**Scenario:**

In this task, I worked on developing a financial forecasting tool that uses a recursive algorithm to predict future values based on past data trends or growth rates. The goal was to understand recursion and apply it to a real-world problem.

1. **What I Understood – Recursive Algorithms**

While solving this, I understood that recursion is when a function calls itself to break down a problem into smaller sub-problems until a base condition is met.

Key Points I Learned:

It’s like solving a big problem by solving smaller versions of the same problem.

Every recursive function must have:

* A base case (when to stop)
* A recursive case (when to call itself again)

It is useful in problems where the output depends on previous values, like financial forecasting.

Example: To predict value after n years, we can use the value from n-1 year and apply the growth rate recursively.

2. **Setup – Method for Forecasting**

We define a method to forecast future values based on:

Initial investment or value

Annual growth rate (as a decimal)

Number of years (n)

3. **What I Understood – Analysis**

Time Complexity:

The recursive function runs once for each year, so the time complexity is:

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

It’s not very heavy for small n, but if the recursion depends on multiple calls per step (like in Fibonacci), it can become expensive.

**How to Optimize:**

In this case, recursion is linear and doesn't repeat calculations, so it's acceptable.

For more complex recursion, we can use:

Memoization (store previously calculated values)

Dynamic Programming

Or just convert to iterative solution (using loops) for better performance.

In real-world financial tools where performance matters, iterative methods are preferred for speed and memory control.

**Final Understanding**

* I learned how to use recursion to predict future values from past trends.
* I understood how recursive calls can mirror real-world repetitive patterns.
* I also realized that recursion is powerful but must be used carefully to avoid performance issues.