**FINANCIAL FORECASTING**

**Concept of Recursion**

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

**Components of Recursion:**

1. Base Case:

- The condition under which the recursion stops. It prevents infinite loops and provides a direct solution to the simplest form of the problem.

- Example: In the factorial calculation, the base case is when the input is 0 or 1, for which the factorial is 1.

2. Recursive Case:

- The part of the function where the problem is broken down into smaller instances of itself.

- Example: In factorial calculation, the recursive case is `factorial(n) = n \* factorial(n-1)`.

How Recursion Simplifies Problems

1. Divide and Conquer:

- Recursion breaks down a problem into smaller sub-problems, making it easier to manage and solve. Each recursive call works on a simpler version of the original problem.

2. Elegant Solutions:

- Recursive solutions can be more concise and easier to read compared to iterative solutions, especially for problems with a natural recursive structure (e.g., tree traversals, factorial calculation).

3. Natural Fit for Certain Problems:

- Some problems are inherently recursive. For example, the structure of a tree (each node can be seen as a subtree) or problems like the Fibonacci sequence or the Tower of Hanoi are naturally suited for recursive solutions.

4. Reduction of Complexity:

- For problems where the solution involves multiple stages or levels, recursion can simplify the solution process by allowing each function call to handle a specific stage of the problem.

Drawbacks and Considerations

- Stack Overflow:

- Deep recursion can lead to stack overflow errors if the recursion depth is too large. This is because each function call adds a new layer to the call stack.

- Performance Issues:

- Recursive solutions may have higher time complexity compared to iterative solutions due to repeated calculations. Techniques like memoization (caching results of recursive calls) can help optimize performance.

- Debugging Complexity:

- Recursive solutions can sometimes be harder to debug, especially if there are logical errors in the base case or recursive case.

Overall, recursion is a powerful tool when used appropriately, providing a clear and elegant way to solve problems that can be divided into smaller, similar problems.

**Complexity Analysis of the Recursive Algorithm**

Time Complexity: O(n), where n is the number of years. The memoization technique ensures that each unique subproblem (specific number of years) is computed only once, and subsequent requests for the same number of years retrieve the result from the cache in constant time.

Space Complexity:

Space Complexity: O(n) due to the storage used by the memoization cache. Each entry in the cache corresponds to a unique number of years.