**DATA STRUCTURES AND ALGORITHMS**

**Exercise 7: Financial Forecasting**

**1. Understand Recursive Algorithms:**

**Recursion** is a programming technique where a function calls itself to solve smaller instances of the same problem. It continues this process until it reaches a **base case** — a condition that stops the recursion.

**Why it helps:**  
Recursion simplifies problems that have **repetitive patterns** or **mathematical relationships**, like:

* Growth over time (as in financial forecasting),
* Tree/graph traversal,
* Factorial, Fibonacci, etc.

In financial forecasting, the future value for a given year can be represented as:

CopyEdit

futureValue(n) = futureValue(n - 1) \* (1 + growthRate)

This naturally fits a recursive solution where each year depends on the previous year.

**2. Setup:**

Create a method to calculate the future value using a recursive approach. The method will take:

* year: number of years to forecast,
* currentValue: the current monetary value,
* growthRate: the annual growth rate.

**3. Implementation:**

**package sf;**

**public class FinancialForecasting {**

**public static double forecastValue(int year, double currentValue, double growthRate) {**

**if (year == 0) {**

**return currentValue;**

**}**

**return *forecastValue*(year - 1, currentValue, growthRate) \* (1 + growthRate);**

**}**

**public static void main(String[] args) {**

**int yearsAhead = 5;**

**double initialValue = 10000;**

**double growthRate = 0.10;**

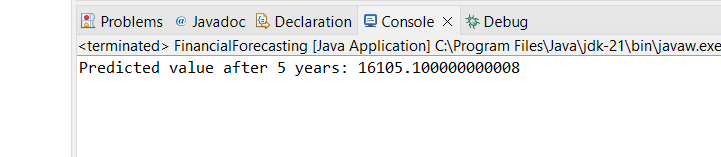
**double predictedValue = *forecastValue*(yearsAhead, initialValue, growthRate);**

**System.*out*.println("Predicted value after " + yearsAhead + " years: " + predictedValue);**

**}**

**}**

**OUTPUT:**

****

**4. Analysis:**

**Time Complexity:**

* The time complexity is **O(n)** where n is the number of years.
* Each call reduces the year by 1 until it reaches 0.

**Optimization:**

* For simple calculations like this, recursion is efficient.
* For more complex recursive problems, **memoization** or **iteration** can be used to avoid recomputation and **stack overflow** in large inputs.