**Exercise 7: Financial Forecasting**

**Recursive Algorithms:**

Recursion is a method where a function calls itself to solve a problem by breaking it down into smaller instances of the same problem. It's especially useful in mathematical computations where the result depends on previous values.

In this case, we use recursion to calculate compound growth over time.

Recursion helps avoid complex looping structures and makes the logic more readable when the relationship is naturally repetitive or mathematical.

**Setup:**

We are using the standard compound interest formula for forecasting:

**Future Value = Present Value × (1 + rate) ^ years**

Where:

* Present Value is the current amount
* Rate is the annual growth rate (in decimal form)
* Years is the number of future years to forecast

A recursive method is used to repeatedly apply the growth rate until the base case (0 years) is reached.

**Implementation:**

Below is the Java method to calculate future value using recursion:

|  |
| --- |
| **public** **static** **double** **calculateFutureValue**(**double** presentValue, **double** rate, **int** years) {  **if** (years == **0**) {  **return** presentValue;  }  **return** (**1** + rate) \* calculateFutureValue(presentValue, rate, years - **1**);  } |

**Analysis:**

The time complexity of this recursive algorithm is O(n), where n is the number of years. Each recursive call processes one year, reducing the count until it reaches zero.

The space complexity is also O(n) due to the call stack that stores each recursive level.

**Optimization:**

For larger values of years, recursion can become inefficient due to stack usage. Two common optimizations are:

* **Memoization –** Store previously calculated results in an array to avoid redundant computation.

|  |
| --- |
| **public** **static** **double** **calculateFutureValueMemo**(**double** pv, **double** rate, **int** years, **double**[] memo) {  **if** (years == **0**) **return** pv;  **if** (memo[years] != **0**) **return** memo[years];  memo[years] = (**1** + rate) \* calculateFutureValueMemo(pv, rate, years - **1**, memo);  **return** memo[years];  } |

* **Iterative Approach –** Use a simple loop instead of recursion for more efficient memory usage.

|  |
| --- |
| **public** **static** **double** **calculateFutureValueIterative**(**double** pv, **double** rate, **int** years) {  **double** fv = pv;  **for** (**int** i = **0**; i < years; i++) {  fv \*= (**1** + rate);  }  **return** fv;  } |

**Conclusion:**

Recursive algorithms are useful for expressing repetitive mathematical computations like future value forecasting. While the recursive approach is elegant and easy to understand, it may not be the most efficient for large inputs. For better performance, especially in production systems, memoization or iterative methods are recommended.