**Exercise 7: Financial Forecasting**

Explain the concept of recursion and how it can simplify certain problems.

Recursion is a programming technique where a function calls itself repeatedly until it reaches a base case that stops the recursion. The function solves a problem by breaking it down into smaller subproblems of the same type, which are then solved by the same function, until the solution to the original problem is found.

Recursion can simplify certain problems in several ways:

1. Divide and Conquer: Recursion allows you to break down complex problems into smaller, more manageable subproblems, making it easier to solve them.

2. Elegant Code: Recursive solutions can be more concise and elegant than iterative solutions, making the code easier to read and maintain.

3. Reduced Code Duplication: Recursion can reduce code duplication by avoiding the need for explicit loops and conditional statements.

4. Solves Tree-like Problems: Recursion is particularly useful for solving problems that have a tree-like structure, such as traversing a tree or graph.

5. Solves Dynamic Programming Problems: Recursion can be used to solve dynamic programming problems by breaking down the problem into smaller subproblems and solving each subproblem only once.

However, recursion also has some limitations, such as:

1. Stack Overflow: Deep recursion can lead to a stack overflow error, especially for large inputs.

2. Inefficient: Recursion can be less efficient than iteration due to the overhead of function calls and returns.

3. Difficult to Debug: Recursive functions can be challenging to debug due to the complexity of the call stack.

//Java code

public class RecursiveFinancialForecast {

public double calculateFutureValue(double presentValue, double growthRate, int years) {

if (years == 0) {

return presentValue;

} else {

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

}

}

}

Analysis

Discuss the time complexity of your recursive algorithm.

- Time complexity: O(n), where n is the number of years

- The recursive algorithm has a high time complexity due to the repeated calculations involved in the recursive calls.

Explain how to optimize the recursive solution to avoid excessive computation.

Optimization

- Memoization: Store the results of expensive function calls and reuse them when the same inputs occur again.

- Dynamic programming: Break down the problem into smaller subproblems, solve each subproblem only once, and store the solutions to subproblems to avoid redundant computation.

Optimized Implementation

public class OptimizedFinancialForecast {

private double[] memo;

public OptimizedFinancialForecast(int maxYears) {

memo = new double[maxYears + 1];

Arrays.fill(memo, -1);

}

public double calculateFutureValue(double presentValue, double growthRate, int years) {

if (years == 0) {

return presentValue;

} else if (memo[years] != -1) {

return memo[years];

} else {

double futureValue = calculateFutureValue(presentValue \* (1 + growthRate), growthRate, years - 1);

memo[years] = futureValue;

return futureValue;

}

}

}

The optimized implementation uses memoization to store the results of expensive function calls and avoid redundant computation, reducing the time complexity to O(n) with a smaller constant factor.