1. **Understand Recursive Algorithms**

* Recursion is a programming technique where a method calls itself to solve a smaller version of the problem.
* It's useful for repetitive calculations like forecasting, factorial, Fibonacci, etc.

int factorial(int n) {

if (n == 0) return 1;

return n \* factorial(n - 1);

}

1. **Setup the Future Value Forecasting Problem**

**Future Value** after n years = presentValue \* (1 + growthRate)^n

FV(n) = FV(n-1) \* (1 + growthRate)

Base case: n == 0 → return presentValue

1. **Implementation**

public class FinancialForecasting {

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

if (years == 0) return presentValue;

return forecastValue(presentValue, growthRate, years - 1) \* (1 + growthRate);

}

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

double futureValue = presentValue;

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

futureValue \*= (1 + growthRate);

}

return futureValue;

}

public static void main(String[] args) {

System.out.println("== Financial Forecasting Using Recursion ==");

double presentValue = 10000.0;

double growthRate = 0.08;

int years = 5;

double futureValueRecursive = forecastValue(presentValue, growthRate, years);

System.out.println("Future value (recursive): ₹" + futureValueRecursive);

double futureValueIterative = forecastValueIterative(presentValue, growthRate, years);

System.out.println("Future value (iterative): ₹" + futureValueIterative);

System.out.println("\nTime Complexity:");

System.out.println("- Recursive: O(n), but not optimized for large n due to call stack.");

System.out.println("- Iterative: O(n), more efficient and avoids stack overhead.");

}

}

1. **Analysis & Optimization**

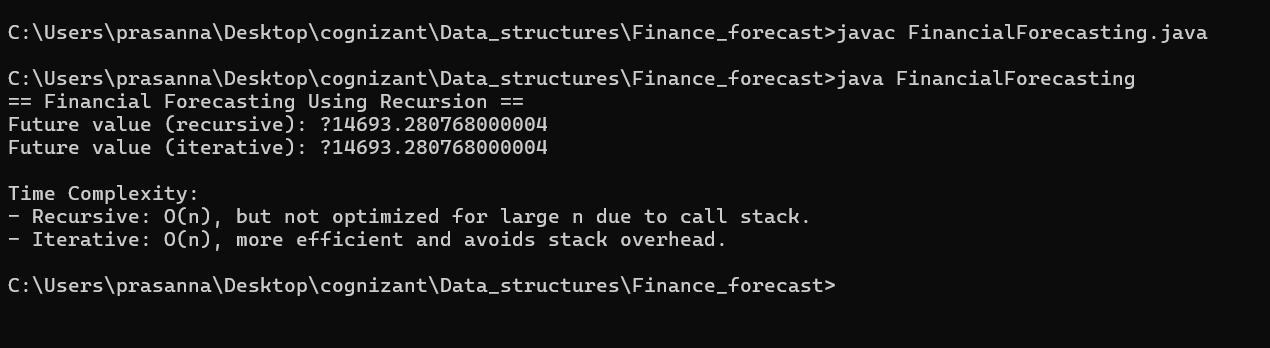
**Time Complexity:**

* The **recursive algorithm** has **O(n)** time complexity since it calls itself n times.

**Optimization Tips:**

* Use **memoization** or convert it to **iteration** if n is large.
* Recursion is clean but can be slower for deep calls due to stack overhead.

**Output**

****