**Exercise 7: Financial Forecasting**

Recursion is a programming technique where a function calls itself directly or indirectly to solve a problem. It breaks down a problem into smaller subproblems of the same type. Recursion can simplify certain problems by reducing complex iterative solutions into more concise and readable code.

**Key Concepts:**

* **Base Case**: The condition under which the recursion stops and the function returns a value without further recursive calls. It prevents infinite recursion.
* **Recursive Case**: The condition under which the function calls itself with modified arguments to solve a smaller instance of the same problem.

**Time Complexity of Recursive Algorithm**

The time complexity of the recursive algorithm described above is O(n)O(n)O(n), where nnn is the number of periods. This is because the algorithm makes nnn recursive calls, each handling a smaller subproblem until it reaches the base case.

**Optimization of Recursive Solution**

To avoid excessive computation and optimize the recursive solution:

* **Memoization**: Use memoization to store previously computed results of the function in a cache. This avoids redundant calculations for the same inputs and improves performance, especially for problems with overlapping subproblems.
* **Tail Recursion**: Ensure that the recursive calls are tail recursive. In Java, this can be optimized by the compiler to use less stack space, but it's not guaranteed to be optimized in all cases.
* **Iterative Approach**: Consider converting the recursive solution into an iterative one if the recursion depth is expected to be large, to avoid potential stack overflow issues.