**FINANCIAL FORECASTING**

Recursion is a programming technique where a function calls itself repeatedly until it reaches a base case that stops the recursion. In other words, a function solves a problem by breaking it down into smaller instances of the same problem, which are then solved by the same function, until the solution to the original problem is found. Recursion is particularly well-suited for problems that have a tree-like structure, such as traversing a directory hierarchy or parsing a nested data structure. Recursion allows us to break down complex problems into smaller, more manageable sub-problems, making it easier to understand and solve them (Divide and Conquer).

Below are the methods by which one can make a recursive solution more efficient, in terms of avoiding repetitive calculations:

**Memoization:** Storing the results so that the previous computations can be fast.

**Dynamic programming:** A method for solving complex problems by breaking them down into simpler sub-problems, solving each sub-problem only once, and storing their solutions.

**Tail recursion optimization:** This method involves making the recursive solutions use less stack space.

**Iterative solution:** Replacing recursive solutions with iterative solutions which use loops.

**Culling:** Removing some of the extra branches from the search space.

Less resource-intensive alternative algorithms; reduce the depth of recursions so that stack overflow does not occur; lazy evaluation—delay doing a computation until it is really necessary.

Using these techniques will allow us to make our recursive solutions lean and efficient, and in doing so, will help eliminate many of the common pitfalls that occur with recursion, such as excessive memory usage.

Time Complexity:

The time complexity of this recursive function is **𝑂(𝑛)**

O(n) because it makes n recursive calls to compute the final value.