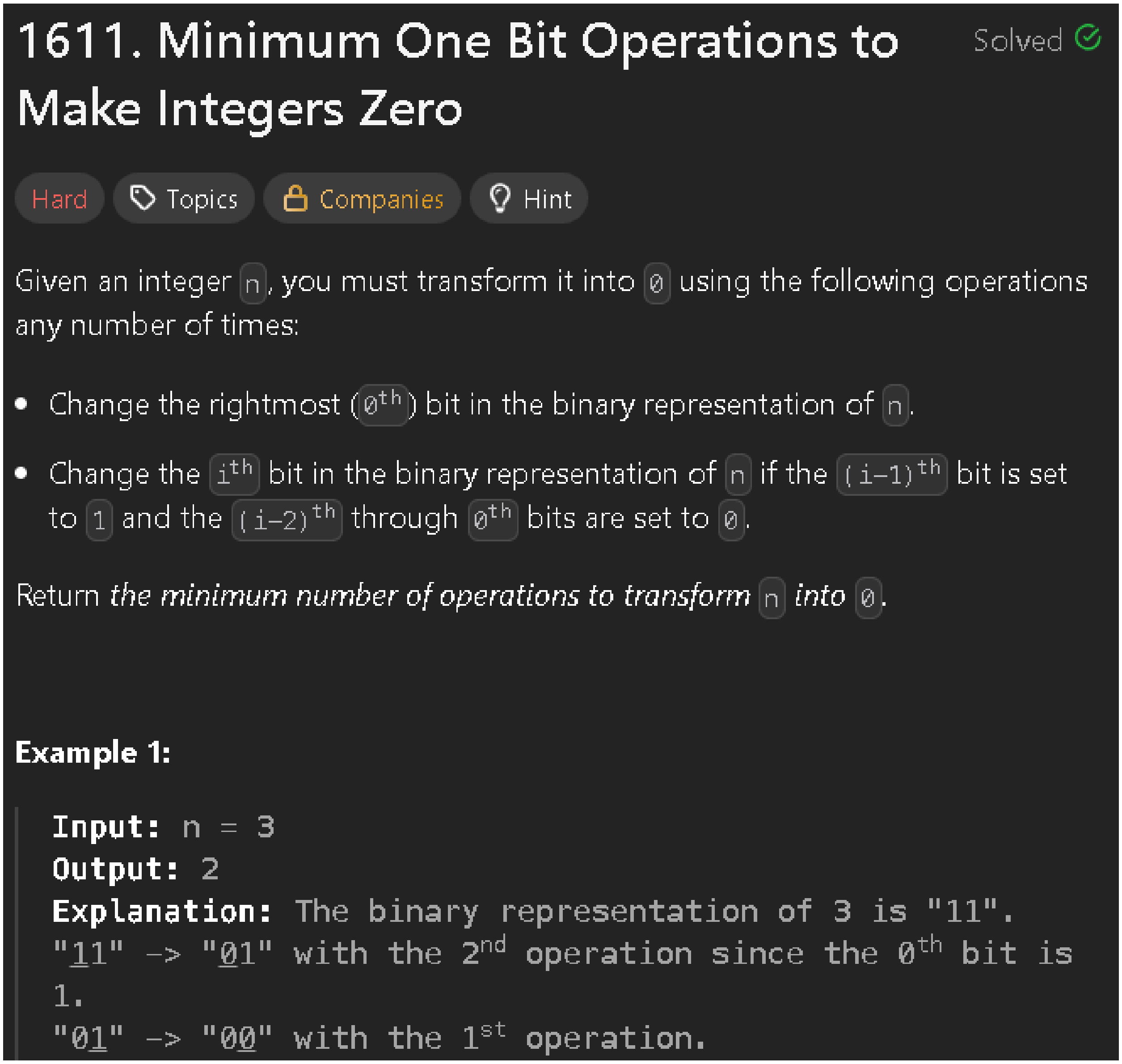


1611. Minimum One Bit Operations to Make Integers Zero

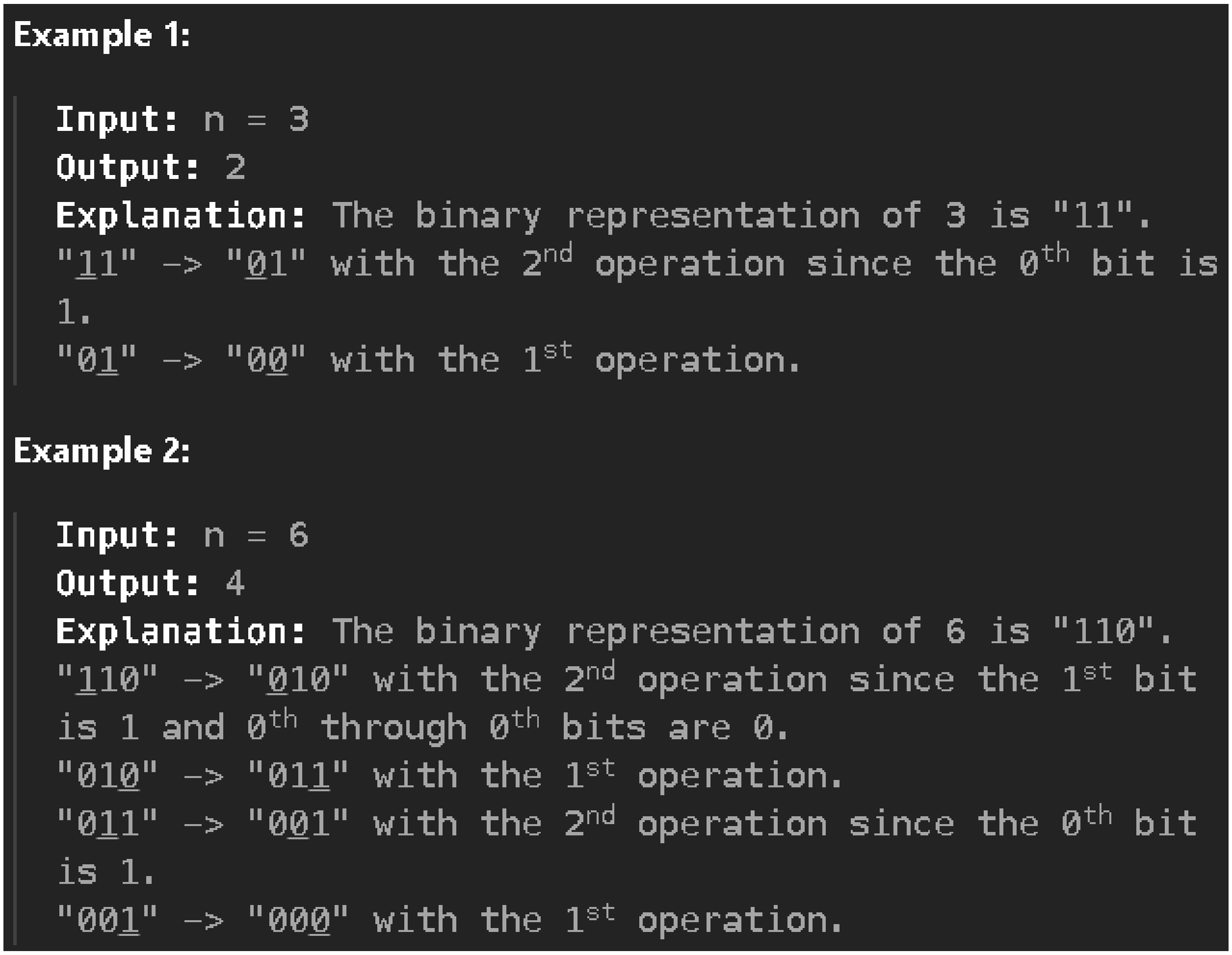
Given an integer `n`, you must transform it into `0` using the following operations any number of times:

- Change the rightmost (0^{th}) bit in the binary representation of `n`.
- Change the i^{th} bit in the binary representation of `n` if the $(i-1)^{th}$ bit is set to `1` and the $(i-2)^{th}$ through 0^{th} bits are set to `0`.

Return *the minimum number of operations to transform* `n` *into* `0`.

Example 1:

```
Input: n = 3
Output: 2
Explanation: The binary representation of 3 is "11".
"11" -> "01" with the 2nd operation since the 0th bit is
1.
"01" -> "00" with the 1st operation.
```

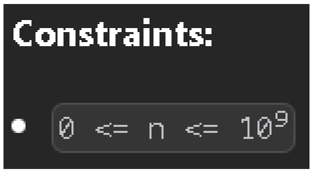


Example 1:

> **Input:** n = 3
> **Output:** 2
> **Explanation:** The binary representation of 3 is "11".
> "11" -> "01" with the 2nd operation since the 0th bit is 1.
> "01" -> "00" with the 1st operation.

Example 2:

> **Input:** n = 6
> **Output:** 4
> **Explanation:** The binary representation of 6 is "110".
> "110" -> "010" with the 2nd operation since the 1st bit is 1 and 0th through 0th bits are 0.
> "010" -> "011" with the 1st operation.
> "011" -> "001" with the 2nd operation since the 0th bit is 1.
> "001" -> "000" with the 1st operation.



Constraints:

- $0 <= n <= 10^{9}$

Python:

```
class Solution:
    def minimumOneBitOperations(self, n: int) -> int:
        res = 0
        while n:
            res = -res - (n ^ (n - 1))
            n &= n - 1
        return abs(res)
```

JavaScript:

```
/**
 * @param {number} n
 * @return {number}
 */
```

```
var minimumOneBitOperations = function(n) {
    if (n <= 1) return n;
    let count = 0;
    while ((1 << count) <= n) count++;
    return ((1 << count) - 1) - minimumOneBitOperations(n - (1 << (count - 1)));
};
```

Java:

```
class Solution {
    public int minimumOneBitOperations(int n) {
        int multiplier = 1;
        int res = 0;
        while (n > 0) {
            res += n ^ (n - 1) * multiplier;
            multiplier = -1 * multiplier;
            n &= n - 1;
        }
        return Math.abs(res);
    }
}
```