Hailstone Sequences

A famous unsolved problem in mathematics involves sequences of numbers called Hailstone sequences. To form such a sequence, you begin with some starting integer and repeatedly apply the following rule:

Let the last number in the sequence be n. If n is even, the next number is $n/2$. If n is odd, the next number is $3n+1$.

So say your first number is 7. Then, since 7 is odd, the next number is 3x7+1 = 22. Since 22 is even, the next number is 22/2 = 11, and so on. The complete sequence beginning with 7 is shown below:

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, ...

Notice that the sequence eventually reaches 1 and cycles 1, 4, 2, 1, 4, 2, 1, ... forever. But what if you were to begin with a different initial number instead of 7?

Mathematicians have conjectured that, no matter what number you begin with, you will always reach 1 and cycle as before. However, nobody has been able to prove it.

Having proved Fermat's Last Theorem merely months after Wiles did in 1993, you are determined this time to win this year's Fields medal. Thus you set out to investigate the Hailstone sequences. Your first idea is to find a relationship between the starting number and the number of steps required to reach 1. So you grab your laptop and begin programming.

Your task is, given the starting number for the sequence, to determine how many steps are required to reach 1.

Input

Input will consist of a list of positive integers, each on a separate line. At the end of the list will be the number 0.

Output

For each positive integer n in the input list, assuming your sequence begins with n, you must output the number of steps required to reach 1. Each answer must be output on a separate line. No output is required for the final 0 in the input list.

You may assume that, for each starting integer in the input, you can always reach 1. You may also assume that the integers in the sequence will grow no larger than 1,000,000,000.

Sample Input

```
7
5
1
6
0
```

Sample Output

```
16
5
0
8
```

Scoring

The score for each number in the input file will be 100% if the correct answer is written to the output file, and 0% otherwise.