**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Steps:**

1. **Understand Recursive Algorithms:**
   * Explain the concept of recursion and how it can simplify certain problems.
2. **Setup:**
   * Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
   * Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
   * Discuss the time complexity of your recursive algorithm.
   * Explain how to optimize the recursive solution to avoid excessive computation.

Recursion is a programming technique where a function calls itself to solve a problem by breaking it down into smaller, similar subproblems. It simplifies complex problems by reducing them to simpler, repeatable steps, often leading to more elegant and concise code.

**Setup:**

// Method to calculate future value recursively

static double calculateFutureValue(double currentValue, double growthRate, int years) {

if (years == 0) {

return currentValue;

}

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

**Implementation:**

public class FinancialForecasting {

// method we have setup

static double calculateFutureValue(double currentValue, double growthRate, int years) {

if (years == 0) {

return currentValue;

}

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

}

public static void main(String[] args) {

double currentValue = 1000.0;

double growthRate = 0.10;

int years = 5;

double futureValue = calculateFutureValue(currentValue, growthRate, years);

System.out.println("Future value after " + years + " years: ₹" + futureValue);

}

}

**Output:**

Future value after 5 years: ₹1610.51

**Analysis:**

Time Complexity:  
The time complexity is O(n), where n is the number of years. Each recursive call processes one year, so the function runs n times.

Optimization:  
For small n, recursion is fine. But for larger n, recursion may become slow or risk stack overflow. we can:

* Use a **loop** instead of recursion .