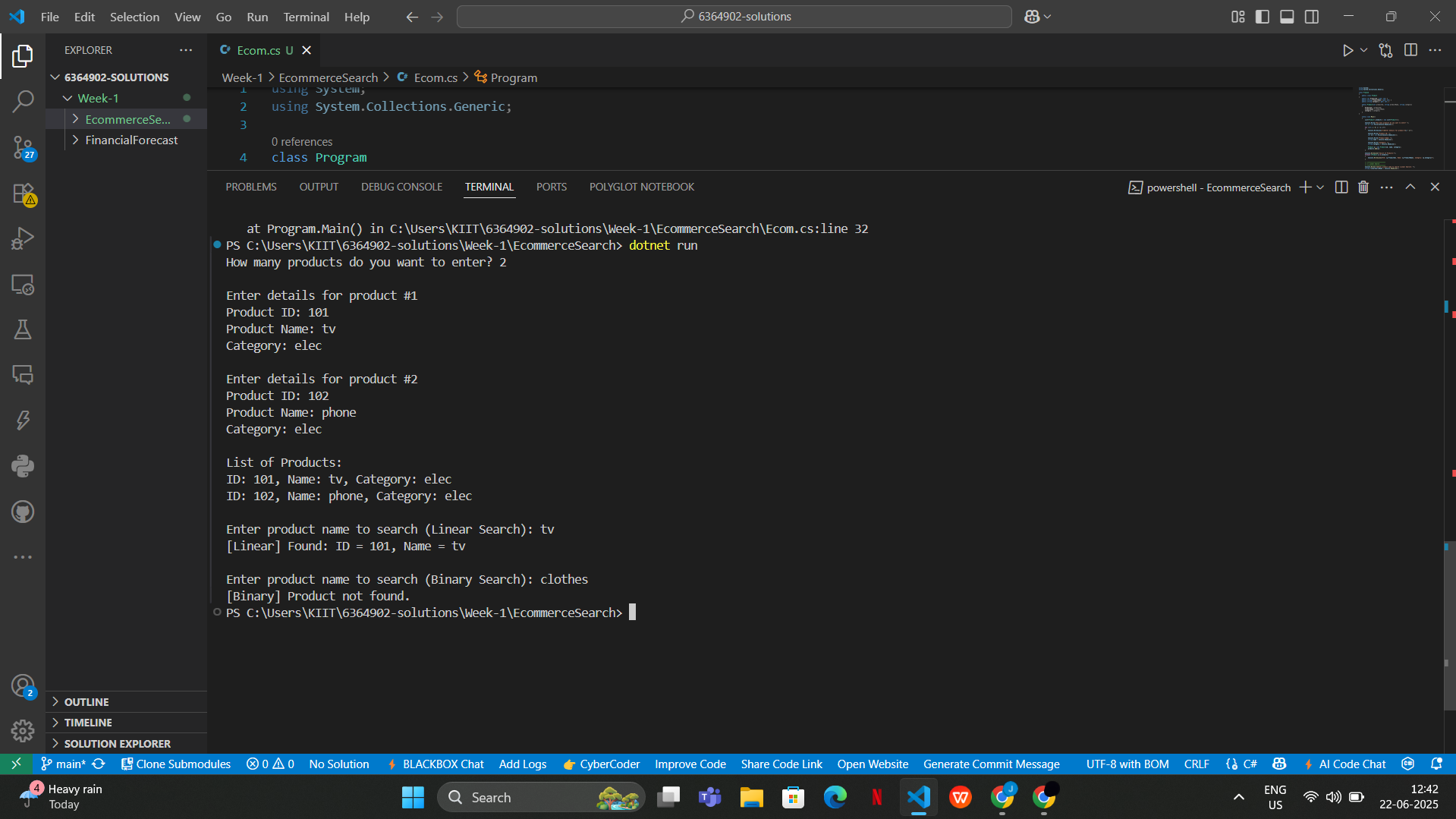
**Week-1 Solutions and Output**

**Data Structures:-**

**Exercise 2: E-commerce Platform Search Function**

Output-

Question- Explain Big O notation and how it helps in analyzing algorithms.

Answer- **Big O Notation:**

Big O notation describes the **upper bound of the time or space complexity** of an algorithm as the input size grows. It's a way to express how the **runtime scales**.

* **O(1)** – Constant time
* **O(n)** – Linear time
* **O(log n)** – Logarithmic time
* **O(n log n)** – Linearithmic time
* **O(n²)** – Quadratic time

Question- Describe the best, average, and worst-case scenarios for search operations.

#### **Answer- Best, Average, and Worst-Case Scenarios:**

| **Case** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| Best Case | O(1) | O(1) (middle element) |
| Average Case | O(n) | O(log n) |
| Worst Case | O(n) | O(log n) |

Compare the time complexity of linear and binary search algorithms.

Discuss which algorithm is more suitable for your platform and why.

Answer- **Time Complexity Comparison**

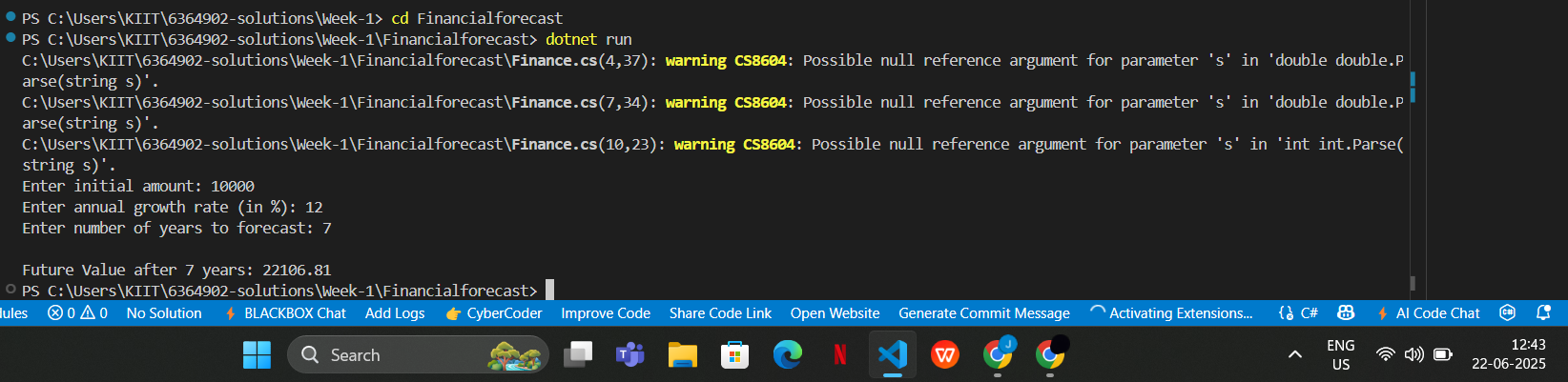
| **Algorithm** | **Time Complexity** |
| --- | --- |
| Linear Search | O(n) |
| Binary Search | O(log n) |

#### **Recommendation**

* Use **Linear Search** for **unsorted or small datasets**.
* Use **Binary Search** for **large, sorted datasets** – it is much more efficient.

**Exercise 7: Financial Forecasting**

Output-



Question:

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

Answer:

**Recursion** is a programming technique where a method **calls itself** to solve a smaller sub-problem of the original problem.

Recursion simplifies problems that have a **repeating pattern** or can be **broken into smaller subproblems**, like:

* Factorials
* Fibonacci series
* Tree traversal
* Financial growth prediction over time (compounding)

Questions:

**Discuss the time complexity of your recursive algorithm.**

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

Answers:

### **Time Complexity**

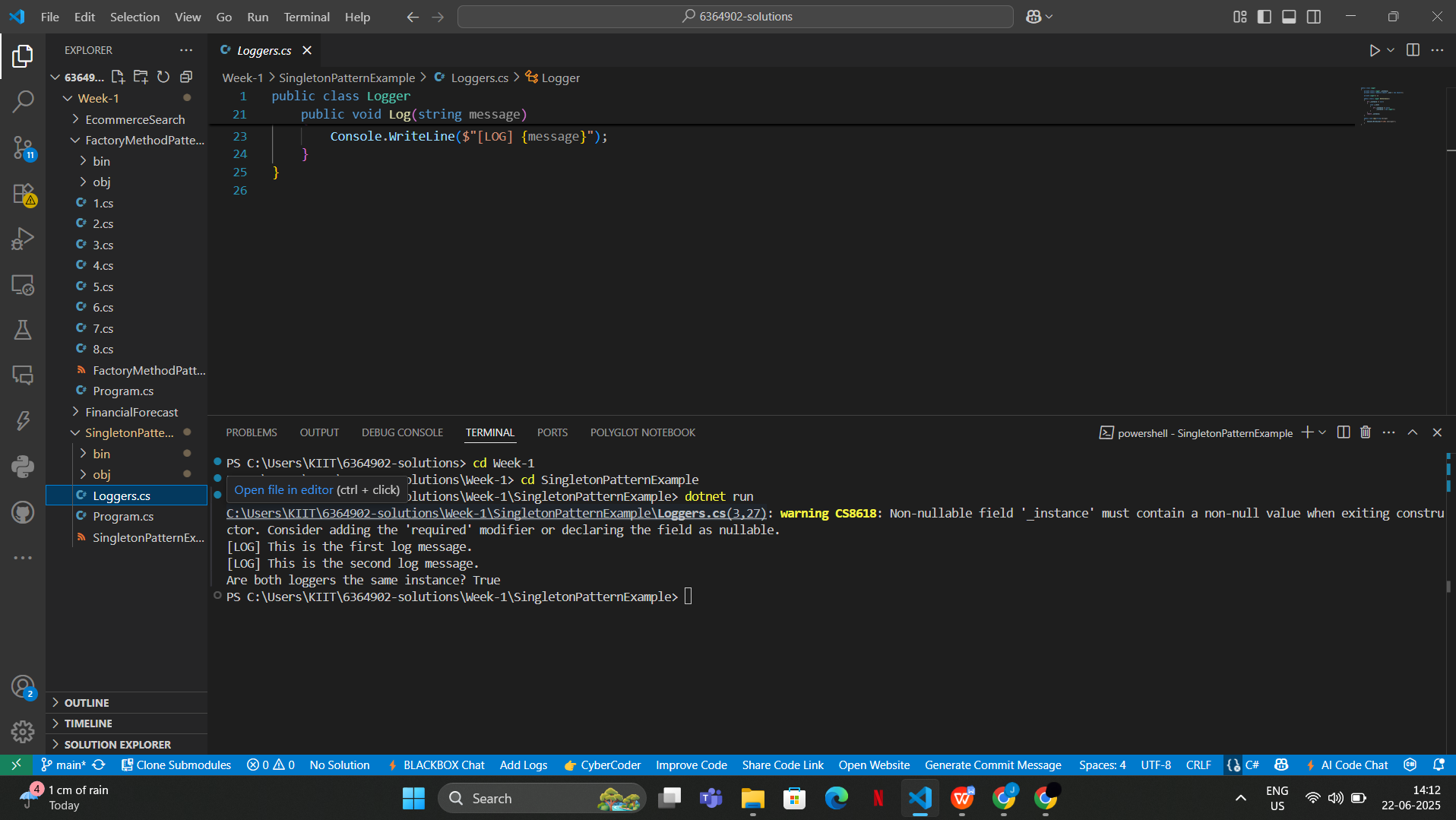
* Each recursive call reduces years by 1.
* Total recursive calls = n, where n = years.

**Time Complexity:** **O(n)** — Linear time (1 recursive call per year)

**Design Patterns and Principle:**

**Exercise 1: Implementing the Singleton Pattern**

Output:



**Exercise 2: Implementing the Factory Method Pattern**

Output:

