**Step 1: Understand Recursive Algorithms**

**1. Concept of Recursion:**

* **Definition:** Recursion is a method of solving a problem where a function calls itself as a subroutine. This allows the function to be repeated several times, as it can call itself during its execution.
* **Advantages:** Simplifies the code for problems that can be broken down into smaller, similar subproblems.
* **Disadvantages:** Can lead to excessive computation and stack overflow if not optimized properly.

**Step 4: Analysis**

**1. Time Complexity:**

* **Time Complexity:** O(n)
  + Each recursive call represents one year of prediction, so the algorithm runs in linear time relative to the number of years.
* **Space Complexity:** O(n)
  + Each recursive call adds a frame to the call stack, leading to linear space complexity.

**2. Optimization:**

* **Avoiding Excessive Computation:** The current implementation is already optimal in terms of computation since each year's value is calculated based on the previous year's value, making it a straightforward linear recursion.
* **Tail Recursion Optimization:** In some languages, tail recursion can be optimized to prevent stack overflow. Java does not support tail call optimization, but ensuring minimal stack usage is helpful.

**Explanation of Recursive Approach**

* **Base Case:** The base case is when the number of years (years) is 0. At this point, the function returns the current value, as no more future values need to be calculated.
* **Recursive Case:** For each recursive step, the next year's value is calculated by multiplying the current value by (1 + ‘growthRate’). The function then calls itself with the updated value and decrements the number of years by 1.