--- Day 10: Knot Hash ---

You come across some programs that are trying to implement a software emulation of a hash based on knot-tying. The hash these programs are implementing isn't very strong, but you decide to help them anyway. You make a mental note to remind the Elves later not to invent their own cryptographic functions.

This hash function simulates tying a knot in a circle of string with 256 marks on it. Based on the input to be hashed, the function repeatedly selects a span of string, brings the ends together, and gives the span a half-twist to reverse the order of the marks within it. After doing this many times, the order of the marks is used to build the resulting hash.

```
  4--5   pinch   4  5           4   1
 /    \  5,0,1  / \/ \  twist  / \ / \
3      0  -->  3      0  -->  3   X   0
 \    /         \ /\ /         \ / \ /
  2--1           2  1           2   5
```

To achieve this, begin with a list of numbers from `0` to `255`, a current position which begins at `0` (the first element in the list), a skip size (which starts at `0`), and a sequence of lengths (your puzzle input). Then, for each length:

- Reverse the order of that length of elements in the list, starting with the element at the current position.
- Move the current position forward by that length plus the skip size.
- Increase the skip size by one.

The list is circular; if the current position and the length try to reverse elements beyond the end of the list, the operation reverses using as many extra elements as it needs from the front of the list. If the current position moves past the end of the list, it wraps around to the front. Lengths larger than the size of the list are invalid.

Here's an example using a smaller list:

Suppose we instead only had a circular list containing five elements, `0, 1, 2, 3, 4`, and were given input lengths of `3, 4, 1, 5`.

- The list begins as `[0] 1 2 3 4` (where square brackets indicate the current position).
- The first length, `3`, selects `([0] 1 2) 3 4` (where parentheses indicate the sublist to be reversed).
- After reversing that section (`0 1 2` into `2 1 0`), we get `([2] 1 0) 3 4`.
- Then, the current position moves forward by the length, `3`, plus the skip size, 0: `2 1 0 [3] 4`. Finally, the skip size increases to `1`.

- The second length, `4`, selects a section which wraps: `2 1) 0 ([3] 4`.
- The sublist `3 4 2 1` is reversed to form `1 2 4 3`: `4 3) 0 ([1] 2`.
- The current position moves forward by the length plus the skip size, a total of `5`, causing it not to move because it wraps around: `4 3 0 [1] 2`. The skip size increases to `2`.

- The third length, `1`, selects a sublist of a single element, and so reversing it has no effect.
- The current position moves forward by the length (`1`) plus the skip size (`2`): `4 [3] 0 1 2`. The skip size increases to `3`.

- The fourth length, `5`, selects every element starting with the second: `4) ([3] 0 1 2`. Reversing this sublist (`3 0 1 2 4` into `4 2 1 0 3`) produces: `3) ([4] 2 1 0`.
- Finally, the current position moves forward by `8`: `3 4 2 1 [0]`. The skip size increases to `4`.

In this example, the first two numbers in the list end up being `3` and `4`; to check the process, you can multiply them together to produce `12`.

However, you should instead use the standard list size of `256` (with values `0` to `255`) and the sequence of lengths in your puzzle input. Once this process is complete, what is the result of multiplying the first two numbers in the list?

To begin, get your puzzle input.

Answer: [Submit]

You can also [Share] this puzzle.