**Exercise: Financial Forecasting**

**1. Understanding Recursive Algorithms**

Recursion is a programming technique where a method calls itself to solve smaller instances of the same problem. It is particularly useful for problems that can be broken down into similar sub-problems, such as calculating compound values, traversing trees, or computing Fibonacci numbers. Recursive algorithms can lead to clean and elegant code but must be used carefully to avoid excessive function calls and redundant calculations.

In the context of financial forecasting, recursion can help compute future values based on consistent growth patterns by repeatedly applying a growth formula over time.

**2. Setup**

We aim to create a method that calculates the future value of an investment recursively using:

* An initial investment amount
* A fixed annual growth rate
* A number of years

The formula we will use is:

Future Value = Initial Amount × (1 + Rate) ^ Years

Instead of using the power operator, we'll implement the repeated multiplication recursively.

**3. Implementation**

public class FinancialForecast {

public static double calculateFutureValue(double initialAmount, double rate, int years) {

if (years == 0) {

return initialAmount;

}

return (1 + rate) \* calculateFutureValue(initialAmount, rate, years - 1);

}

public static void main(String[] args) {

double initialAmount = 10000;

double annualGrowthRate = 0.08;

int years = 5;

double futureValue = calculateFutureValue(initialAmount, annualGrowthRate, years);

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

}

}

**4. Analysis**

The recursive algorithm for calculating the future value reduces the number of lines of code and makes the logic intuitive. However, for each year, the method makes a recursive call, resulting in a total of n recursive calls, where n is the number of years. Therefore, the time complexity of this recursive approach is O(n).

To optimize this recursive solution and avoid stack overflow or excessive computations, memoization could be used, although in this specific use case (a simple multiplication chain), iteration is more memory-efficient. Alternatively, an iterative solution using a loop would be more practical and avoids the function call overhead associated with recursion.