**Exercise 7: Financial Forecasting**

**Scenario:**

I were developing a financial forecasting tool that predicts future values based on past data.

**1. Understand Recursive Algorithms**

**Recursion** is a programming technique where a function calls itself to solve a smaller instance of the same problem.

It simplifies many real-world problems, especially those with repetitive calculations, like financial forecasting.

**Example:** If futureValue(n) = futureValue(n - 1) \* (1 + growthRate),  
then we can calculate the value for any year using the previous year's value.

**2. Setup**

To forecast future investment value, we define a recursive method:

java public static double forecastRecursive(int year, double initialAmount, double growthRate)

**3. Analysis**

**➤ Time Complexity:**

| **Approach** | **Time Complexity** | **Space Complexity** |
| --- | --- | --- |
| Recursive | O(n) | O(n) (call stack) |
| Memoized Recursive | O(n) | O(n) (for memoization) |

**➤ Optimization:**

* Even though both approaches are O(n), memoization improves performance by:

Avoiding repeated calculations

Reducing function call overhead

Handling large inputs without stack overflow.

**Solution:**

**FinancialForecast.java**

public class FinancialForecast {

// Recursive method to calculate future value

public static double forecast(double currentValue, double growthRate, int years) {

// Base case: no years left

if (years == 0) {

return currentValue;

}

// Recursive step: apply growth rate for one year and recurse

return forecast(currentValue \* (1 + growthRate), growthRate, years - 1);

}

public static void main(String[] args) {

double initialValue = 1000.0; // Starting amount

double growthRate = 0.05; // 5% annual growth

int years = 5;

double futureValue = forecast(initialValue, growthRate, years);

System.out.printf("Future value after %d years: %.2f\n", years, futureValue);

}

}

**Output:**

