**Exercise 7: Financial Forecasting**

**1. Understand Recursive Algorithms**

**Concept of recursion:**

Recursion simplifies problems by breaking them down into smaller, more manageable sub-problems, solved by the same function calling itself.

**2. Setup**

**Create a method to calculate future value using recursion:**

public class FinancialForecasting {

// Recursive method to calculate future value

public static double futureValue(double presentValue, double growthRate, int periods) {

if (periods == 0) {

return presentValue;

}

return futureValue(presentValue \* (1 + growthRate), growthRate, periods - 1);

}

public static void main(String[] args) {

double presentValue = 1000;

double growthRate = 0.05;

int periods = 10;

double futureValue = futureValue(presentValue, growthRate, periods);

System.out.println("Future Value: " + futureValue);

}

}

**4. Analysis**

**Time complexity:**

* **Recursive solution:** O(n) - each call decreases the period by 1, leading to linear time complexity.

**Optimization:**

To avoid excessive computation, consider using **memoization** or **iterative solutions**. Memoization stores already computed results to avoid redundant calculations.

import java.util.HashMap;

public class FinancialForecasting {

private static HashMap<Integer, Double> memo = new HashMap<>();

// Optimized recursive method with memoization

public static double futureValue(double presentValue, double growthRate, int periods) {

if (periods == 0) {

return presentValue;

}

if (memo.containsKey(periods)) {

return memo.get(periods);

}

double result = futureValue(presentValue \* (1 + growthRate), growthRate, periods - 1);

memo.put(periods, result);

return result;

}

public static void main(String[] args) {

double presentValue = 1000;

double growthRate = 0.05;

int periods = 10;

double futureValue = futureValue(presentValue, growthRate, periods);

System.out.println("Future Value: " + futureValue);

}

}