Recursion Practice: Tribonacci-like Series

Objective

This challenge helps you learn the concept of recursion.

A recursive function is one that calls itself. In C, recursion is supported, but you must define an exit condition (base case) to prevent infinite recursion.

Problem Description

Consider a series T_n where the next term is the sum of the previous three terms:

$$T_n = T_{n-1} + T_{n-2} + T_{n-3}, \quad \text{for } n > 3$$

Given the first three terms T_1, T_2, T_3, write a recursive function to find the n^{th} term.

Recursive Formula

$$T_n = \begin{cases} T_1 & \text{if } n = 1 \\ T_2 & \text{if } n = 2 \\ T_3 & \text{if } n = 3 \\ T_{n-1} + T_{n-2} + T_{n-3} & \text{if } n > 3 \end{cases}$$

Input Format

- First line contains an integer n denoting the term to find.
- Second line contains three space-separated integers: T_1, T_2, T_3.

Output Format

Print the n^{th} term of the series T_n.

Constraints

- $1 \leq n \leq 30$
- T_1, T_2, T_3 are integers.

Sample Input

```
5
1 2 3
```

Sample Output

```
11
```

Explanation

$$T_4 = T_3 + T_2 + T_1 = 3 + 2 + 1 = 6$$
$$T_5 = T_4 + T_3 + T_2 = 6 + 3 + 2 = 11$$

C Recursive Code Example

```c
#include <stdio.h>

int tribonacci(int n, int t1, int t2, int t3) {
    if (n == 1) return t1;
    if (n == 2) return t2;
    if (n == 3) return t3;
    return tribonacci(n-1, t1, t2, t3)
           + tribonacci(n-2, t1, t2, t3)
           + tribonacci(n-3, t1, t2, t3);
}

int main() {
```

```c
    int n, t1, t2, t3;
    scanf("%d", &n);
    scanf("%d %d %d", &t1, &t2, &t3);
    printf("%d\n", tribonacci(n, t1, t2, t3));
    return 0;
}
```

Listing 1: Recursive function to find nth term of the series

Common Problems and Variations

- **Fibonacci sequence:** Similar, but sum of previous two terms.
- **Generalized sequences:** Sum of previous k terms, where k can vary.
- **Memoization:** Optimizing recursive calls by storing already computed values to prevent exponential time complexity.
- **Iterative approach:** Calculating the n^{th} term without recursion to improve performance.
- **Large inputs:** Handling big n values that exceed integer limits, requiring use of larger data types or arbitrary precision arithmetic.

How Can It Be Harder?

- Increase n to very large values (e.g., $n > 10^6$) — recursion becomes impractical; must use dynamic programming or matrix exponentiation.
- Change the recursion relation to include multiplication or non-linear terms.
- Introduce modulo operations to keep results within bounds.
- Allow input to include negative indices or require calculating terms backward.
- Ask for the sum of multiple terms or some property of the series rather than a single term.