**Exercise-7 : Financial Forecasting**

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

Recursion is a process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. This technique provides a way to break complicated problems down into simple problems which are easier to solve. Each recursive call creates a **new copy of the method** in memory. When the base condition is met, these calls start returning one by one. It consists of two parts the base case condition which works as a stopping condition and a recursive case this is a part where the method calls itself.

**formulas used to solve :**

1. **Growth Rate Formula** :

growthRate = (currentValue - previousValue) / previousValue;

1. Average Growth Rate Formula:

averageGrowthRate = totalGrowthRateSum / (numberOfValues - 1);

1. Recursive Future Value Formula

futureValue(n) = futureValue(n - 1) \* (1 + averageGrowthRate);

**Setup**

**Creating a method to calculate the future value using a recursive approach.**

public class ForecastSetup {

// this is to create the average growth rate

public static double getAvgRate(List<Double> values) {

double sum = 0.0; for (int i = 1; i < values.size(); i++) {

double rate = (values.get(i) - values.get(i - 1)) / values.get(i - 1);

sum += rate;

}

return sum / (values.size() - 1);

}

public static void main(String args[]) {

//defining the scanner values to take the values from the users only or to take the random test cases

Scanner sc = new Scanner([System.in](http://System.in));

// Input number of past values

int count = sc.nextInt();

List<Double> data = new ArrayList<>();

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

System.out.print("Value " + (i + 1) + ": ");

data.add(sc.nextDouble());

}

//entering the values to predict the past values

int years = sc.nextInt(); double rate = getAvgRate(data);

double last = data.get(data.size() - 1);

//this is the average growth rate specified and printed

System.out.printf("Average Growth Rate: %.2f%%\\n", rate \* 100);

// Predict future values

Forecast.predictValues(last, rate, years);

sc.close();

}

}

**Implementation:**

**Implement a recursive algorithm to predict future values based on past growth rates.**

public class ForecastImplementation {

public static double predict(double base, double rate, int year) {

if (year == 0) return base;

return predict(base \* (1 + rate), rate, year - 1);

}

// we have created a loop to display predictions

public static void predictFutureValues(double base, double rate, int totalYears) {

for (int i = 1; i <= totalYears; i++) {

double result = predict(base, rate, i);

System.out.printf("Predicted value : %d, %.2f\\n", i, result);

}

}

}

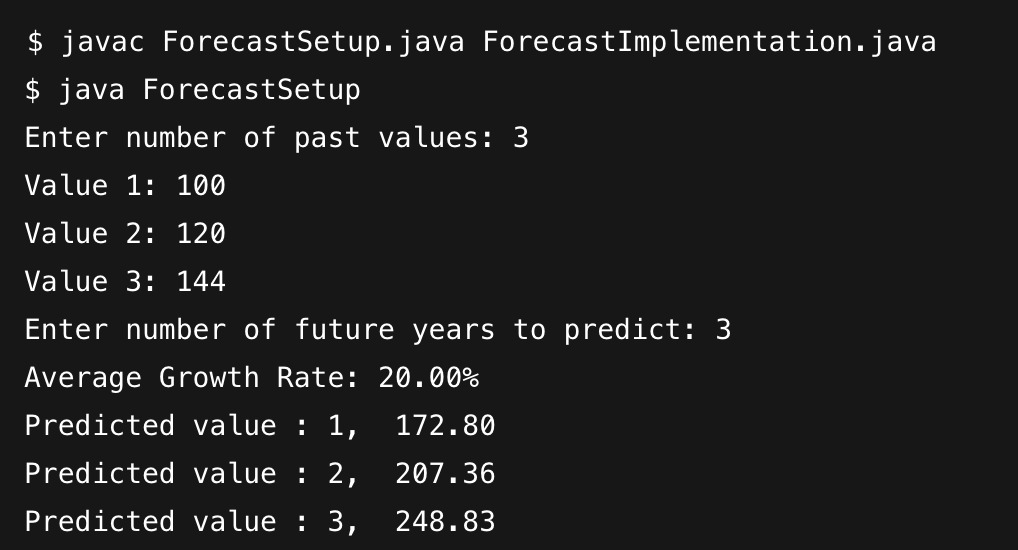
**Discuss the time complexity of your recursive algorithm.**

The time complexity of the recursive algorithm is **O(n)**, where **n** is the number of future years to predict. This is because the function calls itself once for each year, decreasing the year by 1 in every call until it reaches 0. So, it makes one recursive call per year, resulting in a total of **n** recursive calls.

**Explain how to optimize the recursive solution to avoid excessive computation.**

To optimize the recursive solution, we can replace the recursion with a simple loop. In our case, each future value depends only on the previous one, and there are no repeated or overlapping calculations. Using recursion here means that for each year, a new function call is added to the stack, which can lead to stack overflow if the number of years is very large. By switching to an iterative approach using a for loop, we can compute the same result without using extra memory on the call stack. This makes the program more efficient and safer for large inputs.

**OUTPUTS –**

****

