Step 1: Understand Recursive Algorithms

What is Recursion?

Recursion is a technique where a function calls **itself** to solve a smaller subproblem. It usually involves:

- **Base case**: Terminates recursion.
- **Recursive case**: Breaks down the problem into smaller calls.

Why use recursion?

- Simplifies problems like tree traversal, Fibonacci, and repetitive calculations like compound interest.

Step 2: Setup – Recursive Forecasting Method

We'll calculate future value based on:

- `initialAmount`
- `growthRate` (percentage)
- `years` (number of periods)

Formula:

```
futureValue = initialAmount * (1 + growthRate)^years
```

Step 3: Implementation in Java

```
public class FinancialForecaster {

    //Optimized method -> O(n)
   public static double OptimizedforecastIterative(double amount, double rate, 
int years){
    for (int i = 0; i < years; i++) {
        amount *= (1 + rate);
    }
    return amount;
   }

    // Recursive method to calculate future value -> O(n)
    public static double forecast(double amount, double rate, int years) {
        // Base case: no growth
        if (years == 0) {
            return amount;
        }
        // Recursive case: apply rate for one year, then recurse
        return forecast(amount * (1 + rate), rate, years - 1);
    }

    public static void main(String[] args) {
        double initialAmount = 10000; // ₹10,000
        double annualGrowthRate = 0.09; // 9%
        int years = 5;

        double futureValue1 = forecast(initialAmount, annualGrowthRate, years);
        System.out.printf("Recursive Forecasted value after %d years: ₹%.2f\n",
years, futureValue1);
double futureValue2 = OptimizedforecastIterative(initialAmount, annualGrowthRate,
years);
        System.out.printf("Iterative Forecasted value after %d years: ₹%.2f\n",
years, futureValue2);

    }
}
```

Step 4: Analysis

Time Complexity For Recursive

- **Recursive Calls**: One call per year → O(n)

- **Space Complexity**: Due to call stack → O(n)

Time Complexity For Iterative

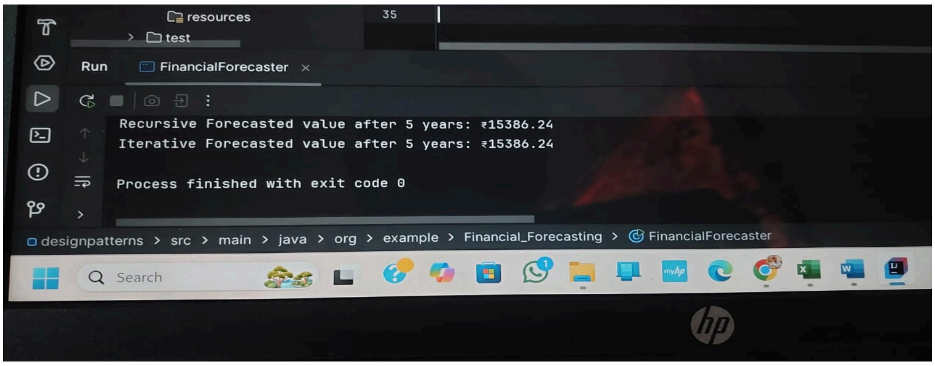
- **Recursive Calls**: One call per year → O(n)

- **Space Complexity**: Due to call stack → O(1)

Expected Output:

```
Recursive Forecasted value after 5 years: ₹15386.24
Iterative Forecasted value after 5 years: ₹15386.24
```

Final Output:


resources
test
35
Run
FinancialForecaster
designpatterns > src > main > java > org > example > Financial_Forecasting > FinancialForecaster
Search

```
Recursive Forecasted value after 5 years: ₹15386.24
Iterative Forecasted value after 5 years: ₹15386.24

Process finished with exit code 0
```

how to optimize the recursive solution to avoid excessive computation?

Option 1: Memoization

If the function is called with repeated inputs, store results in a map to avoid recalculating.

Option 2: Convert to Iterative

An iterative version avoids recursion and is more memory-efficient:

```java
public static double forecastIterative(double amount, double rate, int years) {
    for (int i = 0; i < years; i++) {
        amount *= (1 + rate);
    }
    return amount;
}
```