□ Magical String Problem — Complete Documentation

1. 📌 Problem Statement

Given an integer n, construct a *magical string* and return the number of '1's in the first n characters.

Magical String Rules:

- Only consists of characters '1' and '2'.
- The string is magical because the *group lengths* of consecutive characters in the string form the string itself.

2. 💡 Intuition

This problem is essentially about generating a sequence that describes itself using group run-length encoding. Once we understand how the string builds up, we can simulate its generation and simply count the number of '1's.

3. 🔍 Key Observations

- Start with s = "122" or [1, 2, 2].
- The group lengths of 1s and 2s (i.e., [1, 2, 2, 1, 1, 2, 1, 2, ...]) form the sequence itself.
- We iterate through the string, using the values to determine how many times to append the next character.
- Alternate between appending '1' and '2'.

4. 🛠 Approach

- Initialize:
  - s = [1, 2, 2] (base of magical string)
  - i = 2 → pointer to the run-lengths

- num = 1 → number to append next
- Loop while len(s) < n:
  - Append [num] * s[i] to s
  - Toggle num between 1 and 2 using num = 3 - num
  - Move to the next group with i += 1
- After generation, return the count of '1's in the first n characters.

5. ⚠ Edge Cases

Input n	Expected Output	Notes
1	1	Only the first character '1'
2	1	First two characters are 1, 2
3	1	No extra group creation needed

Make sure to avoid early returns that assume incorrect counts.

6. 📊 Complexity Analysis

Time Complexity

- O(n): Each character is appended at most once until we reach n.

Space Complexity

- O(n): We store up to n characters of the magical string.

7. 🔄 Alternative Approaches

- Precomputation:
  - Generate the magical string once up to a large max size (e.g., 100,000), and reuse it for multiple queries.

- Streaming Count:
    - If memory is limited, count '1's while generating the string without storing the entire list.

8. ✅ Test Cases

```
assert Solution().magicalString(1) == 1
assert Solution().magicalString(2) == 1
assert Solution().magicalString(3) == 1
assert Solution().magicalString(6) == 3
assert Solution().magicalString(10) == 5
assert Solution().magicalString(20) == 10
```

9. Final Thoughts

- This problem is a creative blend of simulation and self-referential sequences.
- Avoid assumptions on early values — always simulate for accuracy.
- Can be optimized further for time/memory if needed, especially for repeated queries or larger constraints.