**Exercise 7: Financial Forecasting**

**1. Understanding Recursive Algorithms**

What is Recursion?

* Recursion is a technique where a method calls itself to solve smaller instances of the same problem.
* Recursion is particularly useful when a problem can be broken down into subproblems that resemble the original.
* Common use cases: factorial, Fibonacci sequence, tree traversal, financial growth modeling, etc.

Why Recursion for Financial Forecasting?

* Recursive functions can model future values using repeated application of a formula, such as compound interest or exponential growth.
* This makes recursive logic ideal for simulating year-over-year financial growth.

**2. Setup**

We need a method that calculates future value recursively based on:

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

**3. Implementation**

**CODE:**

import java.util.Scanner;

public class Forecast {

    static double calculateFutureValue(double principal, double rate, int years) {

        if (years == 0) {

            return principal;

        }

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

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the initial investment amount (₹): ");

        double initialAmount = scanner.nextDouble();

        System.out.print("Enter the annual growth rate (in percentage, e.g., 5 for 5%): ");

        double growthRatePercent = scanner.nextDouble();

        double growthRate = growthRatePercent / 100;

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

        int numberOfYears = scanner.nextInt();

        double futureValue = calculateFutureValue(initialAmount, growthRate, numberOfYears);

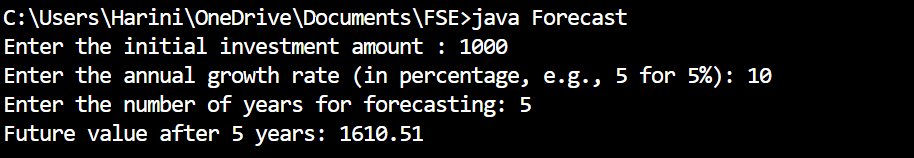
        System.out.printf("Future value after %d years: ₹%.2f%n", numberOfYears, futureValue);

        scanner.close();

    }

}

**OUTPUT:**

****

**4. Analysis**

**Time Complexity**

* The recursive function performs one call per year, so:
  + Time Complexity = O(n) where n is the number of years.

Problem: Redundant Computation

* If we were calculating multiple values (e.g., for different years), repeating calculations can become inefficient.
* For example, calculating year 10 requires year 9, which requires year 8, and so on — leading to repeated work if not handled carefully.

**Optimization Techniques:**

**Use Memoization or Iteration**

**Iterative Alternative (Better for Large Inputs)**

static double calculateFutureValueIterative(double principal, double rate, int years) {

double result = principal;

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

result \*= (1 + rate);

}

return result;

}

**Memoization (when calling recursively multiple times with overlapping inputs):**

static double[] memo;

static double calculateFutureValueMemo(double principal, double rate, int years) {

if (years == 0) return principal;

if (memo[years] != 0) return memo[years];

memo[years] = calculateFutureValueMemo(principal, rate, years - 1) \* (1 + rate);

return memo[years];

}