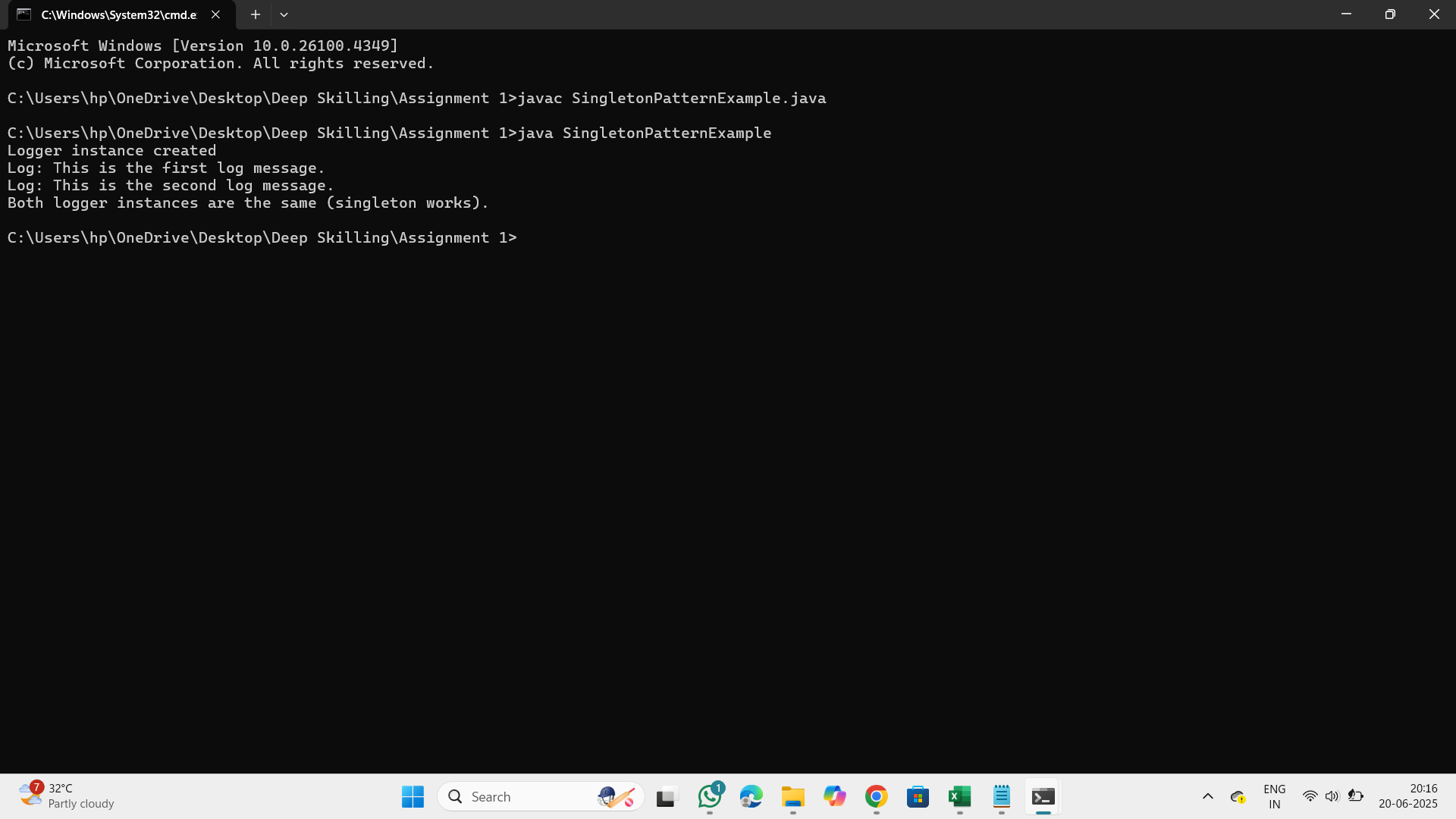
***WEEK1 ASSIGNMENT***

**Design Principles and Patterens**

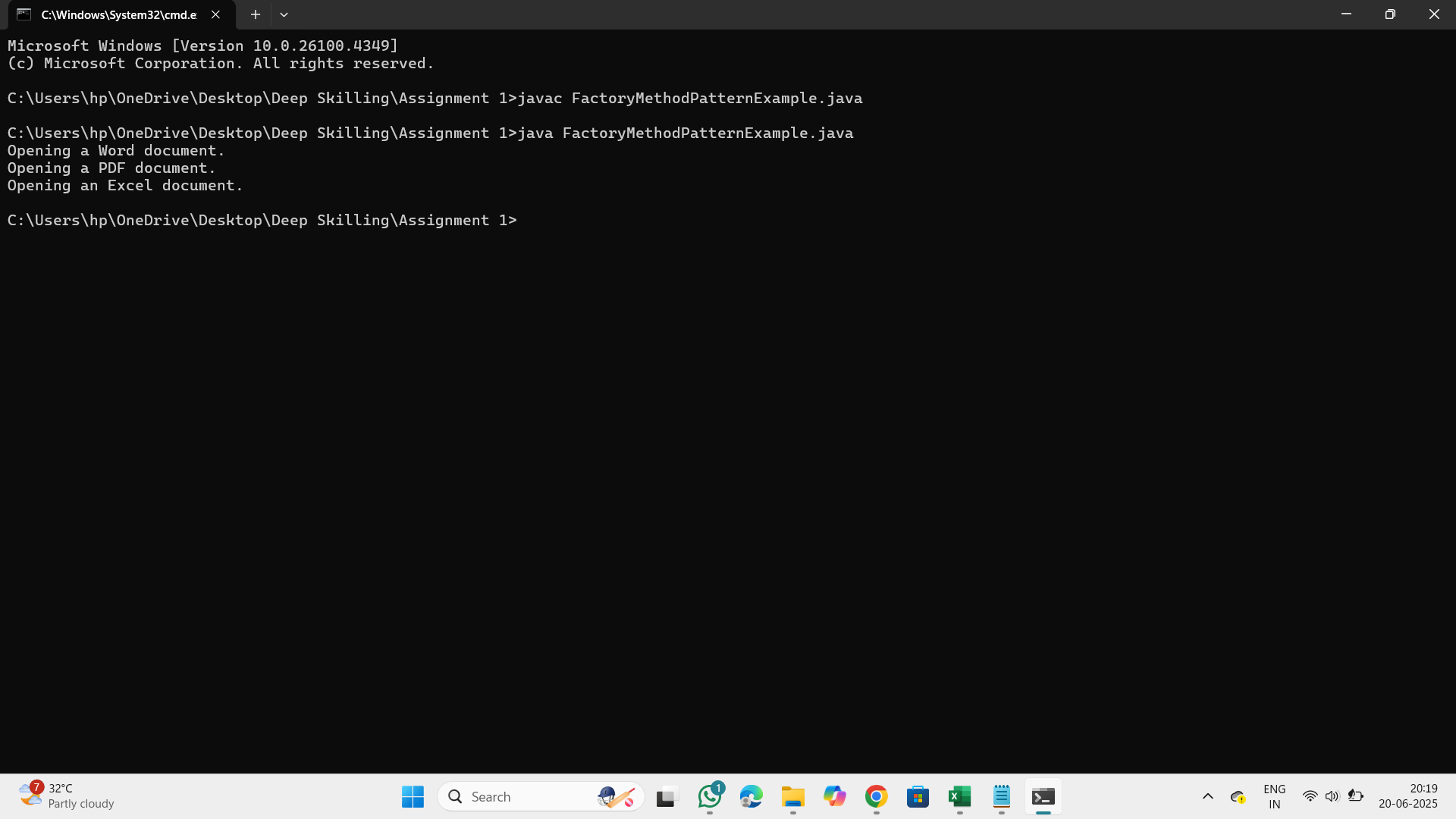
**1.Implementing the Singleton Pattern:**

**Output**

****

**2.Implemeting the Factory Method Pattren**

**Output**

****

**Data Structures and algorithms**

**3.E-Commerce Platform Search Function**

**Why Data Structures and Algorithms are Essential for Handling Large Inventories**

Managing a large inventory involves storing, searching, updating, and retrieving data efficiently. As the size of the inventory grows, performing these operations quickly becomes critical for performance, scalability, and accuracy. Here's why data structures and algorithms are essential:

**Efficient Storage and Retrieval**

Large inventories require optimized storage formats to reduce memory use and allow fast access to data like product ID, name, price, quantity, etc.

**Fast Search Operations**

Algorithms help in quickly locating items based on criteria (like product ID or name). Without proper algorithms, search time can become too slow.

**Dynamic Updates**

Inventories change constantly—new items are added, stock is updated, or products are removed. Efficient data structures allow for quick insertions, deletions, and updates.

**Scalability**

Good algorithms ensure that the system remains fast and responsive even as inventory grows from hundreds to millions of items.

**Decision-Making Support**

Algorithms like sorting and filtering help in making business decisions such as identifying low-stock items or the most sold products.

**Types of Data Structures Suitable for Inventory Management**

**Arrays or Lists**

Use case: Small inventories or when fast iteration is needed.

Limitation: Searching and deleting items is slow (O(n) time complexity).

Hash Tables (HashMaps in Java)

Use case: Fast lookups by unique keys like product ID.

Advantage: Offers average-case O(1) time for insert, search, and delete.

Example: HashMap<ProductID, ProductDetails>

**Trees (Binary Search Tree, AVL Tree, B-Trees)**

Use case: Keeping items sorted by name, category, or stock.

Advantage: Good for range queries and ordered traversal.

Example: B-Trees are used in databases for indexing large datasets.

Heaps (Priority Queues)

Use case: Quickly retrieving items with the highest or lowest stock.

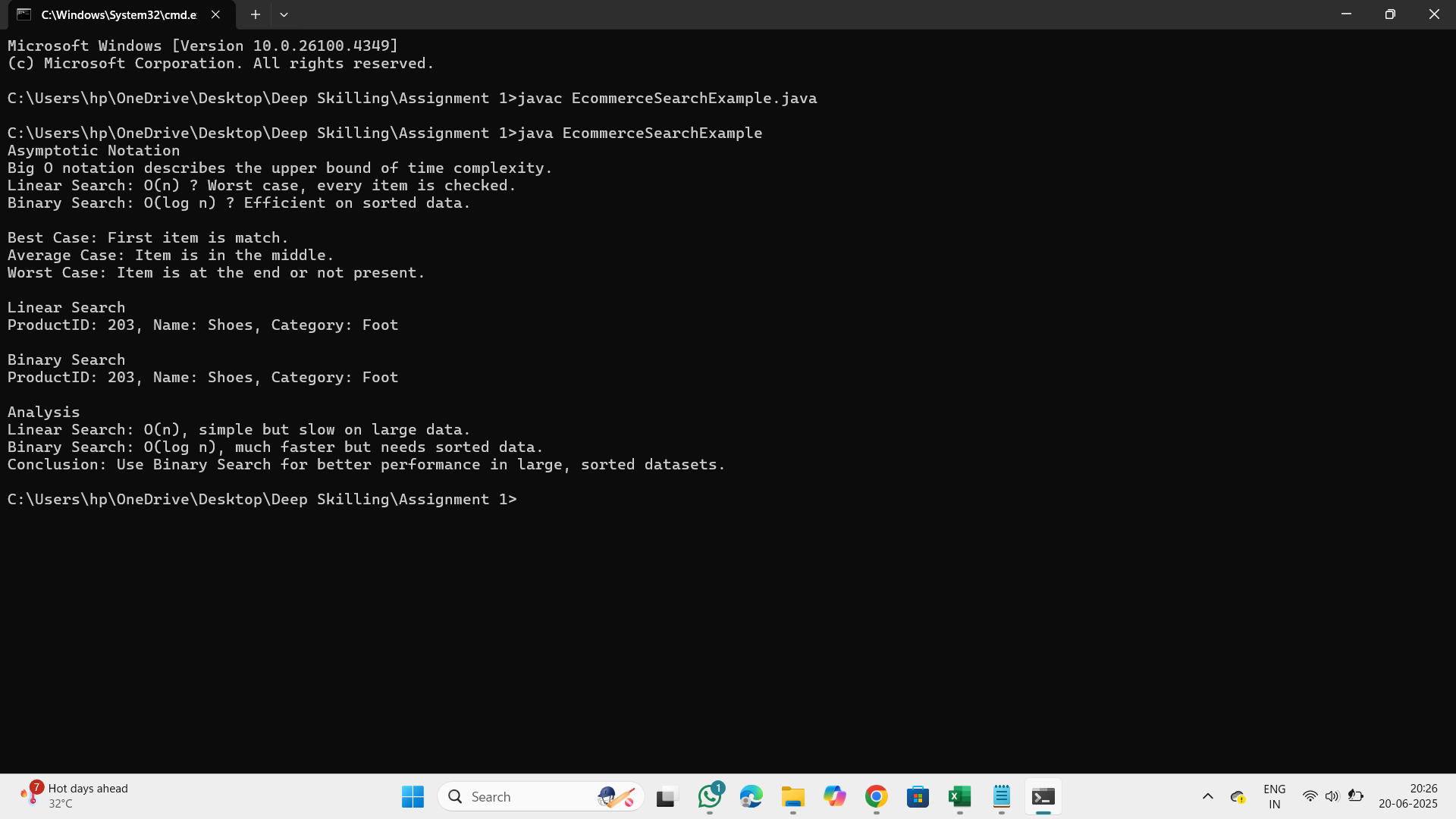
Example: Max-heap for most stocked items, Min-heap for low-stock alerts.

**Graphs**

Use case: For managing inventory in a supply chain across multiple warehouses or locations.

Advantage: Can model complex relationships and dependencies.

**Output**

****

**4.Financial Forecasting**

Recursion is a programming technique where a function calls itself to solve a smaller part of the overall problem. This process continues until it reaches a base case, which stops further recursive calls**.**

**How Recursion Simplifies Problems**

**Breaks Down Complex Problems**

Recursion breaks a large problem into smaller, manageable subproblems. This is useful in problems like tree traversal, backtracking, and divide-and-conquer algorithms.

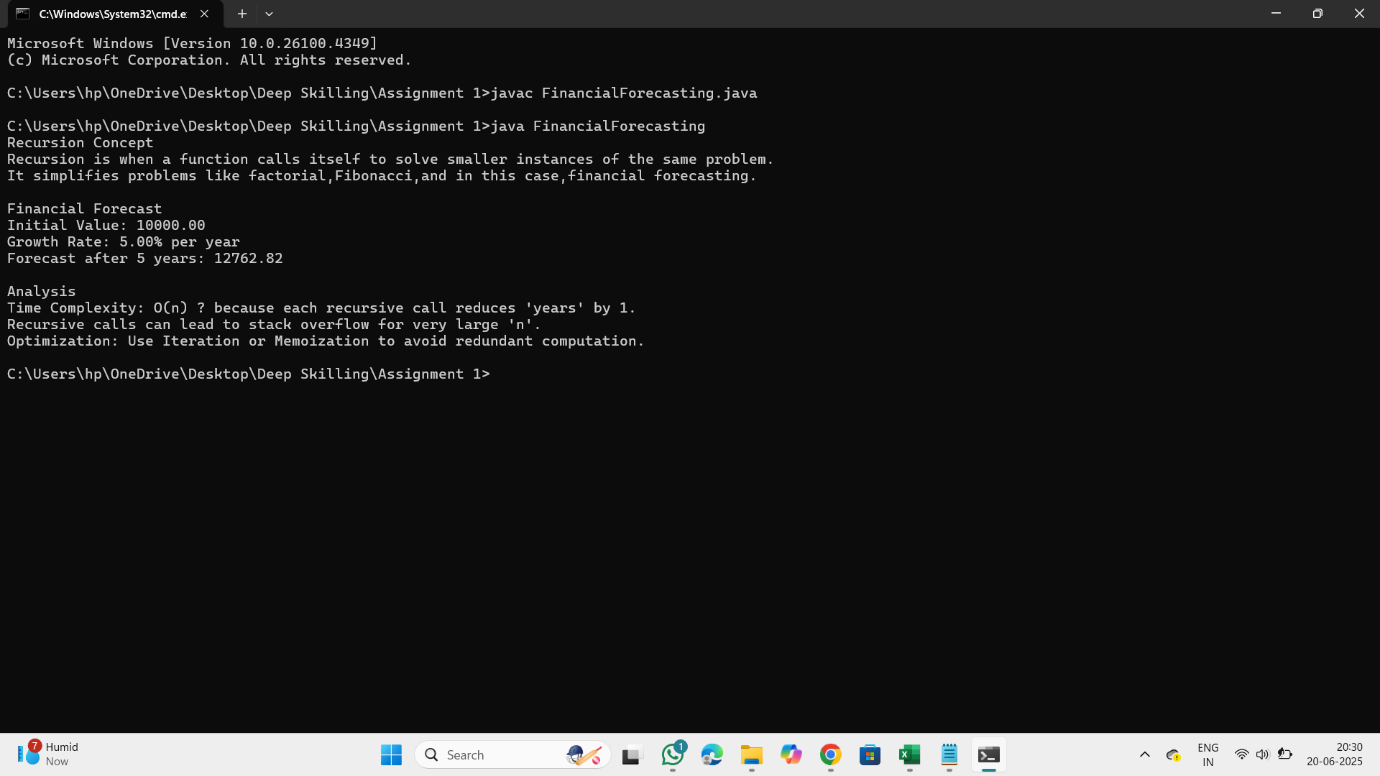
**Reduces Code Complexity**

Recursive solutions are often more concise and readable compared to iterative approaches.

**Natural Fit for Recursive Structures**

Structures like trees, graphs, and nested directories are naturally recursive and are easier to process recursively.

**Output**

****