**Financial Forecasting Tool with Recursive Prediction – EX 7**

**1. Understanding Recursive Algorithms**

**Recursion**:

- **Concept**: Recursion is a programming technique where a function calls itself in order to solve smaller instances of a problem. This approach is particularly useful for problems that can be divided into subproblems of the same type.
- **Benefits**: Recursion can simplify the implementation of complex algorithms by breaking them down into more manageable subproblems. It is often used in problems involving hierarchical data structures, such as trees, or problems with a naturally recursive structure.

**2. Setup**

**Method for Calculating Future Value Using Recursion**: The method calculates the future value of an investment or any quantity based on an initial value (current), a growth rate (growth), and the number of periods (time).

**3. Implementation**

**Recursive Algorithm to Predict Future Values**:

The recursive method calculate computes the future value by multiplying the previous period's value by (1 + growth). To avoid recalculating the same values repeatedly, the method uses memoization, storing previously computed results in a map (result).

```
import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;


public class FinancialForecastingTool {


    private static Map<Integer, Double> result = new HashMap<>();


    public static double calculate(double current, double growth, int time) {

        if (time == 0) {

            return current;

        }

        if (result.containsKey(time)) {
            return result.get(time);
        }
        double future = calculate(current, growth, time - 1) * (1 + growth);
        result.put(time, future);
        return future;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter current value: ");
        double current = scanner.nextDouble();

        System.out.print("Enter annual growth rate (as a decimal): ");
        double growth = scanner.nextDouble();

        System.out.print("Enter number of periods into the future: ");
        int time = scanner.nextInt();

        double future = calculate(current, growth, time);

        System.out.println("The predicted future value after " + time + " periods is: " + future);
    }
}
```

**4. Analysis**

**Time Complexity**:

- **Basic Recursive Approach**: Without memoization, the time complexity would be O(n) for n time periods, as each call depends on the result of the previous one.

- **Optimized Recursive Approach (with Memoization)**: The time complexity remains O(n), but the use of memoization significantly reduces the number of redundant calculations. The space complexity is O(n) due to the storage of intermediate results.

**Optimization to Avoid Excessive Computation**:

- **Memoization**: By storing the results of already computed periods, the algorithm avoids recalculating the future value for the same period multiple times. This optimization drastically reduces the number of function calls, especially when the same period is queried multiple times or when there are overlapping subproblems.

**Considerations**:

- **Iteration vs. Recursion**: While recursion can provide a clear and concise implementation, iterative approaches can sometimes be more efficient in terms of space, as they avoid the overhead of recursive call stacks. In this case, an iterative approach could achieve the same result with a simple loop, potentially using less memory.
- **Handling Large Time Periods**: When forecasting over a large number of periods, ensure that the data types used can handle the potential size of the future value, especially if the growth rate is substantial.