Step 1: Understand Recursive Algorithms

Recursion:

Recursion is a programming technique where a function calls itself repeatedly until it reaches a base case that stops the recursion. Recursion can simplify certain problems by breaking them down into smaller sub-problems that are easier to solve.

Step 2: Setup

Create a method to calculate the future value using a recursive approach:

public class FinancialForecasting {

public double calculateFutureValue(double presentValue, double growthRate, int years) {

// Base case: if years is 0, return the present value

if (years == 0) {

return presentValue;

} else {

// Recursive case: calculate the future value for the next year

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

}

}

}

Step 3: Implementation

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

public class FinancialForecasting {

public double[] predictFutureValues(double presentValue, double growthRate, int years) {

double[] futureValues = new double[years + 1];

futureValues[0] = presentValue;

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

futureValues[i] = calculateFutureValue(presentValue, growthRate, i);

}

return futureValues;

}

private double calculateFutureValue(double presentValue, double growthRate, int years) {

// Base case: if years is 0, return the present value

if (years == 0) {

return presentValue;

} else {

// Recursive case: calculate the future value for the next year

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

}

}

}

Step 4: Analysis

Time Complexity Analysis:

The time complexity of the recursive algorithm is O(years), where years is the number of years to predict. This is because the algorithm makes recursive calls for each year, and each call takes constant time.

Optimization:

To optimize the recursive solution, we can use memoization to avoid excessive computation. Memoization is a technique that stores the results of expensive function calls and returns the cached result when the same inputs occur again.

Here's an optimized version of the algorithm using memoization:

public class FinancialForecasting {

private double[] memo;

public double[] predictFutureValues(double presentValue, double growthRate, int years) {

memo = new double[years + 1];

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

memo[i] = -1; // Initialize memo array with -1

}

double[] futureValues = new double[years + 1];

futureValues[0] = presentValue;

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

futureValues[i] = calculateFutureValue(presentValue, growthRate, i);

}

return futureValues;

}

private double calculateFutureValue(double presentValue, double growthRate, int years) {

// Check if the result is already memoized

if (memo[years]!= -1) {

return memo[years];

}

// Base case: if years is 0, return the present value

if (years == 0) {

return presentValue;

} else {

// Recursive case: calculate the future value for the next year

double result = calculateFutureValue(presentValue \* (1 + growthRate), growthRate, years - 1);

memo[years] = result; // Memoize the result

return result;

}

}

}