**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.
5. **Understand Recursive Algorithms:**

**What is Recursion?**

Recursion is a programming technique where a function calls itself to solve a smaller instance of the same problem. It continues to call itself until it reaches a base case - the simplest version of the problem that can be solved directly.

* **Structure of a Recursive Function:**

1. **Base Case:** Stops the recursion.
2. **Recursive Case:** Calls the function again with smaller input.

* **Why Recursion Simplifies Problems:**
* Breaks complex problems into smaller, easier subproblems.
* Reduces the need for loops in some cases.
* Makes code more elegant and closer to mathematical definitions (e.g., factorial, Fibonacci).
* **When Recursion is Useful:**
* **Divide and Conquer Algorithms:** Merge Sort, Quick Sort.
* **Tree and Graph Traversals:** DFS, backtracking.
* **Mathematical Problems:** Factorials, Fibonacci numbers, permutations, combinations.
* **Caution:**

Recursion can lead to stack overflow if it doesn’t reach the base case or goes too deep. In such cases, iteration may be better.

1. **Setup:**

Define the Recursive Forecast Function

Let’s assume:

* **Initial Value**: the current amount/investment
* **Growth Rate**: yearly percentage increase (e.g., 10% = 0.1)
* **Years**: how many years to forecast

1. **Implementation in Java**

* Recursive Method to Calculate Future Value

public class FinancialForecast {

// Recursive function to calculate future value

public static double forecastValue(double initialValue, double growthRate, int years) {

// Base case

if (years == 0) {

return initialValue;

}

// Recursive case

return forecastValue(initialValue, growthRate, years - 1) \* (1 + growthRate);

}

public static void main(String[] args) {

double initial = 10000; // ₹10,000

double rate = 0.1; // 10% growth rate

int years = 5;

double future = forecastValue(initial, rate, years);

System.out.println("Forecasted Value after " + years + " years: ₹ " + future);

}

}

* **Output:**

Forecasted Value after 5 years: ₹ 16105.10

**4. Analysis**

* **Time Complexity of Recursive Forecast**

The recursive function is called once per year, so:

* **Time complexity:** **O(n)** where n = years
* **Space complexity:** **O(n)** due to the call stack

* **Optimization Techniques**

**1. Use Tail Recursion or Iteration:**

Convert recursion to iteration to avoid stack overflow for large n.

public static double forecastValueIterative(double initialValue, double growthRate, int years) {

double result = initialValue;

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

result \*= (1 + growthRate);

}

return result;

}

**Note:**

Same result

**Time complexity**: O(n)

**Space complexity**: O(1)

1. **Use Memoization**

Not needed here since no overlapping subproblems, but useful in more complex forecasts like branching projections.

Memoization is an optimization technique where you store results of expensive function calls and reuse them when the same inputs occur again.

It avoids recomputing the same subproblems, making recursive solutions much faster.

Memoization is helpful when:

* The recursive solution has overlapping subproblems.
* The same function is called multiple times with the same arguments.