**1. Understand Recursive Algorithms**

**Recursion** is a method of solving problems where the solution depends on solutions to smaller instances of the same problem. A recursive function calls itself with modified parameters to break down a problem into simpler sub-problems.

**Concept of Recursion**

* **Base Case**: This is the condition under which the recursion stops. Without a base case, recursion would continue indefinitely.
* **Recursive Case**: This involves the function calling itself with a simpler or smaller input, eventually leading to the base case.

**Advantages of Recursion**:

* **Simplifies Problem Solving**: For problems that can be naturally divided into smaller similar problems (e.g., calculating factorials, Fibonacci sequences).
* **Reduces Code Complexity**: Recursive solutions can be more concise and easier to understand compared to iterative solutions.

**Disadvantages of Recursion**:

* **Stack Overflow**: Excessive recursion can lead to a stack overflow if the recursion depth is too deep.
* **Overhead**: Recursive calls add overhead due to maintaining the call stack.

**2. Setup**

We will create a method to calculate the future value of an investment based on past growth rates using recursion.

**Future Value Calculation**:

The future value FV can be calculated using the formula:

FV=

where:

* PV = Present Value
* r = Growth Rate
* n = Number of Periods

For recursive calculation, the formula can be expressed as:

FV(n) = FV(n−1) × (1+r)

with the base case being:

FV(0)=PV

**4. Analysis**

**Time Complexity**

* **Time Complexity**: The time complexity of the recursive algorithm is O(n), where n is the number of years. This is because there are n recursive calls, each involving a constant amount of work.
* **Space Complexity**: The space complexity is also O(n) due to the call stack growth. Each recursive call adds a frame to the call stack until the base case is reached.

**Optimization to Avoid Excessive Computation**

To optimize the recursive solution and avoid potential issues such as excessive computation or stack overflow, consider the following approaches:

1. **Memoization**: Store previously computed results to avoid redundant calculations. This technique is particularly useful in problems with overlapping sub-problems.
2. **Iterative Approach**: For simple problems like calculating future value, an iterative approach can be more efficient. Iterative solutions avoid the overhead of recursive calls and are generally more space-efficient.