**Financial Forecasting**

**Cognizant Deepskilling Assessment**

By Vaishnavi K

22/6/2025

Understand Recursive Algorithms:

Recursion is when a function calls itself to solve a problem. Instead of looping or repeating instructions, the function breaks down a big problem into smaller, similar sub‑problems until it reaches a base case (a point where it can stop).

Recursion is helpful because it can make complex problems **easier to understand** and write. Instead of managing lots of loops and variables, you can solve a problem by focusing on:

**Base Case** – The simplest version of the problem.  
**Recursive Case** – Reducing the problem step‑by‑step until it reaches the base case.

**Example:**  
Think of forecasting the future value of an investment. To find the value after n years, you can think:

The value after n years = the value after n-1 years **× (1 + rate)**

The value after 0 years = starting amount (base case)

With recursion, you can directly **describe this relationship** instead of using long loops.

We want to **calculate the future value** of an investment using a **recursive approach**.  
 Future value depends on:

The current value

The growth rate

The number of years

Future Value = Current Value × (1 + Growth Rate) ^ Number of Years

We can implement this recursively:

**Base Case**: If years == 0, the future value is just the current value.

**Recursive Case**: futureValue(value, rate, years) = futureValue(value, rate, years-1) \* (1 + rate)

Now I’ll attach my code and snapshots,

**Main.java:**

package com.example.financial;

public class Main {

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

        if (years == 0) {

            return currentValue;

        }

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

    }

    public static void main(String[] args) {

        double currentValue = 1000.0;

        double rate = 0.05;

        int years = 10;

        double result = futureValue(currentValue, rate, years);

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

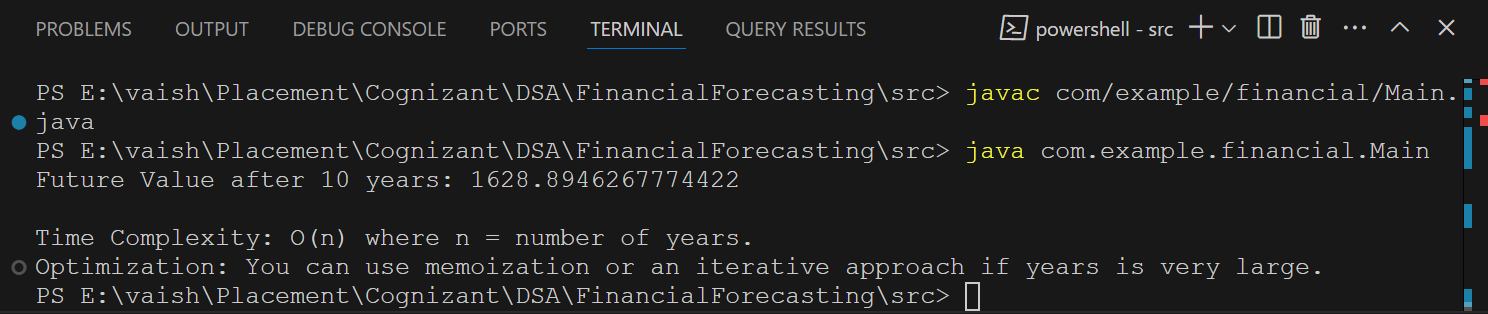
        System.out.println("\nTime Complexity: O(n) where n = number of years.");

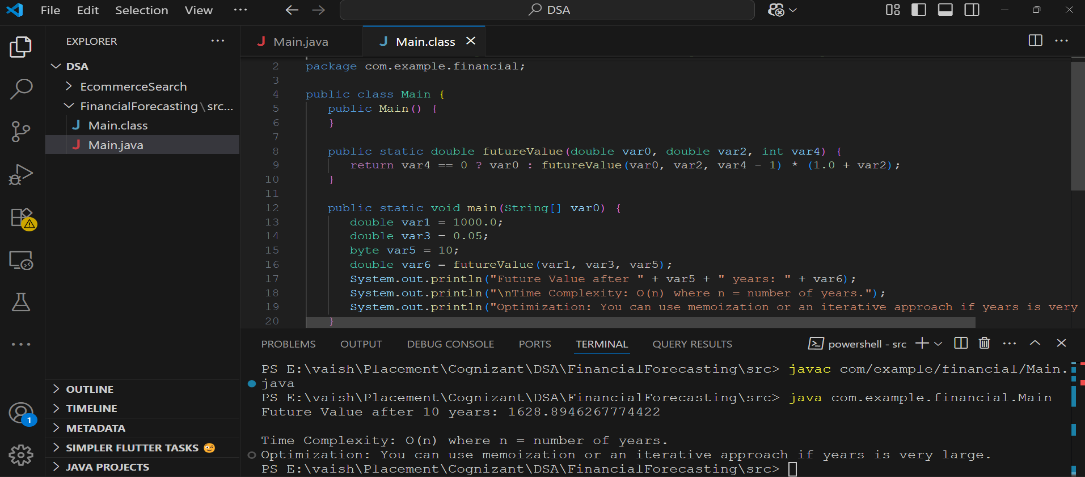
        System.out.println("Optimization: You can use memoization or an iterative approach if years is very large.");

    }

}

**Output:**





**From my analysis,**

| **Algorithm** | **Time Complexity** |  |  |
| --- | --- | --- | --- |
| Recursive Method | O(n) |  |  |
|  |  |  |  |

**How to Optimize the Recursive Solution to Avoid Excessive Computation?**

Recursion is elegant, but for very large inputs (e.g., many years), it can become **inefficient** or cause a **stack overflow**. Here are ways to optimize it:

**Use an Iterative Loop:**  
Instead of calling the method repeatedly, use a simple for or while loop. This removes the overhead of recursive calls and is ideal for long‑term forecasting.

**Apply Memoization:**  
If the calculation involves repeating the same results (in more complex scenarios), **cache** the results. This saves time by not repeating expensive calculations.

**Use the Closed‑Form Equation:**  
For this specific problem (futureValue = currentValue \* (1 + rate)^years), skip recursion and compute the result in one step using **Math.pow()**. This gives an instant result with **O(1)** time.