**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Steps:**

1. **Understand Recursive Algorithms:**
   * Explain the concept of recursion and how it can simplify certain problems.
2. **Setup:**
   * Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
   * Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
   * Discuss the time complexity of your recursive algorithm.
   * Explain how to optimize the recursive solution to avoid excessive computation.

**Answer**

**Recursive Algorithms**

Recursion is a programming technique where a function calls itself to solve smaller instances of a problem until reaching a base case.

* Breaks down complex problems into simpler subproblems
* Makes code cleaner and easier to understand
* Often used in problems involving repetition, hierarchies, or math-based growth

Example: Computing future value with compound growth is inherently repetitive and can be expressed recursively.

**Program**

public class FinancialForecast {

public static double futureValue(double initialAmount, double growthRate, int n) {

if (n == 0) {

return initialAmount;

} else {

return futureValue(initialAmount, growthRate, n - 1) \* (1 + growthRate);

}

}

public static void main(String[] args) {

double initial = 10000;

double rate = 0.05;

int periods = 10;

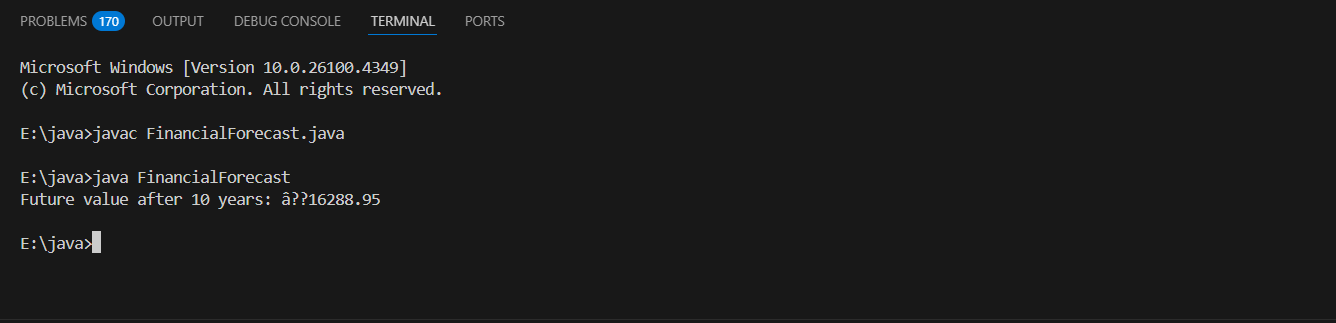
double result = futureValue(initial, rate, periods);

System.out.printf("Future value after %d years: ₹%.2f%n", periods, result);

}

}

**Output**

****

**Time Complexity of the Recursive Algorithm**

The recursive method for calculating future value is:

futureValue(initialAmount, growthRate, n) = futureValue(initialAmount, growthRate, n - 1) \* (1 + growthRate)

* Each call reduces n by 1 and makes exactly one recursive call.
* The recursion continues until n == 0.

**Time Complexity:**

* T(n) = T(n - 1) + O(1)
* This recurrence solves to O(n)

So, the time complexity is O(n).

**Space Complexity:**

* Because each recursive call is placed on the call stack until reaching the base case, the space complexity is also O(n).

**How to Optimize the Recursive Solution?**

**Problem:**

* Recursive calls use extra memory (call stack)
* With very large n, this can lead to stack overflow
* Recalculations in other recursive problems can cause exponential time

**Optimized Approach : Iterative Version**

Replace recursion with a loop:

public static double futureValueIterative(double initialAmount, double growthRate, int n) {

double result = initialAmount;

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

result \*= (1 + growthRate);

}

return result;

}

* Time Complexity: O(n)
* Space Complexity: O(1)
* This is more efficient and avoids recursion limits.

| **Approach** | **Time Complexity** | **Space Complexity** | **Safe for large n** | **Preferred** |
| --- | --- | --- | --- | --- |
| Recursive | O(n) | O(n) | Risk of stack overflow | No |
| Iterative | O(n) | O(1) | Yes | Yes |