**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.

## Answer:

Recursion is a method where a function solves smaller versions of a problem by calling itself. It's useful in forecasting to simplify repeated calculations across multiple time periods.

FinancialForecast.java

public class FinancialForecast {

    public double recursiveForecast(double p, double[] r, int prd){

        if (prd == 0) {

            return p;

        }

        double rate = r[(prd - 1) % r.length];

        double prev = recursiveForecast(p, r, prd - 1);

        return prev \* (1 + rate);

    }

}

FinancialForestOptimised.java

public class FinancialForestOptimised {

    public double iterativeForecast(double p, double[] r, int prd) {

        double result = p;

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

            result \*= (1 + r[i % r.length]);

        }

        return result;

    }

}

Main.java

public class Main {

    public static void main(String[] args) {

        FinancialForecast f = new FinancialForecast();

        double[] r = {0.05, 0.03};

        double amt = 1000.0;

        int y = 4;

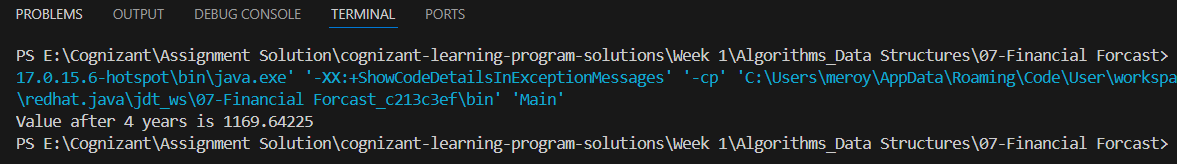
        double res = f.recursiveForecast(amt, r, y);

        System.out.printf("Value after "+ y + " years is "+ res);

    }

}

Output:



The recursive algorithm has a time complexity of **O(n)**, where *n* is the number of periods. Each call depends on the result of the previous period, forming a linear chain of recursive calls.

While there are no overlapping subproblems (so memoization isn’t needed), the recursive depth can grow with large inputs, risking stack overflow. To optimize, use an **iterative approach** instead. It eliminates recursion, reduces memory usage, and improves performance in practical applications.