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

Concept of Recursion:

• Recursion is a technique where a method calls itself to solve a problem. Each recursive call solves a smaller instance of the problem until a base case is reached.

• Base Case: The simplest scenario which can be solved directly without further recursion.

• Recursive Case: The problem is divided into smaller subproblems, and the method calls itself to solve these subproblems.

1. **Setup:**

A method is created to calculate the future value using a recursive approach.

1. **Implementation:**

The implementation code is provided in the wordpad.

1. **Analysis:**

**Discuss the time complexity of your recursive algorithm.**

**Time Complexity:**

* **Time Complexity:** O(n), where n is the number of periods.
  + - * Each recursive call reduces the number of periods by 1 and performs a constant amount of work. Hence, the total time complexity is linear.

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

**Optimization Techniques:**

* + - **Memoization:**

**Concept:** Store the results of previous computations to avoid redundant calculations.

**Application:** In this simple case, memoization is not necessary. However, it becomes useful for more complex recursive functions with overlapping subproblems.

* + - **Tail Recursion Optimization:**

**Concept:** The recursive call is the last operation in the function. Some compilers/interpreters optimize this to avoid increasing the call stack depth.

**Application:** Not directly applicable here because we still perform additional computations after the recursive call. For this specific case, the iterative approach would be more straightforward and efficient.

* **Recursion** offers a clear and intuitive method for calculating future values, but it may not always be the most efficient for large datasets due to potential stack overflow issues.
* For more extensive computations or performance-critical applications, consider iterative methods or advanced optimizations like memoization.