**Exercise 7: Financial Forecasting**

**Scenario:**  
You are developing a financial forecasting tool that predicts future values based on past data.

**1. Understand Recursive Algorithms**

**Recursion:**

* A function calls itself to solve a smaller subproblem of the original problem.
* Helps simplify problems with repetitive structures.
* Requires a **base case** to stop recursion.

**Why Use Recursion for Forecasting?**

* Forecasting can involve computing the next value based on the previous one repeatedly.
* Recursive algorithms mirror this structure naturally.

**2. Setup**

Assume we want to forecast future values using a **compound interest**-like growth model:

Future Value = Present Value \* (1 + growthRate)^n

Where:

* growthRate is a fixed rate (e.g., 0.05 for 5%)
* n is the number of years into the future

**3. Implementation**

public class FinancialForecast {

// Recursive method to calculate future value

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

if (years == 0) {

return presentValue;

}

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

}

// Optimized version using memoization

public static double calculateFutureValueMemo(double presentValue, double rate, int years, Double[] memo) {

if (years == 0) return presentValue;

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

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

return memo[years];

}

public static void main(String[] args) {

double presentValue = 1000;

double rate = 0.05;

int years = 5;

System.out.println("Recursive Future Value: " + calculateFutureValue(presentValue, rate, years));

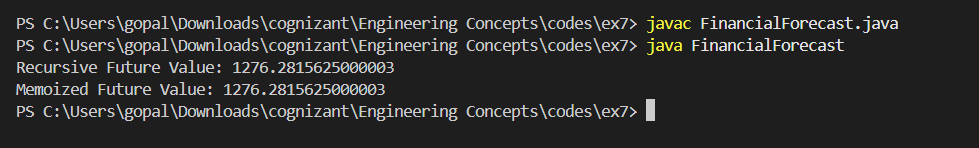
Double[] memo = new Double[years + 1];

System.out.println("Memoized Future Value: " + calculateFutureValueMemo(presentValue, rate, years, memo));

}

}

**Output:**



**4. Analysis**

**Time Complexity:**

* Recursive method: O(n)
* Memoized version: O(n), but with reduced repeated computation.

**Optimization:**

* Use **memoization** to cache results of subproblems.
* Alternatively, an **iterative solution** would also be optimal in terms of space.

**When to Use Recursion:**

* When the problem has a clear recursive structure.
* For problems where intermediate results are reusable, consider memoization.

**5.Conclusion:** Recursion is elegant but may lead to **stack overflow** for large n. Use iterative solutions in such cases.