**Financial Forecasting - Recursive Algorithm Analysis**

**1. Time Complexity Analysis**

The recursive function to forecast future value:

public static double forecastRecursive(double currentValue, double growthRate, int years) {  
 if (years == 0) {  
 return currentValue;  
 }  
 return forecastRecursive(currentValue, growthRate, years - 1) \* (1 + growthRate);  
}

**Time Complexity:**

* The function performs one recursive call per year.
* Each call involves a constant-time multiplication.
* Therefore, the total time complexity is:

**O(n)**, where *n* is the number of years.

**Space Complexity:**

* Each recursive call consumes stack space.
* The maximum depth of recursion is *n*.
* Therefore, the space complexity is:

**O(n)**

**2. Optimization Strategy**

To avoid excessive computation and potential stack overflow, the recursive approach can be replaced with an iterative version:

public static double forecastIterative(double currentValue, double growthRate, int years) {  
 for (int i = 0; i < years; i++) {  
 currentValue \*= (1 + growthRate);  
 }  
 return currentValue;  
}

**Benefits of Iterative Version:**

* Avoids the overhead of recursive calls.
* Consumes constant memory (O(1)).
* Faster execution, especially for large values of *n*.

**3. Summary**

| Approach | Time Complexity | Space Complexity | Use Case |
| --- | --- | --- | --- |
| Recursive | O(n) | O(n) | Simple cases, small *n* |
| Iterative | O(n) | O(1) | Large *n*, better performance |

To further optimize in scenarios with varying growth rates or compound structures, dynamic programming or memoization techniques may be considered.