**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**

**Step 1: Understand Recursive Algorithm**

**Recursion** is a technique where a function calls itself to solve smaller subproblems of the original problem. It is particularly useful when a problem can be **broken down into repetitive or nested substructures**, such as computing factorials, Fibonacci numbers, or forecasting values over time.

In the context of **financial forecasting**, recursion can simplify the process of projecting future values by repeatedly applying a **growth rate** formula to calculate the next value based on the previous one. Instead of using loops, recursion allows us to define the forecasting problem in a **declarative and cleaner manner**, reflecting the real-world logic more intuitively.

## **Step 2: Setup**

We will define a method to forecast the future value of an investment using a **recursive approach** based on:

* Current value
* Growth rate (as a percentage)
* Number of periods (e.g., years or months)

## **Step 3: Implementation**

**File Name: Financial.java**

**package** myproject.financial;

**import** java.util.Scanner;

**public** **class** Forecast {

**public** **static** **double** futureVal(**double** pVal, **double** grp, **int** years) {

**if** (years == 0) {

**return** pVal;

}

**double** nVal = pVal \* (1 + grp / 100.0);

**return** *futureVal*(nVal, grp, years - 1);

}

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter present value (e.g., 10000): ");

**double** pVal = scanner.nextDouble();

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

**double** growthRate = scanner.nextDouble();

System.***out***.print("Enter number of years to forecast (e.g., 10): ");

**int** years = scanner.nextInt();

**double** res = *futureVal*(pVal, growthRate, years);

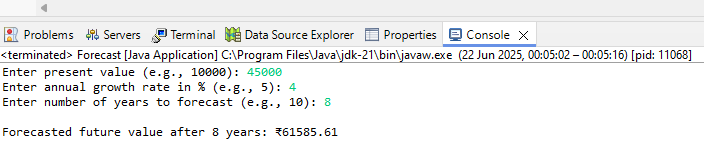
System.***out***.printf("\nForecasted future value after %d years: ₹%.2f\n", years, res);

scanner.close();

}

}

**Output:**



**Step 4: Analysis**

### Time Complexity of the Recursive Algorithm

The recursive function futureVal() makes one recursive call per year, decreasing the year count until it reaches zero. Therefore, the time complexity is:

* Time Complexity: O(n)  
   Where n is the number of years (or periods).

This means that as the number of periods increases, the number of recursive calls also increases linearly.

**To optimize the recursive solution:**

In this financial forecasting scenario, recursion is simple and intuitive but not scalable for large year counts. To optimize the recursive solution and avoid excessive stack usage, it is best to convert the **recursion into an iterative solution**. Iteration reduces space complexity from O(n) to O(1) and prevents potential stack overflow errors, making it more suitable for high-performance financial tools.