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

Recursion is a programming technique where a function calls itself in order to solve a problem. Each recursive call breaks the problem down into smaller, more manageable sub-problems, until it reaches a condition that stops the recursion.

Recursion is particularly useful for problems that are:

* Naturally hierarchical (e.g., file systems, organizational structures)
* Inherently defined in terms of smaller instances (e.g., Fibonacci numbers, tree traversals)
* Solved through divide and conquer strategies (e.g., quicksort, mergesort)

Advantages of recursion:

* Makes code cleaner and easier to understand (for problems that fit)
* Often reduces complex iterative logic into simpler expressions
  + **Discuss the time complexity of your recursive algorithm.**

Let:

* n be the number of years.
* The function is called once per year.
* Each call does one multiplication: principal \* (1 + rate).

Total Number of Recursive Calls:

* The method calls itself once, reducing years by 1 each time.
* It continues until years == 0.

So, for n years:

* The number of calls = n + 1 (including the base case).
* Each call does O(1) work.

Total Time**:**

T(n) = O(1) + O(1)+ ⋯ + O(1) (n times) = O(n)

Final Time Complexity:

O(n)

* + **Explain how to optimize the recursive solution to avoid excessive computation.**

Why optimization Is needed:

* Each recursive call adds a new frame to the call stack.
* For large years, this causes:
  + Stack overflow error.
  + Unnecessary memory usage, even though each call only does a simple multiplication.

Optimisation Strategies:

* Iterative Approach:
  + public static double futureValueIterative(double principal, double rate, int years) {

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

principal \*= (1 + rate);

}

return principal;

}

* + Improved space complexity: O(1)
* Mathematical Formula:
  + public static double futureValueDirect(double principal, double rate, int years){

return principal \* Math.pow(1 + rate, years);

}

* + Improved time complexity: O(1)