**1.Explain Recursion:**

**Recursion** is when a method **calls itself** to solve a smaller version of the problem until reaching a **base case** that stops further calls. It can simplify problems that have a **natural recursive structure**, like:

* Factorials
* Fibonacci numbers
* Financial forecasting with compound growth rates

## 2. Setup: Forecasting Future Values Recursively

* Example:
* currentValue (e.g. starting value),
* growthRate (e.g. 5% per period),
* years (number of periods into the future).

Base Case: If years == 0, the future value is simply currentValue.  
 Recursive Step: forecast(years) = forecast(years-1) \* (1 + growthRate)

**3.**

**ForecastUtil.java**

public class ForecastUtil {  
 public static double forecastFutureValue(double currentValue, double growthRate, int years) {  
 if (years == 0) {  
 return currentValue; // Base case  
 }  
 return *forecastFutureValue*(currentValue, growthRate, years - 1) \* (1 + growthRate);  
 }  
  
 public static void main(String[] args) {  
 double currentValue = 1000.0;  
 double growthRate = 0.05; // 5% per period  
 int years = 10;  
 double futureValue = *forecastFutureValue*(currentValue, growthRate, years);  
 System.*out*.println("Future Value after " + years + " years: " + futureValue);  
 }  
}

**Output:**

yaml

CopyEdit

Future Value after 10 years: 1628.89

4.

1. Tail Recursion or Iteration:  
   Rewrite the recursion as a simple loop to avoid stack overhead.
2. Memoization?  
   Here, memoization doesn’t help much because we only need the next value (years-1). But for problems like the Fibonacci sequence, memoization is great.
3. Formula (Closed-form):  
   We could also use the compound interest formula directly for years:

futureValue=currentValue×(1+growthRate)years\text{futureValue} = currentValue \times (1 + growthRate)^{years}futureValue=currentValue×(1+growthRate)years

This is O(1) and eliminates recursion.