Imagine some kind of measuring apparatus that produces a long stream of bits as the result of one measurement. This could be, for instance, the observation of a distant flickering star at regular intervals of a millisecond, where a 1-bit signals visibility and a 0-bit signals non-visiblity. These bitstreams can get very long, several millions per hour in the case of our flickering star example, and will require lots of storage space. If the value of the bit doesn't change very often, that is, there are long runs of consecutive 1-bits followed by long runs of consecutive 0-bits, a substantial reduction in storage space can be achieved if only the run-lengths of the runs are stored.

In this problem we will denote the compressed bitstream as a sequence of numbers: the first number is the number of 0-bits at the start of the stream, the second is the number of 1-bits following, then the number of 0-bits following, etc. The end of the bitstream is signaled by the number 0 (zero). A zero at the start of the sequence means that the bitstream starts with a 1-bit. Examples:

- "000000111001111111100000111" is compressed as: 6 3 2 8 5 3 0.
- "1110011111000001011111111111" is compressed as: 0 3 2 5 5 1 1 10 0.

To ensure that the compressed bitstream contains no errors, we take two measures: we remember the length, L of the original bitstream, and we calculate a particular checksum of the bitstream. The checksum is determined by the bits in the bitstream and three numbers b, n and m. It is calculated as follows:

- The bitstream is divided into 'chunks' of b bits each. If the length of the bitstream is not divisible by b, we pad the last chunk with 0-bits.
- Let every chunk represent a binary number, with the leftmost bit being the most significant.
- Initialise the variable C with zero, then for every chunk, from the beginning of the stream to the end:
  - multiply C with the number n;
  - add the number that is represented by the chunk;
  - take the remainder of division by m and assign it to C.
- The final value of C is the checksum of the bitstream for a given tripplet b, n and m.

In the first example above, the length L=27. Using b=5, n=17 and m=53, we calculate the checksum as follows:

Chunks are "00000", "01110", "01111", "11110", "00001" and "11000", which represent the numbers 0,14,15,30,1 and 24. (Note the three padding 0-bits in the last chunk).

The checksum is calculated: $(0*17+0) \bmod 53 = 0$; $(0*17+14) \bmod 53 = 14$; $(14*17+15) \bmod 53 = 41$; $(41*17+30) \bmod 53 = 38$; $(38*17+1) \bmod 53 = 11$; $(11*17+24) \bmod 53 = 52$.

So the checksum is 52.

Input

Several (less than 1200) compressed bitstreams in the following format: the number L, the length of the bitstream; the triplet b, n and m, as defined above, needed to calculate the checksum; C, the checksum of the bitstream; the numbers representing the compressed bitstream. Remember that the last number is always zero.

The input of each bitstream starts on a new line and is given in one or more lines, the numbers on a line are separated by spaces.

The input is terminated by a '0' (zero) on a line by itself.

Ranges:

$(1 \le L \le 10^{10})$, $(5 \le b \le 32)$, $(1 \le n \le 4294967295)$, $(1 \le m \le 4294967295)$, $(0 \le C < m)$.

The compressed bitstream consists of up to 100000 numbers with values up to 100000.

Output

For each bitstream in the input, one line stating the bitstream number, starting from 1, and stating the status of the compressed data:

- '`Bitstream` x: `Invalid Length`', if the length of the compressed data is not equal to the value of L;
- '`Bitstream` x: `Invalid Checksum`', if the length is correct, but the checksum of the compressed data is not equal to C;
- '`Bitstream` x: `Compression OK`', if both length and checksum are correct.

Replace x with the appropriate number and omit the surrounding quotes.

Sample Input

```
27 5 17 53 52 6 3 2 8 5 3 0
28 7 65 1111 351
0 3 2 5 5 1 1 10 0
27
7 65 1111
153
0 3 2 5 5
1 1 10 0
26 5 17 53 25 6 3 2 8 5 3 0
0
```

Sample Output

```
Bitstream 1: Compression OK
Bitstream 2: Invalid Length
Bitstream 3: Invalid Checksum
Bitstream 4: Invalid Length
```