Exercise 7 – Financial Forecasting

# Objective

To implement a recursive algorithm that predicts future financial values based on past growth rates and understand how recursion can be used for forecasting problems.

# 2. Problem Statement / Scenario

You are developing a **financial forecasting tool** that predicts the **future value of an investment or account** based on historical growth trends. The growth is assumed to be **compounded annually** at a fixed rate. The tool should implement both a **recursive solution** and discuss optimization techniques to handle larger input sizes efficiently.

# 3.Approach / Steps

**3.1 Understand Recursive Algorithms**

* **Recursion** is a technique where a function calls itself to break down a larger problem into smaller sub-problems
* It's especially useful in problems that follow a repeatable pattern, like computing factorials, Fibonacci numbers, or compound interest over time.
* In financial forecasting, recursion can represent the process of adding growth year after year

## 3.2 Setup

* Create a Product class with attributes such as productId, productName, and category.

**3.3 Implementation**

* FutureValue = PresentValue \* (1 + rate) ^ years

# 4.Code

**FinancialForecast.java**

public class FinancialForecast {

public static double futureValue(double presentValue, double rate, int years) {

if (years == 0) return presentValue;

return futureValue(presentValue \* (1 + rate), rate, years - 1);

}

public static double futureValueIterative(double presentValue, double rate, int years) {

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

presentValue \*= (1 + rate);

}

return presentValue;

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

## double presentValue = 10000;

## double rate = 0.05;

## int years = 10;

## double future = FinancialForecast.futureValue(presentValue, rate, years);

## System.out.printf("Future value after %d years (Recursive): %.2f\n", years, future);

## double futureIter = FinancialForecast.futureValueIterative(presentValue, rate, years);

## System.out.printf("Future value after %d years (Iterative): %.2f\n", years, futureIter);

## }

## }

# Analysis

**Time Complexity**:

****Recursive version**:**

* Time complexity: **O(n)** —> where n = years
* Each recursive call processes 1 year until reaching 0.

****Iterative version**:**

* Also **O(n)**, but more efficient in practice as it avoids function call overhead.

# 6. Conclusion

# Recursive algorithms can elegantly solve forecasting problems where each year's value builds upon the previous one. While recursion makes the code simple and expressive, it’s important to recognize when to switch to iterative solutions for performance and memory efficiency, especially for large timeframes.

# 7. Output (Screenshot)

Below is the output of the program :

