**Exercise 7 : Financial Forecasting**

**1. Recursive Algorithms**

Recursion is a programming technique where a method calls itself to solve smaller instances of the same problem. It’s a way of breaking down complex problems into simpler, more manageable sub-problems.

Key characteristics of a recursive function include:

• Base Case: Stops recursion when a condition is met.

• Recursive Step: Calls itself with smaller inputs, moving towards the base case.

Example : Calculating the factorial of a number n

- n! = n times (n-1)!

- Base case: ( 0! = 1 or 1! = 1)

public class FactorialExample {

// Recursive method to calculate factorial

public static int factorial(int n) {

// Base case: if n is 0 or 1, return 1

if (n == 0 || n == 1) {

return 1;

}

// Recursive case

return n \* factorial(n - 1);

}

public static void main(String[] args) {

// Test the function

int result = factorial(5); // Calculate factorial of 5

System.out.println("Factorial of 5 is: " + result); // Output: 120

}

}

**2. How recursion simplifies the problems**

Recursion simplifies problems by breaking them down into smaller, more manageable sub-problems of the same type. It allows for an elegant and intuitive way to solve problems that have repetitive or self-similar structures, such as trees, sequences, or mathematical computations.

1. Divide-and-Conquer Approach

* Recursion splits a complex problem into smaller instances of the same problem.
* Each smaller problem is solved independently and combined to get the final solution.
* Example: Calculating a factorial (n!=n×(n−1)!) involves solving smaller factorials recursively.

2. Simplified Logic

* The recursive function only needs to define:

Base Case: The simplest instance of the problem that can be solved directly.

Recursive Case: The relation that breaks the problem into smaller sub-problems.

This eliminates the need for explicit loops or additional logic to handle smaller cases manually.

3. Intuitive for Self-Similar Problems

* Problems like tree traversal, Fibonacci series, or solving a maze have a naturally recursive structure.
* Example: Traversing a binary tree is intuitive with recursion since each node's subtrees can be treated as smaller trees.

4. Code Readability

* Recursive solutions are often more concise and easier to understand compared to iterative solutions, especially for problems like generating permutations or solving puzzles (e.g., Tower of Hanoi).

**3. Analysis**

Time Complexity of the recursive algorithm used in the financial forecasting:

1. Recursive Calls:

* For each call to forecast, the function makes exactly one recursive call with n−1.
* The recursion continues until n=0, resulting in n recursive calls.

1. Work Done Per Call:

* Each call performs constant work O(1) to compute the multiplication and addition.

Time Complexity: O(n)

**Optimization**

To avoid excessive computation, consider using an iterative approach. Here is the optimized version:

public static double calculateFutureValueIterative(double currentValue, double growthRate, int years) {  
 for (int i = 0; i < years; i++) {  
 currentValue \*= (1 + growthRate);  
 }  
 return currentValue;  
}

• Recursive Approach: Elegant and easy to understand for small inputs.

• Iterative Approach: More efficient for larger inputs due to no stack usage.