Below are my step by step thought process. Hope it helps

Observation #1:  
A bulb is ON if it is toggled **odd** times: (ON), (ON -> OFF -> ON), (ON -> OFF -> ON -> OFF -> ON) ...  
A bulb is OFF if it is toggled **even** times: (ON -> OFF), (ON -> OFF -> ON -> OFF ) ...

Observation #2:  
Once we reach and toggle the ***i-th*** bulb, it will never be touched anymore.

for exmaple, let n = 5 ( changes in each round denoted as bold ), initial state is all OFF:  
1-th round : **ON, ON, ON, ON, ON**  
2-nd round: ON, **OFF**, ON, **OFF**, ON  
3-rd round : ON, OFF, **OFF**, OFF, ON  
4-th round : ON, OFF, OFF, **ON**, ON  
5-th round : ON, OFF, OFF, ON, **OFF**

You can see that, after each round, all bulbs before that round will not be touched anymore.

So, given a bulb at position X, how do I know it will be toggled EVEN/ODD times ?  
Answer: We can find the number of divisors of X.

for example, if the position of a bulb is 3, its divisor is [1,3], that mean we will toggle the 3-rd bulb at 1-st round and 3-rd round:  
Combine the observation above,  
at ***3-rd*** round, the third bulb will have been toggeld even times ( [1,3] ), so it is OFF. And it will not be changed later on round.  
at ***5-th*** round, the fifth bulb will have been toggled even times ( [1,5] ), so it is OFF. And it will not be changed later on round.

Suppose we have a func to get all divisors of i : **getDivisors(int)**, a simple algorithm can be deduced:  
for i = 1..n, if getDivisors(i) % 2 == 1, count++;  
We loop through all positions, if it has odd number of divisors, it is ON and hence count + 1.

Unfortunately, it will timeout. How to optimize ?

Observation #3:  
Divisors always come in pair. For example:  
1: [1,1] --- 2: [1,2] --- 3: [1,3] --- 4: [1,4] [2,2] --- 5:[1,5] --- 6:[1,6][2,3] --- 7:[1,7]  
8: [1,8][2,4] --- 9:[1,9][3,3]

In above examples, we know that:  
Only when ***i*** has **perfect square root**, its number of divisors is ODD, because it contains duplicated divisor. In above example,  
4: [1,2,4] --- 9: [1,3,9]  
You can pick any number to examine yourself.

Here comes a better algorithm:  
for i = 1...n, if i^2 <= n, count++ ; if i^2 > n, break;  
We loop through all position, if **i^2** is <= n, that means there exists a valid bulb which has perfect square root ***i***, and we need to count that **i^2**, so count + 1. If the **i^2** exceed n, we can terminate it because we have found all ***i*** with perfect square root within n already.

Many of you might have seen the answer, so why the answer can be simply written as Math.sqrt(n) ?  
consider n = 9, the process of aforementioned algorithm will be:  
check (1 \* 1 ) <= 9, count++  
check (2 \* 2 ) <= 9, count++;  
check (3 \* 3) <= 9, count++;  
check (4 \* 4) > 9, break;

We can say, if square root of 9 is 3, that mean **1^2** and **2^2** must exists for n = 9. Hence, the Math.sqrt(i) reflect this fact.

Hope it helps