**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.

**Answers:**

1.**Recusive Algorithms:**

* Recursion is a programming technique where a function calls itself to solve smaller instances of the same problem.
* Example: Problems like calculating factorial, Fibonacci series, or future value over time can be simplified using recursion.

2.**Setup:**

Future Value = Present Value \* (1 + rate)^n

Where:

* rate is the growth rate per period (e.g., 5% → 0.05)
* n is the number of future periods

3**.Implementation:**

//file name FinancialForecast.java

public class FinancialForecast {

// Recursive method to calculate future value

public static double calculateFutureValue(double presentValue, double rate, int periods) {

if (periods == 0) {

return presentValue;

}

return calculateFutureValue(presentValue, rate, periods - 1) \* (1 + rate);

}

public static void main(String[] args) {

double presentValue = 10000; // Initial investment

double growthRate = 0.05; // 5% annual growth

int years = 5; // Forecast for 5 years

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

System.out.println("Initial Investment : " + presentValue);

System.out.println("Annual Growth Rate : " + (growthRate \* 100) + "%");

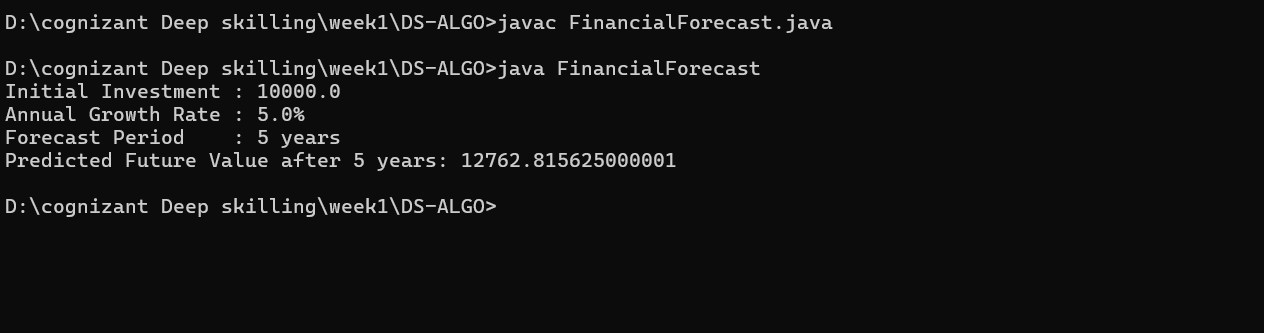
System.out.println("Forecast Period : " + years + " years");

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

}

}

**Output:**

****

4**. Analysis**:

**Time Complexity**

* **Recursive function time complexity**: O(n)
* Because the function calls itself n times (where n is the number of periods).

**Space Complexity**

* **O(n)** due to the call stack used in recursion.

**Optimization Idea**

Recursion can be inefficient for large n because of:

* Repeated function calls
* Stack overflow risk
* This version uses a **loop**, avoiding deep recursion.
* **Time Complexity** is still O(n), but **space complexity** becomes O(1).