**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Steps:**

1. **Understand Recursive Algorithms:**
   * Explain the concept of recursion and how it can simplify certain problems.
2. **Setup:**
   * Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
   * Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
   * Discuss the time complexity of your recursive algorithm.
   * Explain how to optimize the recursive solution to avoid excessive computation.

**My Answers:**

1. **Understand Asymptotic Notation:**

**Time Complexity of the Recursive Algorithm:**

Each call depends on the result of the previous year → years - 1

So it calls itself n times (where n = number of years):

1. **Optimize Recursion:**

Memoization (HashMap ) 🡪 create cache & store the value

**TIME COMPLEXITY ANALYSIS:**

|  |  |  |  |
| --- | --- | --- | --- |
| Cases | Best case | Average Case | Worst case |
| Without memorization | O(n) | O(n) | O(n) |
| With memorization | O(1) | O(n) | O(n) |

**SPACE COMPLEXITY ANALYSIS:**

|  |  |  |  |
| --- | --- | --- | --- |
| Cases | Best case | Average Case | Worst case |
| Without memorization | O(n) | O(n) | O(n) |
| With memorization | O(n) | O(n) | O(n) |

**My Code:**

import java.util.HashMap;  
import java.util.Map;  
import java.util.Scanner;  
  
public class Main {  
  
 static Map<Integer, Double> memo = new HashMap<>();  
  
 static double getGrowthFactor(int years, double rate) {  
 if (years == 0) return 1.0;  
  
 if (memo.containsKey(years)) return memo.get(years);  
  
 double result = getGrowthFactor(years - 1, rate) \* (1 + rate);  
 memo.put(years, result);  
 return result;  
 }  
  
 static double forecastFutureValue(double principal, double rate, int years) {  
 double factor = getGrowthFactor(years, rate);  
 return principal \* factor;  
 }  
  
 public static void main(String[] args) {  
 Scanner input = new Scanner(System.in);  
 System.out.print("Enter principal amount: ");  
 double principal = input.nextDouble();//1000  
 System.out.print("Enter Interest rate : ");  
 int interestRate = input.nextInt();  
 double rate = ( (double) interestRate /100) ;  
 System.out.print("Enter years : ");  
 int years = input.nextInt();  
  
 double futureValue = forecastFutureValue(principal, rate, years);  
  
 System.out.println("----- Financial Forecast Report -----");  
 System.out.printf("Principal : Rs: %.2f%n", principal);  
 System.out.printf("Interest Rate : %.2f%% per year%n", rate \* 100);  
 System.out.printf("Forecast Period : %d years%n", years);  
 System.out.printf("Growth Factor : %.6f%n", memo.get(years));  
 System.out.printf("Future Value : Rs: %.2f%n", futureValue);  
  
 System.out.println("\nMemoization Cache (Year → Growth Factor):");  
 for (Map.Entry<Integer, Double> entry : memo.entrySet()) {  
 System.out.printf("Year %2d → %.6f%n", entry.getKey(), entry.getValue());  
 }  
 }  
}  
/\* op: for this code :  
Enter principal amount: 1000  
Enter Interest rate : 7  
Enter years : 10  
----- Financial Forecast Report -----  
Principal : Rs: 1000.00  
Interest Rate : 7.00% per year  
Forecast Period : 10 years  
Growth Factor : 1.967151  
Future Value : Rs: 1967.15  
  
Memoization Cache (Year → Growth Factor):  
Year 1 → 1.070000  
Year 2 → 1.144900  
Year 3 → 1.225043  
Year 4 → 1.310796  
Year 5 → 1.402552  
Year 6 → 1.500730  
Year 7 → 1.605781  
Year 8 → 1.718186  
Year 9 → 1.838459  
Year 10 → 1.967151  
  
Process finished with exit code 0  
  
 \*/

