**Financial Forecasting**

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

Recursion is a method where a function calls itself to solve a smaller part of the same problem. It continues calling itself until it reaches a base case, which stops the recursion.

* **Example:**

To find factorial of n,  
factorial(n) = n \* factorial(n-1), and factorial(0) = 1 (base case).

## ****How Recursion Simplifies Problems****

* **Breaks Big Problems into Small Ones:**  
  Makes it easier to solve complex problems step-by-step.
* **Reduces Code Length:**  
  Problems like factorial, Fibonacci, and power functions can be written in fewer lines using recursion.
* **Helpful in Tree/Graph Problems:**  
  Recursion is a natural fit for problems like file directory traversal, binary trees, etc.
* **Easy to Understand Logic:**  
  Solving one small part and repeating helps in understanding and designing logic easily.
* **Discuss the time complexity of your recursive algorithm.**

In the financial forecasting code, we used a recursive method to calculate the future value for n years based on a growth rate.

public static double futureValue(double current, double rate, int years) {

if (years == 0)

return current;

return futureValue(current, rate, years - 1) \* (1 + rate);

}

* **Time Complexity: O(n)**
* The function calls itself **once for each year**, so if there are n years, it will make n recursive calls.
* Each call does a constant amount of work, so total time = **O(n)**.
* **Space Complexity: O(n)**
* Because each recursive call is added to the **call stack**, it uses extra memory.
* For n years, **n stack frames** are created, so the space complexity is also **O(n)**.
* **Explain how to optimize the recursive solution to avoid excessive computation.**

1. **Use Iteration Instead of Recursion:** Recursion uses extra stack memory and can cause **stack overflow** for large input. The same logic can be written using a simple loop:

public static double futureValueIterative(double current, double rate, int years) {

for (int i = 0; i < years; i++) {

current = current \* (1 + rate);

}

return current;

}

This version runs in **O(n)** time and **O(1)** space — no stack usage.

1. **Use Memoization:** If the function is called with the **same input multiple times**, you can **store results** in an array or map (memoization). But in forecasting, this is usually not needed because each year is unique.