Financial Forecasting Explaination

Example Scenario

We will forecast the future value of an investment using:

📍 Future Value (FV) = Current Value (CV) × (1 + Growth Rate)ⁿ

where:

CV = current investment value

Growth Rate = expected growth rate per period (as a decimal, e.g., 0.05 for 5%)

n = number of periods into the future

📌 Time Complexity of the Recursive Algorithm

The recursive method:

**calculateFutureValue(currentValue, growthRate, periods)**

has this structure:

If periods == 0: return current value — O(1)

Else: one recursive call with periods - 1

Since at each step, we do one recursive call and decrease periods by 1, the recursion depth will be exactly periods.

👉 Time Complexity = O(periods) (linear in the number of periods)

👉 Space Complexity = O(periods) because each recursive call adds a frame on the call stack.

📌 How to Optimize the Recursive Solution

Recursion is fine for small periods, but:

For large periods (say, 10,000 or more), recursion risks stack overflow.

Also, this problem does not require storing intermediate states (no overlapping subproblems), so recursion isn’t ideal.

Optimizations:

✅ 1. Replace recursion with iteration

Use a simple for loop — more efficient and avoids stack issues:

**public static double calculateFutureValueIterative(double currentValue, double growthRate, int periods) {**

**double futureValue = currentValue;**

**for (int i = 0; i < periods; i++) {**

**futureValue \*= (1 + growthRate);**

**}**

**return futureValue;**

**}**

👉 Now, Time Complexity = O(periods),

👉 Space Complexity = O(1) — no stack usage.

✅ 2. Use Mathematical Formula

Since this is a compound growth problem, the entire recursive process can be replaced with this formula:

**FV=CV×(1+r)^n**

In Java:

**public static double calculateFutureValueFormula(double currentValue, double growthRate, int periods) {**

**return currentValue \* Math.pow(1 + growthRate, periods);**

**}**

👉 Time Complexity = O(1)

👉 Space Complexity = O(1)

This is the fastest and most optimized way to compute future value in this case.

Output --🡪

