Financial Forecasting – Analysis

Why Recursion?

Ponder compounding as a chain reaction: you use one period's growth on your initial amount, then take that new amount and use the next period's growth, and repeat. Recursion captures that process exactly by allowing each function call to work on one period and then pass on the revised value to the next call.

How We Did It

We defined a method futureValue(principal, growthRates, index).

If index is equal to the number of elements in the growthRates array, we know we're finished and simply return the current amount.

Otherwise, we compute the new principal as principal \* (1 + growthRates[index]), and recursively call the same method with the new principal and index + 1.

How Fast Is It?

Time: It iterates over each growth rate only once, so it takes O(n) time, with n being the number of periods.

Space: Since each period causes a new function call, the call stack will go as deep as n, so it takes O(n) space.

Making It Even Better

For quite long lists of rates, recursion can threaten a stack overflow and adds additional overhead per call. You may turn to a straightforward loop that performs the same multiplications in succession—this still has O(n) time but only O(1) additional space.

Although our recursive form is technically tail-recursive (the last step is the recursive call), Java won’t eliminate that call for us, so the loop is the safer bet in practice.

This approach is clean and closely matches the idea of compounding, but if you’re working with a lot of periods, you’ll probably prefer the iterative version for reliability.