1. Explain the concept of recursion and how it can simplify certain problems.

Recursion is a programming technique where a function calls itself to solve a problem by breaking it down into smaller subproblems. Each recursive call reduces the original problem's size until a base case is reached, which stops further recursion.

- Recursion allows for cleaner, more readable code.
- Problems that involve repetition, sequences, or branching (like tree traversal, factorial, Fibonacci, or compound interest) are naturally modeled using recursion.
- In our financial forecasting, recursion simplifies computing future value by modeling each year's value based on the previous year, avoiding complex loops.

4. (a)Discuss the time complexity of your recursive algorithm.

The recursive forecasting algorithm computes the future value by calling itself n times (where n is the number of years), multiplying the previous value by the growth rate at each step.

Time Complexity:

- Time Complexity: O(n) — one call for each year.
- Space Complexity: O(n) — due to the call stack (each recursive call takes space in memory).

Though efficient for small n, recursive depth becomes a concern as n grows larger.

4.(b)Explain how to optimize the recursive solution to avoid excessive computation.

Ans: To avoid excessive computation or stack overflow in recursion, especially for large values of n, the recursive algorithm can be optimized in two main ways:

Optimization Techniques:

i. Use Iteration Instead of Recursion
   - Convert the recursion into a for loop (iterative approach) to eliminate call stack usage.
   - This reduces space complexity from O(n) to O(1).

ii. Memoization (for problems with overlapping subproblems)
   - Store already computed results in a table (array or map).
   - While not needed for this financial forecasting task (since it’s linear), it’s crucial in recursive algorithms like Fibonacci or dynamic programming.