--- Day 17: Spinlock ---

Suddenly, whirling in the distance, you notice what looks like a massive, pixelated hurricane: a deadly spinlock. This spinlock isn't just consuming computing power, but memory, too; vast, digital mountains are being ripped from the ground and consumed by the vortex.

If you don't move quickly, fixing that printer will be the least of your problems.

This spinlock's algorithm is simple but efficient, quickly consuming everything in its path. It starts with a circular buffer containing only the value `0`, which it marks as the current position. It then steps forward through the circular buffer some number of steps (your puzzle input) before inserting the first new value, `1`, after the value it stopped on. The inserted value becomes the current position. Then, it steps forward from there the same number of steps, and wherever it stops, inserts after it the second new value, `2`, and uses that as the new current position again.

It repeats this process of stepping forward, inserting a new value, and using the location of the inserted value as the new current position a total of `2017` times, inserting `2017` as its final operation, and ending with a total of `2018` values (including `0`) in the circular buffer.

For example, if the spinlock were to step `3` times per insert, the circular buffer would begin to evolve like this (using parentheses to mark the current position after each iteration of the algorithm):

- `(0)`, the initial state before any insertions.
- `0 (1)`: the spinlock steps forward three times (`0`, `0`, `0`), and then inserts the first value, `1`, after it. `1` becomes the current position.
- `0 (2) 1`: the spinlock steps forward three times (`0`, `1`, `0`), and then inserts the second value, `2`, after it. `2` becomes the current position.
- `0  2 (3) 1`: the spinlock steps forward three times (`1`, `0`, `2`), and then inserts the third value, `3`, after it. `3` becomes the current position.

And so on:

- `0  2 (4) 3  1`
- `0 (5) 2  4  3  1`
- `0  5  2  4  3 (6) 1`
- `0  5 (7) 2  4  3  6  1`
- `0  5  7  2  4  3 (8) 6  1`
- `0 (9) 5  7  2  4  3  8  6  1`

Eventually, after 2017 insertions, the section of the circular buffer near the last insertion looks like this:

```
1512  1134  151 (2017) 638  1513  851
```

Perhaps, if you can identify the value that will ultimately be after the last value written (`2017`), you can short-circuit the spinlock. In this example, that would be `638`.

What is the value after `2017` in your completed circular buffer?

Your puzzle answer was `1311`.

--- Part Two ---

The spinlock does not short-circuit. Instead, it gets more angry. At least, you assume that's what happened; it's spinning significantly faster than it was a moment ago.

You have good news and bad news.

The good news is that you have improved calculations for how to stop the spinlock. They indicate that you actually need to identify the value after `0` in the current state of the circular buffer.

The bad news is that while you were determining this, the spinlock has just finished inserting its fifty millionth value (`50000000`).

What is the value after `0` the moment `50000000` is inserted?

Your puzzle answer was `39170601`.

Both parts of this puzzle are complete! They provide two gold stars: **

At this point, you should return to your advent calendar and try another puzzle.

Your puzzle input was `371`.

You can also [Share] this puzzle.