A matching engine is a software program that matches buy and sell orders for a stock exchange. The engine accepts a series of limit orders and either fills them immediately if there is a matching order or adds them to the order book if there is not. To create a matching engine, we can use a variety of algorithms to determine how to fill orders. Some common algorithms include first-in-first-out (FIFO), pro rata, random, and most satisfied customers. First-in-first-out (FIFO) is a simple algorithm that fills orders in the order that they are received. This means that the first order received is the first one to be filled. Pro rata is an execution principle taking into account only the price or, respectively, the limit price of the orders in the order book, and not the usual price/time priority. As soon as new orders are entered into the trading system, it searches the order book and automatically executes matching orders in proportion to the orders contained in the order book. Random is another simple algorithm that fills orders randomly. This means that orders are filled in a random order, rather than in the order that they are received. Most satisfied customers is an algorithm that prioritizes filling orders from customers who have had their orders filled the least number of times. This can help to ensure that all customers are treated fairly and that orders are filled in a way that maximizes customer satisfaction.

To create a matching engine, we can use a variety of data structures and algorithms to manage the order book and fill orders. Some common data structures for managing the order book include linked lists and binary trees. These data structures can be used to efficiently store and retrieve orders from the order book. Additionally, we can use metrics such as average time an order is fulfilled to evaluate the performance of the matching engine. By tracking this metric, we can identify areas for improvement and make changes to the algorithm to optimize the engine's performance.

I am trying to solve the problem of having a matching engine that could be used by everyday people such as you and me. Most stock exchanges do not disclose their algorithms or code for their matching engine and in turn many everyday people do not understand how matching engines work even though they are a crucial aspect of how the stock market works. I personally didn’t know how they worked until I was taught about it in this class. To solve this problem, I created two matching engines, one in the MatchingEngine.py file that is tested in the Test.py file, and another in the OrderBookGUI.py file that creates a GUI using the tkinter python library and implements all of the methods listed above. The MatchingEngine.py file implementation of stock matching engine uses binary heaps to store prices in bid/ask order books and balances the orders recursively. The value of order\_id reflects the order of arrival to the exchange, and the orders are executed in the priority of the order\_id’s. Orders with smaller order\_id values are given priority in execution, as these are the orders that arrived earlier. Both the buys\_prices and sells\_prices heaps are min-heaps, with the buys\_prices heap containing negative values of submitted buy prices. These heaps only contain unique entries, and when new order information is stored in the buys\_info and sells\_info at each price level, it is checked first to ensure that the price is not already present and to avoid duplicates. If the price is not already present, it is also added to the corresponding heap.

The matchOrders function is a recursive, FIFO function that fills all the orders in a class by adjusting the attributes buys\_filled and sells\_filled. It does this by first checking if the current best prices in the buy and sell books have zero volume and, if so, removes them from the heaps and order info books. It then checks if the books are balanced, which happens when either book is empty or the best buy price is lower than the best sell price. The function then takes the earliest orders from the best buy and sell prices, records their order IDs and sizes, and fills the smaller sized order. It executes the orders at a price that is the mean of the submitted limit price and the best limit price from the opposite order book, potentially executing each order in different portions at different sizes. It records the prices and sizes in buys\_filled and sells\_filled and removes the filled volume from the total volumes of the buy and sell books, as well as subtracting the filled size from the orders' volumes. It also cleans up any orders that have been completely filled. The function continues to call itself recursively until all orders have been filled.

The Test.py file tests the MatchingEngine.py file by running a changeable number of iterations adding either a buy or sell order to the orderbook. It then prints out the randomly generated order to be added to the order book, whether certain orders were filled, and then the buy and sell orderbooks. Example outputs have been provided below with using 100 iterations. The first output is after the first order, and the second output is after a few orders have been added:

Adding order, side=B, price=101.1, size=24

Buys filled:

{}

Sells filled:

{}

The buy book:

[-101.1]

{101.1: {'ids': [0], 'sizes': {0: 24}, 'tot': 24}}

The sell book:

[]

{}

Adding order, side=S, price=102.9, size=104

Buys filled:

{3: [[102, 103.05]]}

Sells filled:

{4: [[102, 103.05]]}

The buy book:

[-101.1, -98.9, -93.7]

{101.1: {'ids': [0], 'sizes': {0: 24}, 'tot': 24}, 98.9: {'ids': [1], 'sizes': {1: 170}, 'tot': 170}, 93.7: {'ids': [2], 'sizes': {2: 46}, 'tot': 46}}

The sell book:

[102.9]

{102.9: {'ids': [4], 'sizes': {4: 2}, 'tot': 2}}

The Test.py file also prints out a histogram of the balanced buy/sell order books as well as a bid/ask prices time series. Both of them are attached below for the 100-iteration test.

![Chart, histogram

Description automatically generated]()![Chart

Description automatically generated]()

Additionally, I also made an orderbook/matching engine GUI in the OrderBookGUI.py file. The order class has the following attributes: price, quantity, customerID, and timelimit. The purpose of this file was the solve the program of having an interactive GUI for a matching engine, unlike my other matching engine that is used to demonstrate how matching engines work and run tests on it via the Test.py file. For the OrderBook class, I implemented the FIFO algorithm (also implemented in the MatchingEngine.py file), the pro rata algorithm, the most satisfied customers algorithm, and it also tracks the amount of time that a customer order takes to be filled and is able to cancel the order if the time limit is reached. The GUI for the file is attached below:

Graphical user interface, application

Description automatically generated

As you can see, you can enter a customerID, whether it is a buy or a sell order, the quantity, and the price and it is added into the orderbook. I did run out of time when making the GUI, as I wanted to ensure most of my time was spent properly implementing an example matching engine and implementing all of the algorithms listed above. The biggest limitation of my project would be the GUI, and that is something that I will be working on over break. I also will be adding functionality that allows a user/customer/employee at a stock exchange to add order for specific real life stocks in real time.

My project is related to the class, as we both talked about matching engines and how they work, and it was also listed on the example projects list. I wanted to choose a matching engine as I knew that I would most likely be the only person to choose it and it would therefore be a unique and interesting challenge. I spent a very long time both creating the matching engine that can be tested on as well as having to implement all of the different algorithms that I listed above. I spent way more than 1.5x effort on this project and am very proud of the amount of work that I was able to finish. As previously mentioned, I would love to be able to work on the GUI more and also add functionality to implement real life stocks into the matching engine. I would also like to be able to allow the user to enter in which algorithm they would like to see used into the GUI so that they could compare the results and better understand how matching engines work, which was the original problem I was attempting to solve.

I really enjoyed this class! I was definitely pushed and learned a lot. Thanks for a great semester!