**Recursive Algorithms**

**Understanding Recursion**

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function. A recursive algorithm takes one step toward solution and then recursively call itself to further move. The algorithm stops once we reach the solution.

Since called function may further call itself, this process might continue forever. So it is essential to provide a base case to terminate this recursion process. It's useful for problems like financial forecasting where future values depend on previous ones (e.g., compound growth, Fibonacci-type sequences, etc.)

**Analysis And Optimization**

#### **Time Complexity: O(n)** where n is the number of years. This is mainly because there is only one recursive call happening in each function call unlike in Fibonacci series where there are 2 recursive calls happening at each function call so O(2^n).

**Space Complexity:** O(n) due to recursive call stack. This can be optimized with iteration.

**Optimization:** We can use iteration to avoid call stack overflow for large input size.

**Java Code:**

public static double calculateFV(double principal, double rate, int years) {

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

principal \*= (1+rate);

}

return principal;

}