**Order Matching with Price-Time Priority Algorithm (Simplified)**

The order matching process is driven by two key principles:

1. **Price Priority**: Orders with better prices (lower for buy orders, higher for sell orders) are matched first.
2. **Time Priority**: If multiple orders have the same price, earlier orders (based on timestamp) are matched first.

**Buy Order:-**

**Market Orders**:

* First, attempts to fulfill the order from the company's available shares at the current market price.
* If not fully fulfilled, matches it with pending Sell orders from other traders, **sorted by the lowest price and earliest time**. PTPA take place

**Limit Orders:**

* Matches company shares or other traders' Sell orders priced **below or equal** to the buyer's limit price, sorted by the same priority (price-time).
* If unfulfilled by the end of the day, cancels the remaining quantity.

**Sell Order:-**

Validates if the seller owns enough stock to execute the order.

**Market Orders**:

* Matches with Buy orders from other traders, sorted by the highest price and earliest time. PTPA take place

**Limit Orders**:

* Matches with Buy orders priced **at or above the seller's limit price**, using the same price-time priority.
  + Unfulfilled Limit orders are canceled at the end of the day.

**We plan to use LSTM for stock price prediction and show stock trends**

We are:

1. **Fetching historical stock data** from your database (stocks\_trade table).
2. **Preparing the data** for training an LSTM model to predict future stock prices.
3. **Training the model** using historical closing prices.
4. **Saving the trained model** for use in future predictions.

* **Fetch and Aggregate Historical Data**:
  + From stocks\_trade, we group the data by stock ID and trade date.
  + We calculate:
    - Opening price (first trade of the day),
    - Closing price (last trade of the day),
    - High price (maximum price of the day),
    - Low price (minimum price of the day),
    - Total volume (sum of all quantities traded in a day).
* **Filter Data for One Stock**:
  + Select one stock (e.g., stock\_id = 1) for simplicity.
* **Scale the Closing Prices**:
  + Convert the closing prices to a range of 0 to 1 for better LSTM performance.

**How It Works in Your Project**

1. **Historical Data**:
   * Ensure the stocks\_trade table contains data for the stocks you want to predict.
2. **Run the Script**:
   * Execute train\_lstm\_model.py. It will:
     + Fetch and preprocess the data.
     + Train the LSTM model using historical closing prices.
     + Save the model for future predictions.
3. **Use the Trained Model**:
   * In your prediction script (stock\_price\_prediction.py), load lstm\_model.h5.
   * Use it to predict stock prices based on current and recent trade data.

**Short Summary**

* **Input**: Historical trade data (e.g., opening price, closing price, etc.) from your database.
* **Output**: A trained LSTM model (lstm\_model.h5) that can predict stock prices.
* **Django**: Ideal for building secure, scalable, and fast backend systems with its powerful ORM, built-in security features, and rapid development capabilities.
* **Angular**: Perfect for creating dynamic, responsive, and modular frontends with its component-based architecture, two-way data binding, and robust toolset.
* **PostgreSQL**: A reliable, scalable, and feature-rich database with support for advanced querying, data integrity, and handling large-scale applications efficiently.