
- $c = -2n$

ii. Solve for k using the quadratic formula:

k = (-1 + sqrt(1 + 8 * n)) / 2

iii. Take the integer part (floor value) to get the maximum number of complete rows.

## 5. Edge Cases

| **Case** | **Explanation** |
| --- | --- |
| n = 1 | Only 1 row can be formed. Output: 1 |
| n = 0 | No coins available. Output: 0 |
| n = 2 | Only 1 full row (*) can be formed. Output: 1 |
| Large n (e.g., $2^{31} - 1$) | Tests efficiency and integer handling. |

## 6. Complexity Analysis

Time Complexity

- The formula uses only one square root calculation: O(1)
- No loops are required.
- Overall Time Complexity: O(1)

Space Complexity

- No extra space is used.
- Overall Space Complexity: O(1)

## 7. Alternative Approaches

i. Linear Iteration (Brute Force)

- Start with k = 1 and subtract coins row by row until n < k.
- Time Complexity: O(√n).
- Inefficient for large values of n.

ii. Binary Search

- Use binary search on the range [1, n] to find k such that:

  $k * (k + 1) / 2 \leq n$

- Time Complexity: O(log n), better than brute force but still slower than O(1).

## 8. Test Cases

Basic Cases

```
assert Solution().arrangeCoins(5) == 2  # Example 1
assert Solution().arrangeCoins(8) == 3  # Example 2
assert Solution().arrangeCoins(1) == 1  # Smallest case
assert Solution().arrangeCoins(0) == 0  # Edge case with no coins
assert Solution().arrangeCoins(2) == 1  # Incomplete second row
```

Large Input

```
assert Solution().arrangeCoins(2147483647) == 65535  # Large case
```

## 9. Final Thoughts

- The mathematical approach using the quadratic formula is the best solution (O(1) time complexity).
- Alternative methods (linear, binary search) work but are less efficient.
- The formula-based solution is clean, efficient, and precise for handling large n.