Problem 3:

How I interpreted it

I read “indices of gems she splits differ” as: once you decide how many gems you’re using, choosing *which* of those are split matters. That turns the task into counting sequences built from blocks of size **1** and **M**, where choosing a split counts distinctly. This naturally leads to a one-step/long-jump style recurrence: the ways to make **n** come from appending either a **1**-sized choice or an **M**-sized choice to smaller totals.

Initial thoughts

- A basic dynamic program over **n = 0..N** would be simple, but **N** can be up to 101810^{18}1018, so a linear loop is impossible.
- The transition only looks back by **1** and by **M**, which is a classic “linear recurrence” pattern that can be accelerated.

Final approach

I used **matrix exponentiation** on the standard companion matrix for this type of recurrence.

- Build a small **M×M** matrix that, when multiplied by a state vector holding the last **M** answers, advances the state by one step.
- Raise this matrix to the needed power using **binary exponentiation** (logarithmic number of squarings).
- Multiply once by the base vector (for totals 0..M−1). For those, the count is 1 because only 1-unit choices fit.
- Take the first entry of the resulting state; that’s the answer for **N**.

This gives time roughly proportional to **log N** with a small constant depending on **M** (at most 100), which is easily fast.

Key implementation notes

- **Modulo safety:** the first version of my matrix multiplication forgot to reduce values modulo 109+710^9+7109+7 on each accumulate, which allowed 64-bit overflow during exponentiation. Fixing this by applying the modulo in the inner loops makes it correct and safe.
- **Sparsity:** the matrix is mostly zeros; iterating in an order that skips zeros speeds things up a bit.
- **Base/edges:** handle **N = 0** (answer 1) and **N < M** (also 1) explicitly to avoid off-by-one mistakes.

How I arrived here

I started from the simple counting perspective (add a 1-block or an M-block), wrote the small DP, realized it can’t run for huge **N**, then recognized the structure as a linear recurrence. From there, the standard tool is matrix power. I validated the idea on the sample cases and a few small random cases with a brute-force DP to be sure the transition and base window were consistent.