

Arranging Coins

You have a total of n coins that you want to form in a staircase shape, where every k -th row must have exactly k coins.

Given n , find the total number of **full** staircase rows that can be formed.

n is a non-negative integer and fits within the range of a 32-bit signed integer.

Example 1:

$n = 5$

The coins can form the following rows:

```
✖
✖ ✖
✖ ✖
```

Because the 3rd row is incomplete, we return 2.

Example 2:

$n = 8$

The coins can form the following rows:

```
✖
✖ ✖
✖ ✖ ✖
✖ ✖
```

Because the 4th row is incomplete, we return 3.

Solution 1

The idea is about quadratic equation, the formula to get the sum of arithmetic progression is

$$\text{sum} = (x + 1) * x / 2$$

so for this problem, if we know the the sum, then we can know the $x = (-1 + \sqrt{8 * n + 1}) / 2$

```
public class Solution {  
    public int arrangeCoins(int n) {  
        return (int)((-1 + Math.sqrt(1 + 8 * (long)n)) / 2);  
    }  
}
```

written by [mysun](#) original link [here](#)

Solution 2

[JAVA] Clean Code with Explanations and Running Time [2 Solutions]

Full Solutions and Explanations

Solution 1

```
public class Solution {
    public int arrangeCoins(int n) {
        int start = 0;
        int end = n;
        int mid = 0;
        while (start <= end){
            mid = (start + end) >>> 1;
            if ((0.5 * mid * mid + 0.5 * mid ) <= n){
                start = mid + 1;
            }else{
                end = mid - 1;
            }
        }
        return start - 1;
    }
}
```

Complexity Analysis

Uniform cost model is used as Cost Model and `n` is the input number. `b` in this case would be `2`.

Time Complexity:

- Best Case $O(\log_b(n))$: With respect to the input, the algorithm will always depend on the value of input.
- Average Case $O(\log_b(n))$: With respect to the input, the algorithm will always depend on the value of input.
- Worst Case $O(\log_b(n))$: With respect to the input, the algorithm will always depend on the value of input.

Auxiliary Space:

- Worst Case $O(1)$: Additional variables are of constant size.

Algorithm

Approach: Binary Search

The problem is basically asking the maximum length of consecutive number that has the running sum lesser or equal to `n`. In other word, find `x` that satisfy the following condition:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + \dots + x \leq n$$
$$\sum_{i=1}^x i \leq n$$

Running sum can be simplified,

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

Binary search is used in this case to slowly narrow down the `x` that will satisfy the equation. Note that `0.5 * mid * mid + 0.5 * mid` does not have overflow issue as the intermediate result is implicitly autoboxed to `double` data type.

Solution 2

```
public class Solution {
    public int arrangeCoins(int n) {
        return (int) ((Math.sqrt(1 + 8.0 * n) - 1) / 2);
    }
}
```

Complexity Analysis

Uniform cost model is used as Cost Model and `n` is the input number. `b` in this case would be `2`.

Time Complexity:

- Best Case `O(1)` : With respect to the input, the algorithm will always perform basic mathematical operation that run in constant time.
- Average Case `O(1)` : With respect to the input, the algorithm will always perform basic mathematical operation that run in constant time.
- Worst Case `O(1)` : With respect to the input, the algorithm will always perform basic mathematical operation that run in constant time.

Auxiliary Space:

- Worst Case `O(1)` : No extra space is used.

Algorithm

Approach: Mathematics

The problem is basically asking the maximum length of consecutive number that has the running sum lesser or equal to `n`. In other word, find `x` that satisfy the following condition:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + \dots + x \leq n$$

$$\sum_{i=1}^x i \leq n$$

Running sum can be simplified,

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

Using quadratic formula, `x` is evaluated to be,

$$x = 1 / 2 * (-\sqrt{8 * n + 1} - 1) \text{ (Inapplicable) or } x = 1 / 2 * (\sqrt{8 * n + 1} - 1)$$

Negative root is ignored and positive root is used instead. Note that `8.0 * n` is very important because it will cause Java to implicitly autoboxed the intermediate result into `double` data type. The code will not work if it is simply `8 * n`. Alternatively, an explicit casting can be done `8 * (long) n`.

written by [ratchapong.t](#) original link [here](#)

Solution 3

```
public int arrangeCoins(int n) {  
    //convert int to long to prevent integer overflow  
    long nLong = (long)n;  
  
    long st = 0;  
    long ed = nLong;  
  
    long mid = 0;  
  
    while (st <= ed){  
        mid = st + (ed - st) / 2;  
  
        if (mid * (mid + 1) <= 2 * nLong){  
            st = mid + 1;  
        }else{  
            ed = mid - 1;  
        }  
    }  
  
    return (int)(st - 1);  
}
```

written by [larrywang2014](#) original link [here](#)

From [LeetCoder](#).