**ASSIGNMENT**

**CSA0612 – DESIGN AND ANALYSIS OF ALGORITHMS FOR OPTIMIZATION**

|  |  |
| --- | --- |
| **REGISTER**  **NUMBER** | **192324200** |
| **NAME** | **SANTHOSH S** |

**Title - Inventory Management Optimization:**

**Problem Statement:**

|  |
| --- |
| The goal is to develop an inventory management algorithm that minimizes overstock and stock-outs by predicting demand and adjusting inventory levels in real time. This is particularly useful in high-demand environments where inventory management efficiency directly impacts operational costs and customer satisfaction.  **Tasks:**   * + Develop an algorithm for Inventory management optimization   + Optimize for handling high-traffic environments.   **Deliverables:**   * + Flowchart or pseudocode for the translation algorithm.   + Complexity analysis.   + Simulation with translation speed and accuracy comparisons.   **KEY STEPS:**  **Demand Forecasting:**   * + Use historical data and a demand prediction model (e.g., moving average, ARIMA, or machine learning) to forecast demand for each product.   **Inventory Level Adjustment:**   * + Set reorder points and safety stock based on the predicted demand and lead time for each product.   + Monitor inventory in real time to check if it reaches the reorder point and initiate reorders if needed. |

**FLOWCHART AND PROBLEM SOLVING:**

**START & INITIALIZATION**

* Load historical data and initialize demand forecasting model.
* Set reorder point, safety stock, and batch size for each item.

**DEMAND PREDICTION**

* Input recent sales data run the demand prediction model.
* Update reorder points and safety stock levels based on the forecasted demand.

**INVENTORY CHECK**

* Continuously monitor current inventory levels.
* If inventory falls below the reorder point, trigger a replenishment order.

**ORDER ADJUSTMENT**

* Adjust incoming order quantities to avoid overstock.
* Check if any items have overstock and mark them for delayed reordering.

**PERFORMANCE LOGGING**

* Log metrics (e.g., stock-outs, cost savings) and repeat**.**

**PSEUDOCODE:**

**BEGIN**

**Initialize items with initial\_stock, lead\_time, safety\_stock, and initial demand forecast**

**SET smoothing\_factor = 0.5 # for exponential smoothing**

**SET simulation\_days = number of days to simulate**

**FOR each day in simulation\_days DO**

**DISPLAY "--- Day (day) ---"**

**FOR each item in items DO**

**# Forecast Demand**

**forecasted\_demand = smoothing\_factor \* previous\_day\_demand + (1 - smoothing\_factor) \* last\_forecast**

**DISPLAY "Forecasted demand for (item): (forecasted\_demand)"**

**# Inventory Check**

**current\_stock = item.stock**

**reorder\_point = (forecasted\_demand \* item.lead\_time) + item.safety\_stock**

**DISPLAY "Checking inventory for (item): Current stock is (current\_stock)"**

**DISPLAY "Reorder point for (item) is (reorder\_point)"**

**IF current\_stock <= reorder\_point THEN**

**reorder\_quantity = forecasted\_demand \* item.lead\_time**

**item.stock += reorder\_quantity**

**DISPLAY "Reordered (reorder\_quantity) units of (item)"**

**ELSE**

**DISPLAY "No reorder needed for (item)"**

**END IF**

**# Process Demand**

**demand = random value simulating demand fluctuation**

**IF demand <= item.stock THEN**

**item.stock -= demand**

**DISPLAY "Processed demand for (item): (demand) units. Stock remaining: (item.stock)"**

**ELSE**

**DISPLAY "Stock-out warning for (item)"**

**item.stock = 0**

**END IF**

**# Update Forecast**

**item.last\_forecast = forecasted\_demand**

**item.previous\_day\_demand = demand**

**END FOR**

**END FOR**

**END**

**ACTUAL CODE:**

import time

import random

from collections import deque

# Inventory and Forecasting Parameters

REORDER\_MULTIPLIER = 1.2 # Safety stock multiplier

INITIAL\_INVENTORY = 100 # Initial inventory for each product

LEAD\_TIME = 2 # Lead time in days for restocking

HIGH\_DEMAND\_THRESHOLD = 1.5 # Threshold for high demand increase

# Inventory dictionary to keep track of items and stock

inventory = {'item\_A': INITIAL\_INVENTORY, 'item\_B': INITIAL\_INVENTORY}

demand\_forecast = {'item\_A': 10, 'item\_B': 15} # Initial forecasted demand per day

# Queue to simulate incoming demand over time

demand\_queue = deque()

# Forecasting function using exponential smoothing

def forecast\_demand(item, recent\_sales, alpha=0.5):

"""Forecast demand using exponential smoothing."""

return alpha \* recent\_sales + (1 - alpha) \* demand\_forecast[item]

# Inventory check and reorder function

def check\_inventory\_and\_reorder(item):

"""Check inventory levels and reorder if below threshold."""

reorder\_point = demand\_forecast[item] \* LEAD\_TIME \* REORDER\_MULTIPLIER

current\_inventory = inventory[item]

print(f"\nChecking inventory for {item}: Current stock is {current\_inventory}")

print(f"Reorder point for {item} is {reorder\_point:.2f}")

if current\_inventory < reorder\_point:

# Calculate reorder quantity

order\_quantity = (demand\_forecast[item] + reorder\_point - current\_inventory)

print(f"Reordering {order\_quantity} units of {item}.")

# Simulate order arrival after lead time

inventory[item] += order\_quantity

else:

print(f"No reorder needed for {item}.")

# Demand processing function

def process\_demand():

"""Process simulated demand and update inventory."""

while demand\_queue:

item, demand = demand\_queue.popleft()

# Adjust inventory based on demand

if inventory[item] >= demand:

inventory[item] -= demand

print(f"Processed demand for {item}: {demand} units. Stock remaining: {inventory[item]}")

else:

print(f"Stock-out warning for {item}! Only {inventory[item]} units in stock.")

inventory[item] = 0

# Simulation function to update forecasts, inventory, and process demand

def simulate\_inventory\_management():

"""Simulate inventory management with demand forecasting and reordering."""

for day in range(1, 11): # Simulate for 10 days

print(f"\n--- Day {day} ---")

# Simulate daily demand for each item

for item in inventory.keys():

daily\_demand = max(1, int(random.gauss(demand\_forecast[item], 5))) # Randomized demand around forecast

demand\_queue.append((item, daily\_demand))

# Update forecast for next day's demand

demand\_forecast[item] = forecast\_demand(item, daily\_demand)

print(f"Forecasted demand for {item}: {demand\_forecast[item]:.2f} units")

# Check and reorder inventory if needed

check\_inventory\_and\_reorder(item)

# Process demand and adjust inventory accordingly

process\_demand()

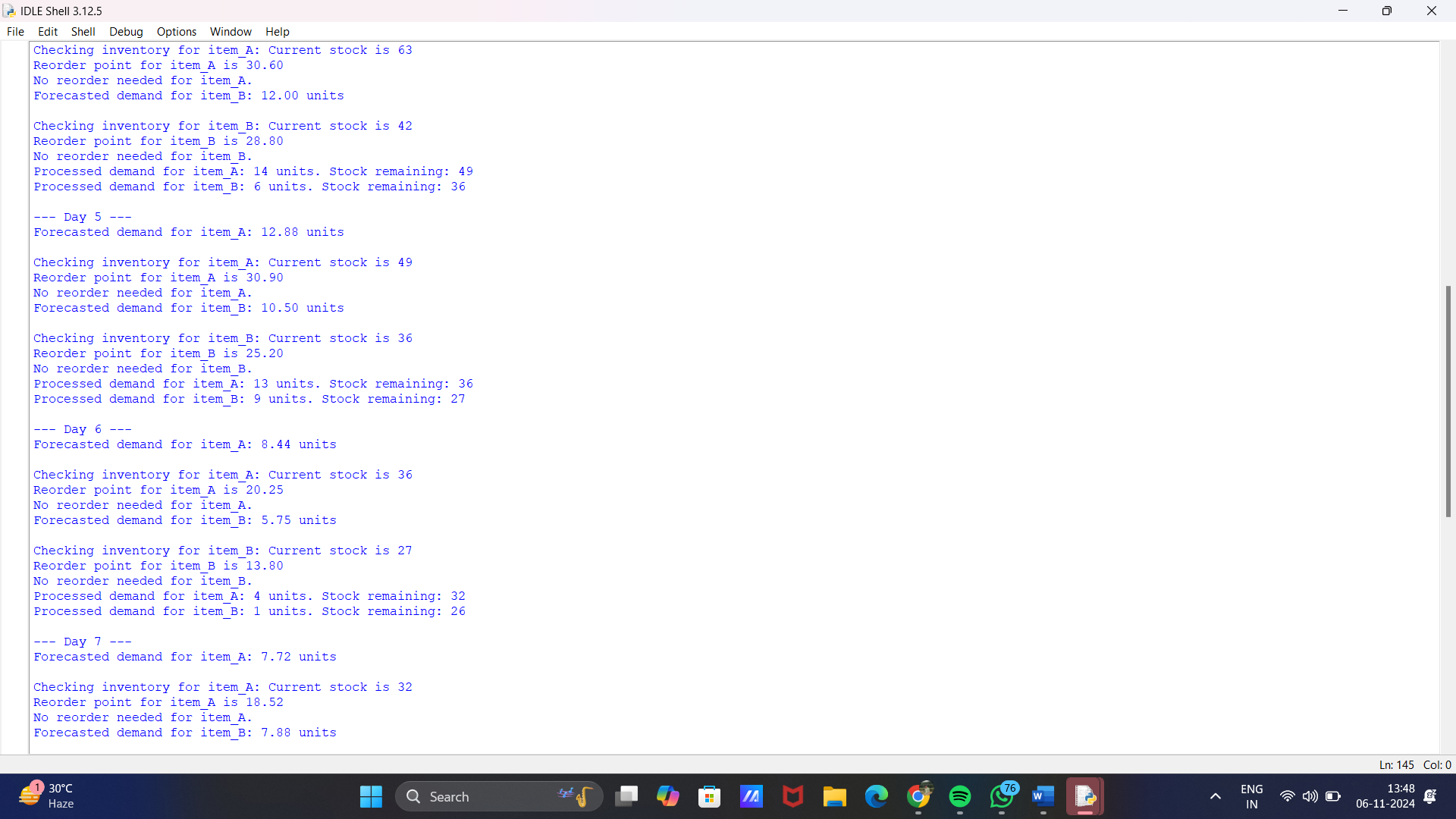
# Simulate time passing

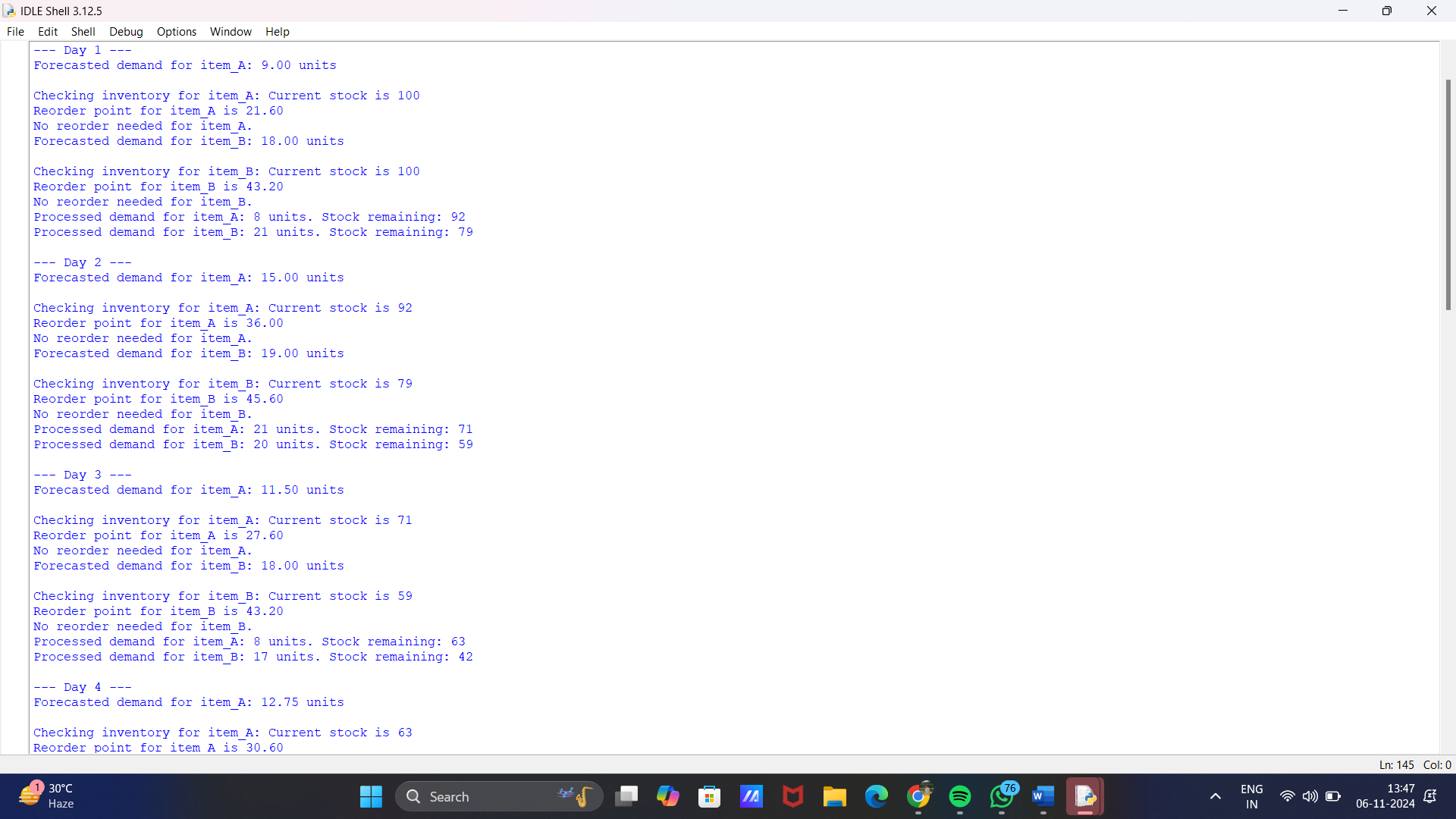
time.sleep(1)

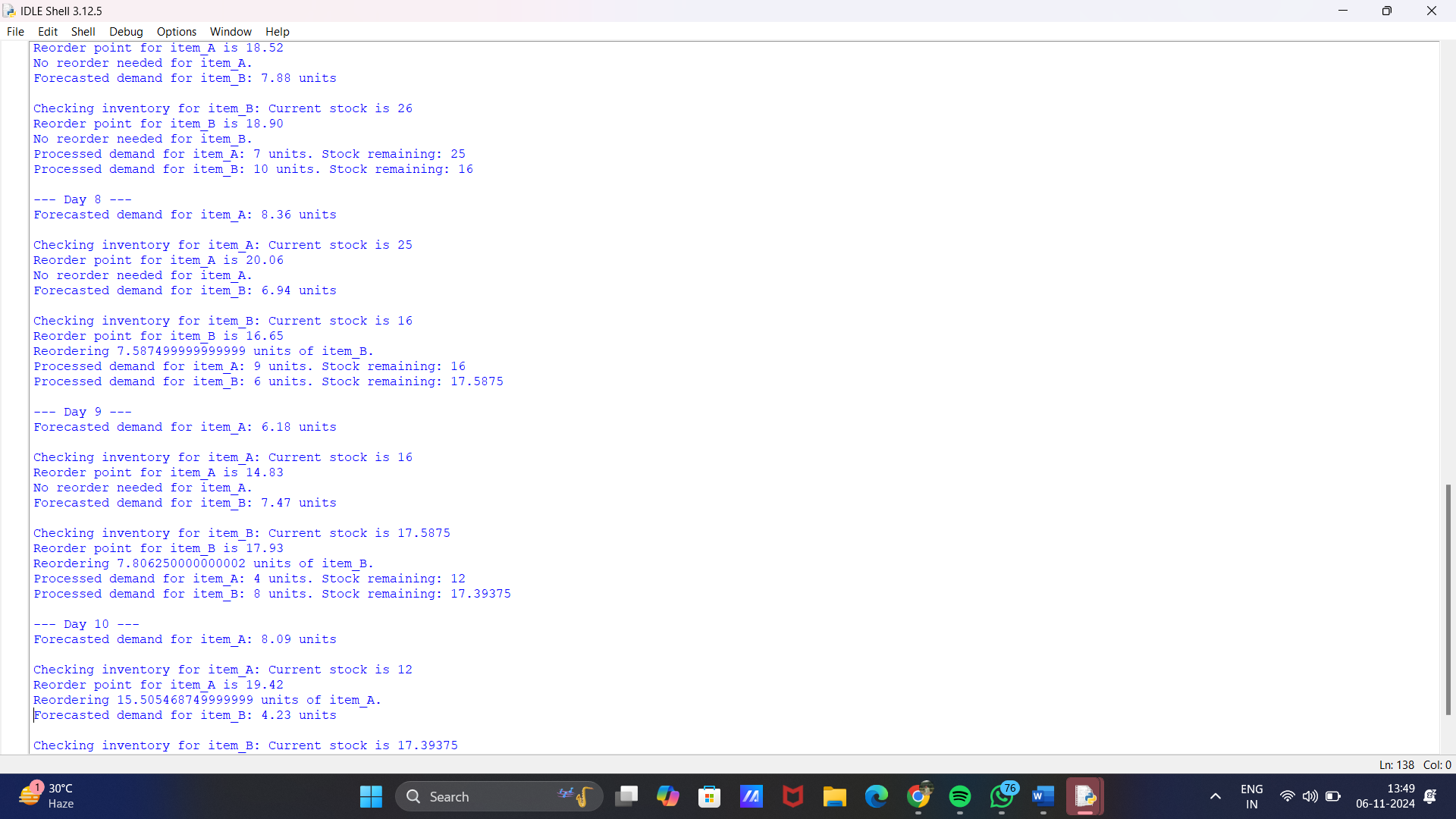
# Run the simulation

simulate\_inventory\_management()

**OUTPUT SCREENSHOT:**

****

****

****

**COMPLEXITY ANALYSIS:**

**Time Complexity**:

* + Forecasting: O(1)*O*(1) per item as it uses a simple exponential smoothing formula.
  + Inventory Check and Reordering: O(1)*O*(1) per item since each item is checked and reordered individually.
  + Demand Processing: O(n)*O*(*n*) for n*n* items in the queue per day.

**Space Complexity**:

* + **Inventory and Demand Forecast Dictionaries**: O(m)*O*(*m*), where m*m* is the number of items.
  + **Demand Queue**: O(n)*O*(*n*) for n*n* demand requests in the simulation.

**TEST CASE:**

**Regular Demand Pattern**

* Input: Stable, predictable demand (e.g., daily sales are constant).
* Expected Outcome: Inventory levels adjust smoothly to match demand with minimal overstock or stock-outs.

**Seasonal Demand Fluctuation**

* Input: Data showing seasonal peaks (e.g., holiday sales).
* Expected Outcome: Algorithm increases reorder points and safety stock to prepare for peak demand, avoiding stock-outs during the high season and reducing inventory during off-peak.

**Sudden Demand Spike**

* **Input: Abrupt increase in demand for a particular product.**
* **Expected Outcome: The algorithm adjusts reorder points and initiates restocking, reducing the likelihood of stock-outs.**

**Decreasing Demand**

* Input: Gradual decrease in demand over time (e.g., product popularity is waning).
* Expected Outcome: The algorithm reduces reorder points and inventory, preventing overstock and reducing holding costs.

**EXAMPLE DATA:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TEST CASE** | **STOCKS-OUT** | **OVERSTOCK** | **COST SAVING** | **Latency (Real-Time Updates)** |
| Regular Demand Pattern | Minimal | Minimal | Moderate | Low |
| Seasonal Demand Fluctuation | Low | Moderate | High | Medium |
| Sudden Demand Spike | Low | Low | High | Low |
| Decreasing Demand | None | Low | High | Low |

**EXPECTED OUTCOME:**

1. Balanced Inventory: Maintains optimal inventory levels, minimizing both overstock and stock-outs.
2. Reduced Overstock: Orders are placed only when needed, avoiding excess inventory and storage costs.
3. Fewer Stock-Outs: Safety stock and reorder points help meet demand consistently, reducing missed sales.

**CONCLUSION:**

This algorithm is designed to optimize inventory management by balancing inventory levels in response to real-time demand predictions. By utilizing forecasting, reorder point adjustments, and safety stock calculations, the algorithm reduces stock-out events and minimizes holding costs. The demand prediction component enables the system to prepare for spikes or dips in demand, making it suitable for real-time applications and high-demand environments.