**WEEK -1  
DATA STRUCTURES AND ALORITHM**

**Exercise 7**: Financial Forecasting  
**UNDERSTAND RECURSIVE ALGORITHMS**

1. What is Recursion?

* Recursion is a technique where a method calls itself to solve a smaller version of a problem.
* It consists of:
  + Base Case: The condition where recursion stops.
  + Recursive Case: The condition where the function calls itself.

1. Why use Recursion?

* Helps solve problems that have a repetitive substructure (e.g., factorial, Fibonacci, tree traversal).
* Leads to simpler and cleaner code for problems that are naturally recursive.
* Can be used to break down complex problems into smaller, more manageable ones.

**CODE:**

**PROJECT NAME:** FinancialForecasting

**CLASS NAME:** ForecastCalculator  
ForecastCalculator.java:  
public class ForecastCalculator {

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

if (years == 0) {

return presentValue;

}

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

}

public static void main(String[] args) {

double presentValue = 10000;

double rate = 0.1;

int years = 5;

double futureValue = futureValueRecursive(presentValue, rate, years);

System.out.printf("Future Value after %d years = ₹%.2f%n", years, futureValue);

}

}

**ANALYSIS:  
1. Time Complexity of Recursive Forecasting:**

* In the given recursive method to calculate future value:

FV(n) = FV(n−1) × (1+r)

The recursive call is made **once per year**, so:

* + **Time Complexity**: **O(n)** → where n is the number of years
  + **Space Complexity**: **O(n)** due to call stack usage
    1. **Optimization Technique:**
* Recursion can be **inefficient** due to:
  + High memory usage (stack frames)
  + Risk of **stack overflow** with large inputs
    1. **Optimized Solution: Iteration**
* Use a **loop** to calculate the future value:
  + No function calls, so **constant space** usage
  + **Time Complexity** remains **O(n)**, but memory usage improves to **O(1)**