**COGNIZANT DIGITAL NURTURE – 3.0**

**JAVA FSE**

**WEEK – 1 EXERCISES**

**DATA STRUCTURES AND ALGORITHMS**

**Exercise 7: Financial Forecasting**

**Step 1: Understand Recursive Algorithms:**

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

A function can call itself to solve a smaller version of the same problem using the programming technique known as recursion. Recursive solutions can be beautiful and straightforward, particularly when applied to issues like tree traversals, factorial computations, and specific sorting algorithms that lend themselves naturally to a recursive definition. Recursion, however, can cause problems with stack overflow and unnecessary computation if it is not handled correctly.

**Step 2: Setup**

***Refer Program Files***

**Step 3: Implementation**

***Refer Program Files***

**Step 4: Analysis**

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

**Time Complexity:**

The time complexity of the provided purely recursive algorithm is O(2^n). This is because each recursive call potentially results in two more calls, leading to an exponential number of calls.

For each call to predictFutureValue(initialValue, growthRate, years), the algorithm recursively calls itself with years - 1 until it reaches the base case where years is 0. This creates a binary tree of calls, where the number of calls grows exponentially with the number of years.

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

To optimize the recursive solution and avoid excessive computation, we can use memoization. Memoization involves:

1. **Storing Results**: Use a map or array to store results of previously computed states.
2. **Checking Stored Results**: Before computing a future value, check if it's already stored. If so, return the stored result.
3. **Computing and Storing**: If the result is not stored, compute it recursively, save it in the map, and return it.

This approach transforms the time complexity from exponential (**O(2^n)**) to linear (**O(n)**) by ensuring each state is computed only once.