Bulb Switcher

There are *n* bulbs that are initially off. You first turn on all the bulbs. Then, you turn off every second bulb. On the third round, you toggle every third bulb (turning on if it's off or turning off if it's on). For the *n*th round, you only toggle the last bulb. Find how many bulbs are on after *n* rounds.

Example:

```
Given n = 3.

At first, the three bulbs are [off, off, off].
After first round, the three bulbs are [on, on, on].
After second round, the three bulbs are [on, off, on].
After third round, the three bulbs are [on, off, off].

So you should return 1, because there is only one bulb is on.
```

Solution 1

```
int bulbSwitch(int n) {
    return sqrt(n);
}
```

A bulb ends up on iff it is switched an odd number of times.

Call them bulb 1 to bulb n. Bulb i is switched in round d if and only if d divides i. So bulb i ends up on if and only if it has an odd number of divisors.

Divisors come in pairs, like i=12 has divisors 1 and 12, 2 and 6, and 3 and 4. Except when i is a square, like 36 has divisors 1 and 36, 2 and 18, 3 and 12, 4 and 9, and double divisor 6. So bulb i ends up on if and only if i is a square.

So just count the square numbers.

Let R = int(sqrt(n)). That's the root of the largest square in the range [1,n]. And 1 is the smallest root. So you have the roots from 1 to R, that's R roots. Which correspond to the R squares. So int(sqrt(n)) is the answer. (C++ does the conversion to int automatically, because of the specified return type).

written by StefanPochmann original link here

Solution 2

```
int bulbSwitch(int n) {
    int counts = 0;

    for (int i=1; i*i<=n; ++i) {
        ++ counts;
    }

    return counts;
}
```

Explanation:
A light will be toggled only during the round of its factors, e.g. number 6 light will be toggled at 1,2,3,6 and light 12 will be toggled at 1,2,3,4,6,12. The final state of a light is on and off only depends on if the number of its factor is odd or even. If odd, the light is on and if even the light is off. The number of one number's factor is odd if and only if it is a perfect square!
So we will only need to loop to find all the perfect squares that are smaller than n!

written by larasieh original link here

Solution 3

```
// For prime numbers, they must be off because we can reach them only twice (The f
irst round and their own round).
/* For other numbers, if we can reach them odd times, then they are on; otherwise,
they are off. So only
 those numbers who have square root will be reached odd times and there are sqrt(
n) of them because
 for every x > sqrt(n), x*x > n and thus should not be considered as the answer.
*/

 return (int)sqrt(n);
```

written by yular original link here