**DATASTRUCTURES AND ALGORITHMS:**

**Exercise 7:Financial Forecasting**

**Step 1:**

**Recursion** is a programming technique where a function **calls itself** to solve a smaller version of the problem until it reaches a base case.

**Why use recursion?**  
Recursion helps simplify problems that can be broken down into smaller, repetitive tasks. For example, calculating compound growth over several years can be viewed as the same calculation applied multiple times.

**Step 2: Setup – Future Value Formula**

To forecast future value with a fixed annual growth rate, we use:

**Formula:**

**FV = PV × (1 + r)^n**

Where:

* **FV** = Future Value
* **PV** = Present Value
* **r** = Growth Rate (as decimal)
* **n** = Number of periods (years)

We can solve this using recursion:

**FV(n) = FV(n - 1) \* (1 + r)**

**Step 3: Implementation – Recursive Forecasting**

**FinancialForecasting.java**

public class FinancialForecasting {

// Recursive method to calculate future value

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

if (years == 0) {

return presentValue; // Base case

} else {

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

}

}

public static void main(String[] args) {

double presentValue = 10000; // Starting amount

double annualGrowthRate = 0.08; // 8% growth

int years = 5; // Forecasting for 5 years

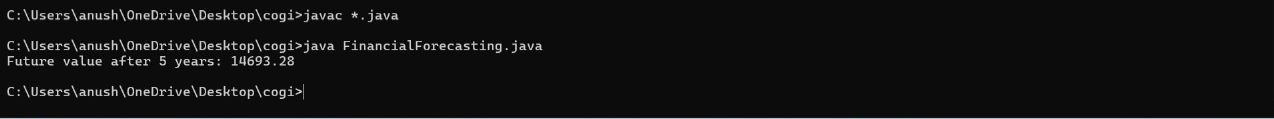
double result = futureValue(presentValue, annualGrowthRate, years);

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

}

}

**OUTPUT:**



**Step 4: Analysis**

**Time Complexity:**

* The recursive function has **O(n)** time complexity because it makes one recursive call for each year.

**Problem with Basic Recursion:**  
If the number of years is large, recursion might cause a **stack overflow** or slow performance due to **repeated function calls**.

**Optimizing Recursive Solution**

1. **Use Memoization:** Store already-computed values in a cache so we don’t recalculate them.
2. **Use Iterative Approach:** For large inputs, converting to a loop is more efficient and avoids stack overflow.
3. **FinancialForecasting.java**

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

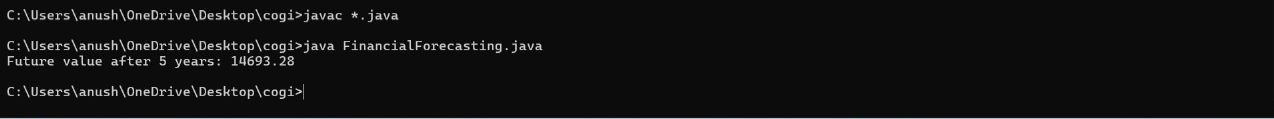
for (int i = 0; i < years; i++) {

presentValue \*= (1 + rate);

}

return presentValue;

}



**Conclusion:**

* Recursion helps write cleaner, logical code for financial forecasting problems.
* However, it must be optimized (or replaced with iteration) for large data to avoid performance issues.