Stock Market Matching Engine

Introduction

For my semester project, I built a **Stock Market Matching Engine**. This system is similar to the basic part of real stock exchanges where buy and sell orders are matched. I tried to make it fast, efficient, and close to how real trading systems work.

The project includes creating users, placing orders, storing a portfolio, matching trades automatically, showing the order book, showing trade history, and canceling orders.
To make everything efficient, I used different data structures like **Hash Maps**, **B-Trees**, and **Queues**.
In this report, I will explain why I selected these structures and how they helped me.

Main Features

Create Users

Allows users to register in the system.

Place Orders

Users can place buy or sell orders at any price and quantity.

Portfolio View

Shows the current holdings of every user.

Matching Engine

Matches buy and sell orders based on price and time priority.

Full Order Book

Shows all buy and sell orders in the system.

Trade History Viewer

Shows all completed trades.

Cancel Order

Allows users to cancel active orders.

Data Structures Used and Why

1. Hash Map

I used Hash Maps in three different places:

a. Hash Map for OrderID → Order

I needed a quick way to find an order from its ID.
A hash map gives **O(1)** average time, which is very fast.

b. Hash Map for UserID → User

To quickly access a user when placing orders or updating portfolio.

c. Hash Map for Symbol → OrderBook

If different stocks exist (like AAPL, TSLA), I can store each one's order book in a hash map.
This makes looking up a stock's order book extremely fast.

Why Hash Map?
Because hash maps are the best choice for **fast search**.
Whenever I needed to find something quickly, I used a hash map.

2. B-Tree (for Order Book)

I used two B-Trees:

- **Buy Tree**
- **Sell Tree**

In real stock markets, buy and sell orders must be sorted by price.
A B-Tree keeps all keys (prices) **sorted** and allows fast insertion, deletion, and search.

Why B-Tree for Order Book?

Because the matching engine must always know:

- the **highest buy** price
- the **lowest sell** price

A B-Tree helps me:

- keep prices sorted automatically
- insert new price levels quickly

- search for best price fast
- find next or previous price easily

This makes order matching very efficient.

3. OrderQueue (Queue for Orders with Same Price)

Sometimes many orders come with the **same price**.
For example:

- BUY 100@10
- BUY 50@10
- BUY 70@10

All are same price, so I used a queue.

Why a Queue?

Because real markets use **FIFO – First In, First Out** for fairness.

The person who placed the order first should get matched first.
A queue maintains exactly this behavior.

What Is Left to Implement

1. Socket Programming

To connect the backend with a frontend application.
After this, the system can work like a real server.

2. Frontend

A user interface is still needed so users can:

- view order book
- place orders
- check portfolio
- cancel orders
- see trade history

Once frontend + backend connect, the system will feel like a real trading platform.

Conclusion

This project helped me understand how real stock exchanges work behind the scenes.
Using Hash Maps, B-Trees, and Queues made my system fast and efficient.
I learned about data structures, order matching, and system design.
With sockets and a frontend, this project can become a complete working trading application.