# Exercise 7: Financial Forecasting

## **Step 1: Understand Recursive Algorithms**

Explain the concept of recursion and how it can simplify certain problems.

**Recursion** is a fundamental programming concept where a method or function calls itself repeatedly to solve a problem. Rather than solving the entire problem at once, recursion solves **smaller sub-problems** until it reaches a **base case**, which is the condition that stops further recursive calls.

For example, calculating factorial.

* **Simpler to understand**
* **Shorter and more elegant**
* **Aligned with mathematical definitions**

In financial forecasting, **future values depend on previous values**, often through a repetitive pattern like:

Future Value = Present Value × (1 + Growth Rate)

This pattern naturally lends itself to recursion because:

* Each future year is **built on the previous year’s value**.
* One can **model compound growth or trends** using recursive steps.
* The same formula is **applied repeatedly**, making recursion an elegant approach.

#### Example:

To predict the value after n years:

FV(n) = FV(n-1) × (1 + rate)

And it continues until:

FV(0) = initial amount

This mirrors exactly how recursion operates — from future back to present — making recursion a **natural and intuitive tool** for forecasting models.

## **Step 4: Analysis**

1. Discuss the time complexity of your recursive algorithm.

The **naive recursive algorithm** we used has a time complexity of **O(n)**:

* For every future year, the method calls itself once with years - 1.
* This builds a **linear chain** of calls (like a stack): one for each year.
* For n years of forecasting, it makes **n recursive calls**.

Years = 5 → Calls: FV(5) → FV(4) → FV(3) → FV(2) → FV(1) → FV(0)

So, total calls = 6 (n + 1), making the complexity O(n).

While this is fine for **small inputs**, recursion can become inefficient or even unsafe (stack overflow) when the value of n becomes very large.

1. Explain how to optimize the recursive solution to avoid excessive computation.

To avoid excessive computation and improve performance, I can **optimize the recursive solution** in the following ways:

### **1. Memoization**

* Store previously calculated results in a dictionary or array.
* Prevents recomputation for the same inputs.
* Especially useful if forecasting involves overlapping sub-problems.

### **2. Iterative Conversion (Dynamic Programming)**

* Instead of recursion, use a loop to compute future values year-by-year.
* Avoids recursion depth and gives **better control** over memory.
* Often faster and easier to debug.

### **3. Strong Base Cases**

* Ensure base case like years == 0 is clearly defined.
* Prevents infinite recursion or crashes.
* Acts as the **foundation of recursion**.

### **4. Smart Reuse of Results**

* In scenarios where a forecast range is reused (e.g., 1–5 years), store intermediate values.
* Reduces complexity and **increases efficiency** in larger simulations.

By combining recursion with smart optimization techniques like **memoization** and **iterative logic**, I can build a **fast, memory-efficient, and scalable financial forecasting tool** suitable for modern finance applications and predictive analytics systems.