Problem 319: Bulb Switcher

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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

i

th

round, you toggle every

i

bulb. For the

n

th

round, you only toggle the last bulb.

Return

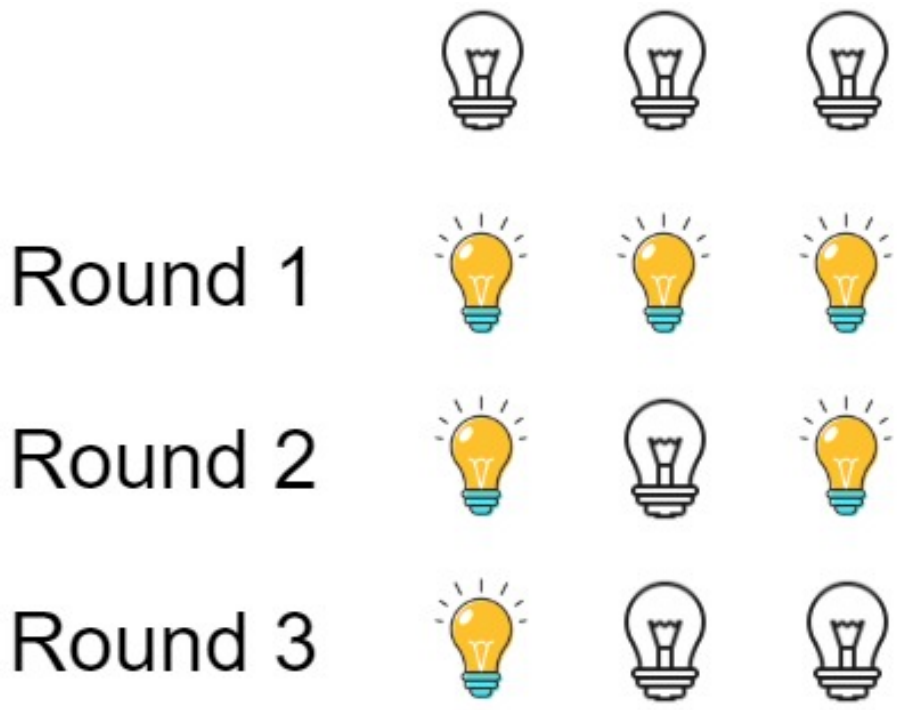
the number of bulbs that are on after

n

rounds

.

Example 1:



Input:

n = 3

Output:

1

Explanation:

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

are [on, off, off]. So you should return 1 because there is only one bulb is on.

Example 2:

Input:

n = 0

Output:

0

Example 3:

Input:

n = 1

Output:

1

Constraints:

0 <= n <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int bulbSwitch(int n) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def bulbSwitch(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

```
int bulbSwitch(int n) {

}
```

Go:

```
func bulbSwitch(n int) int {

}
```

Kotlin:

```
class Solution {
fun bulbSwitch(n: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int bulbSwitch(int n) {

}
}
```

Scala:

```
object Solution {
def bulbSwitch(n: Int): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (bulb-switch n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Bulb Switcher
* Difficulty: Medium
* Tags: math
*
* 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 bulbSwitch(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Bulb Switcher
* Difficulty: Medium
* Tags: math
*
* 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 bulbSwitch(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Bulb Switcher
Difficulty: Medium
Tags: math

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

Python Solution:

```
class Solution(object):
def bulbSwitch(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Bulb Switcher
* Difficulty: Medium
```

```
* Tags: math
*
* 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 bulbSwitch = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Bulb Switcher
* Difficulty: Medium
* Tags: math
*
* 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 bulbSwitch(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Bulb Switcher
* Difficulty: Medium
* Tags: math
*
* 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 BulbSwitch(int n) {

}
}
```

C Solution:

```
/*
* Problem: Bulb Switcher
* Difficulty: Medium
* Tags: math
*
* 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 bulbSwitch(int n) {

}
```

Go Solution:

```
// Problem: Bulb Switcher
// Difficulty: Medium
// Tags: math
//
// 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 bulbSwitch(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun bulbSwitch(n: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Bulb Switcher
// Difficulty: Medium
// Tags: math
//
// 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 bulb_switch(n: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

```
class Solution {
int bulbSwitch(int n) {

}
}
```

Scala Solution:

```
object Solution {
def bulbSwitch(n: Int): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (bulb-switch n)
(-> exact-integer? exact-integer?)
)
```