WEEK 1

Mandatory Hands-On

**Data Structures and Algorithms:**

**Ex 7: Financial Forecasting**

1. Understanding Recursive Algorithms:

Recursion is a programming technique where a function calls itself to solve smaller instances of the same problem, until it reaches a base case that stops the recursion.

Its components are,

* + - Base Case – the condition where the function stops calling itself.
    - Recursive Case – the part where the function calls itself with a smaller/simpler input.

Recursion is useful for,

* Problems that have repetitive patterns.
* Problems that can be broken down into smaller sub-problems.
* Simplifying logic where iteration becomes complex.

1. Setup:

In order to Forecast the future financial values using recursion we can assume that,

* The initial value is taken. (e.g., investment amount or revenue).
* The growth rate in which the value increases per time. (e.g., 5% annual growth).
* Time period for which the growth has to be found. (e.g., after 5 years).

Definition of variables for Recursive formula:

* futureValue (n) = Future value at year n.
* rateOfGrowth = growth rate (e.g., 0.05 for 5%).
* futureValue (0) = Initial value.
* numberOfYears = The time period.

Using these components the recursive formula can be defined as,

**futureValue(numberOfYears) = futureValue(numberOfYears - 1) \* (1 + rate).**

1. Implementation:

public class FinancialForecast {

public static double futureValue(double initialValue, double rate, int years) {

if (years == 0) {

return initialValue;

}

return futureValue(initialValue, rate, years - 1) \* (1 + rate);

}

public static void main(String[] args) {

double initialValue = 10000.0;

double rate = 0.05;

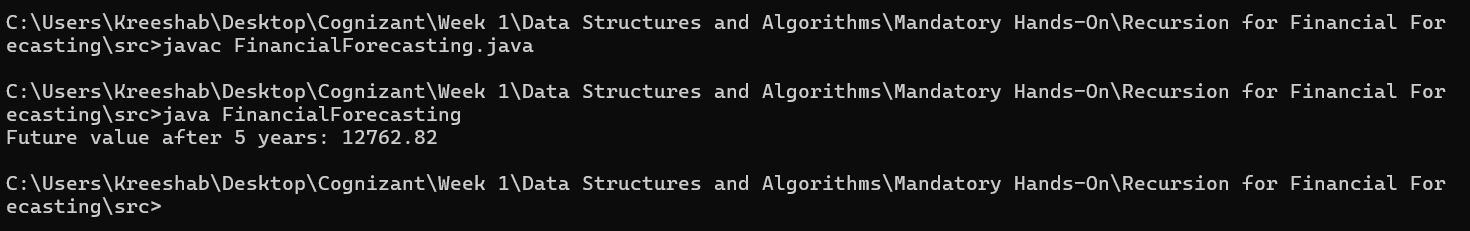
int years = 5;

double result = futureValue(initialValue, rate, years);

System.out.printf("Future value after %d years: %.2f%n", years, result);

}

}

OUTPUT:

1. Analysis of Recursive Algorithms:
2. Time complexity Analysis:

* This function makes one recursive call per year.
* So for numberOfYears years, the function calls itself numberOfYears times before reaching the base case.
* So it can be said that the Time complexity of the considered recursive algorithm is O(n) where n is numberOfYears.

1. Optimization of the algorithm:

Even though the current recursive method is efficient enough for small numberOfYears, recursion can become inefficient or risk stack overflow for large numberOfYears.

* Optimization using Iterative method:

public static double futureValueIterative(double initialValue, double rate, int years) {

double value = initialValue;

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

value \*= (1 + rate);

}

return value;

}

This case can be considered as an optimization in memory usage.

* Optimization using Exponentiation:

public static double futureValueIterative(double initialValue, double rate, int years) {

double value = initialValue;

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

value \*= (1 + rate);

}

return value;

}

This case theoretically gives a time complexity of O(log n) but when practically used gives the time complexity of O(1), which is the best time complexity that can be achieved in a recursive algorithm.