WEEK1

EXERCISE 7:FINACIAL FORESCASTING

Forecastutil.java

package com.finance.forecast;

public class ForecastUtil {

public static double futureValue(double presentValue, double growthRate, int years) {

if (years == 0) {

return presentValue;

}

return futureValue(presentValue \* (1 + growthRate), growthRate, years - 1);

}

public static double futureValueMemo(double presentValue, double growthRate, int years, Double[] memo) {

if (years == 0) return presentValue;

if (memo[years] != null) return memo[years];

memo[years] = futureValueMemo(presentValue \* (1 + growthRate), growthRate, years - 1, memo);

return memo[years];

}

}

Main.java

package com.finance.forecast;

public class Main {

public static void main(String[] args) {

double presentValue = 1000;

double growthRate = 0.10;

int years = 5;

double forecast = ForecastUtil.futureValue(presentValue, growthRate, years);

System.out.println("Predicted Future Value (Recursive): " + forecast);

Double[] memo = new Double[years + 1];

double forecastMemo = ForecastUtil.futureValueMemo(presentValue, growthRate, years, memo);

System.out.println("Optimized Future Value (Memoized): " + forecastMemo);

}

}

OUTPUT:

Predicted Future Value (Recursive): 1610.5100000000002

Optimized Future Value (Memoized): 1610.5100000000002

Discuss the Time Complexity of the Recursive Algorithm

The recursive algorithm we wrote to predict future value works by calling itself one time for each year until it reaches the base case (years == 0).

Time complexity:

If we want the future value after n years, the recursion will do n calls — one for each year.

Time complexity = O(n)

Explanation:

• Each recursive call reduces years by 1.

• Total calls = number of years (n), so the running time grows linearly with n.

That’s why we say O(n).

Explain How to Optimize the Recursive Solution to Avoid Excessive Computation

Key Issues with Recursion:

• Deep recursion can cause overhead.

• For very large n, the program may use too much stack space.

Optimizations:

Use the Direct Formula:

Since we know the formula for compound growth:

future\_value = present\_value \* (1 + growth\_rate)^years

This computes the result immediately without recursion — only one calculation.

Time complexity = O(1).

iterative Approach (as an alternative):

Replace recursion with a simple for loop:

public static double futureValue(double presentValue, double growthRate, int years) {

double future = presentValue;

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

future \*= (1 + growthRate); // accumulate compound growth

}

return future;

}

• Time Complexity = O(n), but no recursion overhead and safer for large n.

• Uses constant stack space