**Recursive Algorithms**

**Recursion :** Recursion is a technique where a function calls itself to solve a problem. It breaks down a problem into smaller instances of the same problem.

* Simplifies code for problems like tree traversal, factorial calculation, Fibonacci series, etc.
* Can naturally represent repetitive calculations like compound growth.

**Analysis**

**Time Complexity:**

* The recursive function has time complexity O(n) where n is the number of years.
* This is because the function is called once for each year.

**Optimization:**

To avoid excessive computation:

1. Memoization is not very useful here since each recursion uses a different presentValue.
2. **Use Iteration instead if performance is critical**:

public static double predictFutureValueIterative(double presentValue, double growthRate, int years) {

for (int i = 0; i < years; i++) {

presentValue \*= (1 + growthRate);

}

return presentValue;

}

1. **Mathematical Power Function** (most efficient):

public static double predictFutureValueMath(double presentValue, double growthRate, int years) {

return presentValue \* Math.pow(1 + growthRate, years); }

**Summary**

| **Method** | **Time Complexity** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| Recursive | O(n) | Simple, elegant | Risk of stack overflow for large n |
| Iterative | O(n) | Efficient, safe | Slightly more code |
| Math.pow() | O(1) | Fastest | Might lose some clarity in logic |