**Exercise 7: Financial Forecasting**

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

**Concept of Recursion**

**Recursion** is a programming technique where a function calls itself in order to solve a problem. This approach can simplify certain problems by breaking them down into smaller, more manageable sub-problems that are easier to solve.

**Key Components of Recursion:**

1. **Base Case**: The condition under which the recursive function stops calling itself. This prevents infinite recursion and provides a solution for the simplest instance of the problem.
2. **Recursive Case**: The part of the function where it calls itself with a modified argument, moving towards the base case.

**How Recursion Simplifies Problems**

Recursion is particularly useful for problems that can be naturally divided into similar sub-problems. Here are a few ways recursion can simplify problem-solving:

1. **Divide and Conquer**: Recursion is a natural fit for divide-and-conquer algorithms, where a problem is divided into smaller sub-problems, each of which is solved recursively. Examples include Quick Sort and Merge Sort.
2. **Tree Traversal**: Recursive algorithms are often used to traverse tree structures, such as binary trees, because each subtree can be processed in the same way as the entire tree.
3. **Dynamic Programming**: Many dynamic programming problems, like the Fibonacci sequence or the knapsack problem, can be solved recursively by breaking them down into overlapping sub-problems.
4. **Backtracking**: Recursion is used in backtracking algorithms to explore all possible solutions to a problem, such as solving mazes, generating permutations, or solving the N-Queens problem.

* **Discuss the time complexity of your recursive algorithm.**
* **Explain how to optimize the recursive solution to avoid excessive computation**

**Time Complexity Analysis:**

The time complexity of the recursive **calculateFutureValue** algorithm is O(years), where **years** is the number of years to calculate the future value. This is because the method recursively calls itself **years** times to calculate the future value.

More specifically, the time complexity can be broken down as follows:

* The recursive call is made **years** times, resulting in a time complexity of O(years).
* Within each recursive call, the calculation **presentValue \* (1 + growthRate)** is performed, which has a constant time complexity of O(1).
* Therefore, the overall time complexity of the algorithm is O(years) \* O(1) = O(years).

**Optimizing the Recursive Solution:**

To optimize the recursive solution and avoid excessive computation, we can use a technique called **memoization**. Memoization involves storing the results of expensive function calls and reusing them when the same inputs occur again.