**Exercise 7: Financial Forecasting**

1. Explain the concept of recursion and how it can simplify certain problems.

**Concept of Recursion:**

* **Definition:** Recursion is a technique where a function calls itself to solve a problem. The problem is broken down into smaller instances of the same problem.
* **Base Case:** The simplest instance of the problem that can be solved directly without further recursion.
* **Recursive Case:** The part of the function where the recursion happens, solving a smaller instance of the problem and combining the results to form a solution to the original problem.

**Advantages of Recursion:**

* **Simplifies Complex Problems:** Certain problems are naturally recursive, such as calculating factorials, Fibonacci numbers, or navigating tree structures.
* **Code Clarity:** Recursive solutions can be more intuitive and easier to understand for problems that involve repeated sub-problems.

1. Create a method to calculate the future value using a recursive approach.

public class Main {  
  
 public static double calculateFutureValue(double currentAmount,double growthRate,int years){  
 if(years == 0)  
 return currentAmount;  
 System.*out*.println(currentAmount);  
 return *calculateFutureValue*(currentAmount \* (1 + growthRate),growthRate,years - 1);  
 }  
  
 public static void main(String[] args) {  
 double amount = 1000D;  
 double growthRate = 0.5;  
 int years = 10;  
  
 System.*out*.println(*calculateFutureValue*(amount,growthRate,years));  
 }  
}

1. Time Complexity

* **Recursive Time Complexity:** The recursive method provided has a time complexity of O(n)O(n)O(n), where nnn is the number of years. This is because each recursive call processes one year and the function is called recursively nnn times.

**Optimization and Avoiding Excessive Computation:**

* **Memoization:** To avoid recalculating the same values multiple times, we can use memoization. This technique involves storing the results of previous computations and reusing them, which can significantly reduce the number of recursive calls.

1. Optimized Approach with Memoization

import java.util.HashMap;

import java.util.Map;

public class FinancialForecasting {

private static Map<Integer, Double> memo = new HashMap<>();

// Method to calculate future value using memoization

public static double calculateFutureValue(double currentValue, double growthRate, int years) {

if (years == 0) {

return currentValue;

}

if (memo.containsKey(years)) {

return memo.get(years);

}

double futureValue = calculateFutureValue(currentValue \* (1 + growthRate), growthRate, years - 1);

memo.put(years, futureValue);

return futureValue;

}

public static void main(String[] args) {

double currentValue = 1000; // Example current value

double growthRate = 0.05; // Example growth rate (5%)

int years = 10; // Number of years to forecast

double futureValue = calculateFutureValue(currentValue, growthRate, years);

System.out.printf("The future value after %d years is: %.2f\n", years, futureValue);

}

}