**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.

**Solution :**

**FinancialForecasting.java**  
public class FinancialForecasting {

public static void main(String[] args) {

double initialAmount = 1000.0; // Initial investment

double[] pastGrowthRates = {0.05, 0.03, 0.04}; // Past growth rates (5%, 3%, 4%)

int futurePeriods = 3; // Periods to forecast

// Step 1: Compute current value after applying past growth

double currentValue = computeCurrentValue(initialAmount, pastGrowthRates);

System.out.printf("Current value after past growth: $%.2f%n", currentValue);

// Step 2: Compute average growth rate

double avgGrowthRate = computeAverageGrowthRate(pastGrowthRates);

System.out.printf("Average growth rate: %.2f%%%n", avgGrowthRate \* 100);

// Step 3: Predict future value recursively

double futureValue = predictFutureValue(currentValue, avgGrowthRate, futurePeriods);

System.out.printf("Future value after %d periods: $%.2f%n", futurePeriods, futureValue);

}

// Computes current value by applying all past growth rates

private static double computeCurrentValue(double initialAmount, double[] growthRates) {

double currentValue = initialAmount;

for (double rate : growthRates) {

currentValue \*= (1 + rate);

}

return currentValue;

}

// Computes average of past growth rates

private static double computeAverageGrowthRate(double[] growthRates) {

double sum = 0;

for (double rate : growthRates) {

sum += rate;

}

return sum / growthRates.length;

}

// Recursively predicts future value

private static double predictFutureValue(double currentValue, double growthRate, int periods) {

if (periods == 0) {

return currentValue; // Base case: no future periods

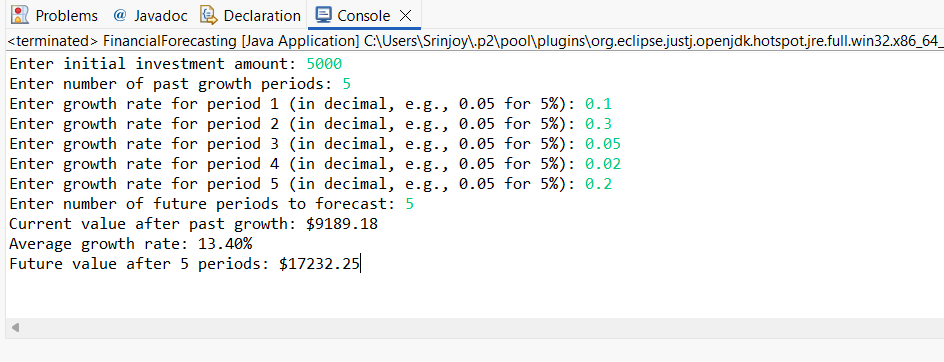
}

// Recursive case: apply growth rate and reduce periods

return predictFutureValue(currentValue \* (1 + growthRate), growthRate, periods - 1);

}

}

**Output :**

 Analysis

1. **Time Complexity**:
   * **Recursive Method**: Each recursive call reduces periods by 1 until it reaches 0. For n periods, there are n recursive calls.
   * **Complexity**: **O(n)** (linear time), where n is the number of future periods.
2. **Optimization**:
   * **Problem**: The naive recursion has **O(n)** space complexity due to recursion stack depth, risking StackOverflowError for large n.

**Solutions**:

* **Iterative Approach**: Replace recursion with a loop to avoid stack overhead:  
    
  private static double predictFutureValueIterative(double currentValue, double growthRate, int periods) {

double result = currentValue;

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

result \*= (1 + growthRate);

}

return result;

}

* **Closed-Form Formula**: Directly compute using exponentiation for **O(1)** time:  
    
  private static double predictFutureValueFormula(double currentValue, double growthRate, int periods) {

return currentValue \* Math.pow(1 + growthRate, periods);

}