**Exercise 7: Financial Forecasting**

**Understanding Recursive Algorithms:**

**Recursion** is a programming technique where a method calls itself to solve a smaller instance of the same problem.

It typically has:

* A **base case** to stop recursion.
* A **recursive case** that breaks the problem down and calls itself.

**How Recursion Simplifies Problems:**

* Makes problems like factorial, Fibonacci, tree traversal, etc. more elegant.
* Divides a large problem into smaller, repeatable sub-problems.
* Cleaner and more intuitive for problems naturally defined in terms of smaller versions of themselves.

**Future Value Calculation**

Let’s say we have:

* A starting value.
* A fixed annual growth rate.
* A number of years.

And we want to predict the value after *n* years using recursion:

Formula:  
FutureValue = PresentValue × (1 + growthRate) ^ years

**Implementation:**

**FinancialForecast.java**

import java.util.Scanner;

public class FinancialForecast {

// Recursive method to calculate future value

public static double calculateFutureValue(double presentValue, double growthRate, int years) {

// Base case: no more years to grow

if (years == 0) {

return presentValue;

}

// Recursive case: increase value and reduce years

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

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the present value (₹): ");

double presentValue = scanner.nextDouble();

// Get user input for growth rate in percentage

System.out.print("Enter the annual growth rate (%): ");

double growthRatePercent = scanner.nextDouble();

// Convert percentage to decimal

double growthRate = growthRatePercent / 100;

// Get user input for number of years

System.out.print("Enter the number of years: ");

int years = scanner.nextInt();

// Call recursive function

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

// Display result

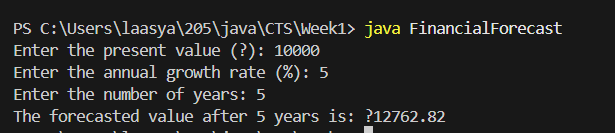
System.out.printf("The forecasted value after %d years is: ₹%.2f\n", years, futureValue);

scanner.close();

}

}

**OUTPUT:**

****

**Analysis:**

**Time Complexity:**

* **T(n) = T(n-1) + O(1)**
* This is a **linear recursive call** reducing years by 1 each time, so:  
  **Time Complexity = O(n)**  
  (where n = number of years)

**How to Optimize Recursive Solution:**

**Problem with Recursion:**  
If years is very large, the recursion stack grows too deep — leading to stack overflow risks.

**Optimization Approaches:**

1. **Use Iteration** instead of recursion (since this is essentially a repetitive multiplication task).
2. **Use Exponentiation by Squaring** (a divide and conquer method) to reduce time complexity to **O(log n)**.

**Example: Optimized Using Exponentiation by Squaring**

public static double calculateFutureValueOptimized(double presentValue, double growthRate, int years) {

double multiplier = power(1 + growthRate, years);

return presentValue \* multiplier;

}

// Fast power function (Exponentiation by Squaring)

public static double power(double base, int exponent) {

if (exponent == 0) return 1;

if (exponent % 2 == 0) return power(base \* base, exponent / 2);

else return base \* power(base, exponent - 1);

}

Time Complexity = O(log n)