

Bitcoin in the Big Data Era

# Homework #2 (Option D: Order book)

## Electronic Order Book for Ethereum

---



### Abstract

Conventionally, an order book is a list of orders that is maintained by a stock exchange to record the interests of buyers and sellers. As such, only the market maker can determine if each of the interests are fulfilled by by matching the interest of individual buyers and sellers. An electronic order book is not different from a conventional order book. it performs identical tasks with the aid of a backend server.

**Keywords:** Electronic order book, Blockchain, Cryptocurrency

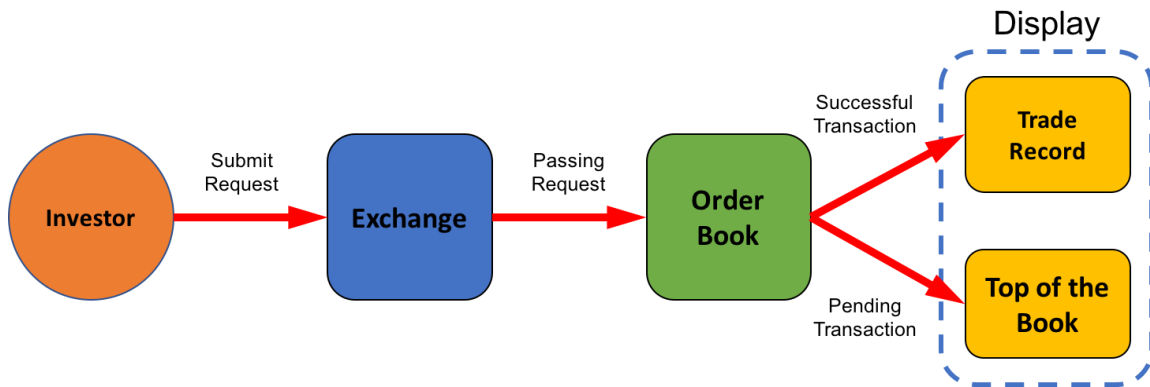
---

## Table of Content

<b>Abstract</b>	<b>1</b>
<b>Table of Content</b>	<b>2</b>
<b>Introduction</b>	<b>3</b>
Data Management	3
<b>Order Driven Model</b>	<b>4</b>
<b>Matching System</b>	<b>5</b>
Red-Black Tree	5
<b>Database</b>	<b>6</b>
Storing of Bidding Request	6
Storing of Asking Request	6
Fetching Data from Database back into RAM	7
<b>Current Price</b>	<b>7</b>
High and Low Price	7
<b>Future improvement</b>	<b>8</b>
Book Depth	8
The 90-Percent Confidence Interval High and Low Price	8
<b>Conclusion</b>	<b>9</b>
Contribution	9
Leader	9
Members	9
<b>Appendix</b>	<b>10</b>
Program Flow Diagram	10

## Introduction

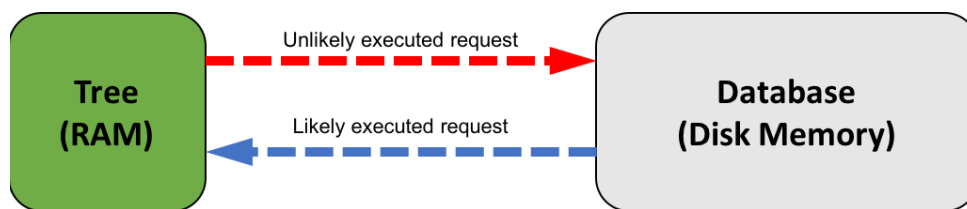
In our project, we aim to implement an electronic order book for Ethereum based on python(2.7.10) as the development platform. The implementation would be an electronic order driven model that receive trade request from investor. The order book will first order the trade request, then run through the ordered record to match the request. Finally, the top of the book and the latest transactions is published to the public(refer to **Figure 1**)



**Figure 1:** Information Flow Overview

## Data Management

The order book is designed to receive high volume of data throughout, hence, a database(SQLite3) is used alongside the implementation to facilitate the data influx. As the request is passed from the exchange into the order book, the order book compares the price of the new request and determines if the new request is likely to be executed. If the new request is unlikely to be executed, it will be stored in the database memory. Conversely, if the top of book request falls below a certain volume, the order book will fetch the most likely executed requests from the database into the top of book(refer to **Figure 2**)




**Figure 2:** Database to store unlikely executed request

## Order Driven Model

The implemented electronic order book is based on an order driven model, where all submitted events are ordered and displayed to the public. The order book is arranged in a manner so that the top of the book will record the highest-bid and lowest-asking price. This is in contrast to a quote driven market, where only the bid and ask price of the market makers are available to the public. Hence, the order driven market implementation empowers investors with information of prices that buyers/sellers are willing to accept.

As such, the order book will display 3 sets of information; asking price, bidding price, and the trade record of latest successful transactions. However, for academic purpose, we simplified the output to display only 10 records for each of the respective displays(refer to **Figure 3**)



ETH/USD:

6900.2366

HIGH

6933.5326

LOW

6898.7833

Bid Book

COUNT	AMOUNT	TOTAL	PRICE	PRICE	TOTAL	AMOUNT
5	1.1	89.8	6,892	6,929	236.6	4.3
2	7.8	97.7	6,891	6,930	480.5	243.8
2	2.2	99.9	6,890	6,931	482.6	2.1
1	0.0	100.0	6,889	6,932	491.1	8.4
3	9.1	109.1	6,888	6,933	491.5	0.4
1	2.0	111.1	6,887	6,934	491.5	0.0
2	4.3	115.5	6,885	6,935	549.1	57.5
4	2.7	2.7	6,915	6,936	554.5	5.4
2	1.0	3.7	6,914	6,910	7.0	7.0

Ask Book

COUNT	TIME	PRICE	AMOUNT
7	11:09:51	6,918.4	0.2246
30	11:09:51	6,918.2	0.01
4	11:09:51	6,918.3	0.121
3	11:09:51	6,918.4	2.2754
3	11:09:51	6,918.2	0.01
2	11:09:50	6,919.0	0.3667
15	11:09:50	6,918.4	0.0053
5	11:09:44	6,918.4	0.0147
7	11:09:42	6,918.6	0.025

**Figure 3:** Display of Ask, Bid, and Successful Transaction

## Matching System

The matching system integrates two values; the bidding price and the asking price, then it matches the highest bidding price with the lowest asking price. In this way, each transaction will provide the exchange with the largest bid-ask spread (refer to **Equation 1**)

$$\max\{Bid-Ask\ spread\} = \max\{Bid\ price\} - \min\{Ask\ price\} \quad (\text{Equation 1})$$

In the implementation, the matching of the submitted requests are facilitated using a Red-Black Tree to store and order the request before matching the highest bidding price with the lowest asking price.

### Red-Black Tree

The Red-Black Tree is a self balancing tree that will ensure that the minimal tree height with the insertion/deletion of new nodes in the tree, and guarantees search time within  $O(\log n)$ , where ' $n$ ' is the number of nodes in the tree.

Within the implementation, there are 2 trees, the Bid Tree and the Ask Tree. The Bid Tree stores only bidding request price, and the Ask Tree stores only the asking request price; each unique price would be inserted as a node in the tree. However, if a new request submits an existing price that already exist as a node in the tree, the new request will not create a separate node, instead the new request will contribute to the volume of request on the existing node. The trees will continue to populate and update nodes as requests are passed into the tree.

The order book will also match appropriate pair of ask and bid price to execute transaction. The execution of matching will draw the highest bidding price and lowest asking price from the respective tree to form a match.

However, storing of the request/transaction information on the Red-Black Tree takes up the Random-access Memory(RAM) of the hosting server and may potentially cause the hosting server to run out of memory overtime(or during high request influx). Hence, part of the requests have to be stored in a database to ensure sustainability of the implementation.

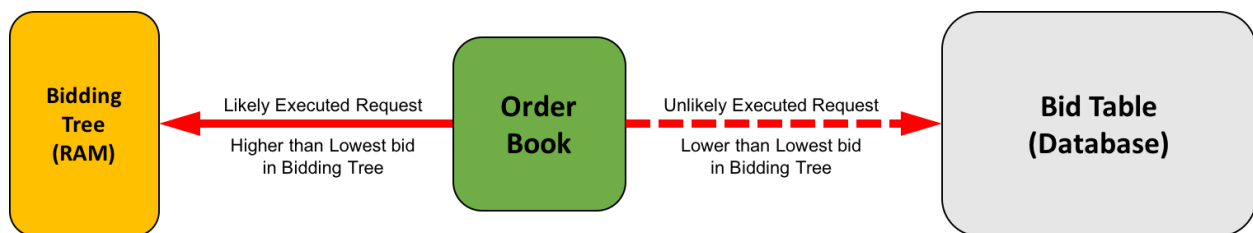
## Database

In contrast to the RAM which typically has very limited memory space(4 Gigabytes to 8 Gigabytes), storing data in the disk memory through the use of database allows the hosting server to be more sustainable in the big data era.

When a new request is passed into the implementation, if the the tree is highly-populated, a comparison with the pending requests on the tree will be done to determine whether the new request is likely be be executed. If the new request is likely to be executed, it shall be inserted into the tree(RAM), otherwise, it will be stored in the database.

### Storing of Bidding Request

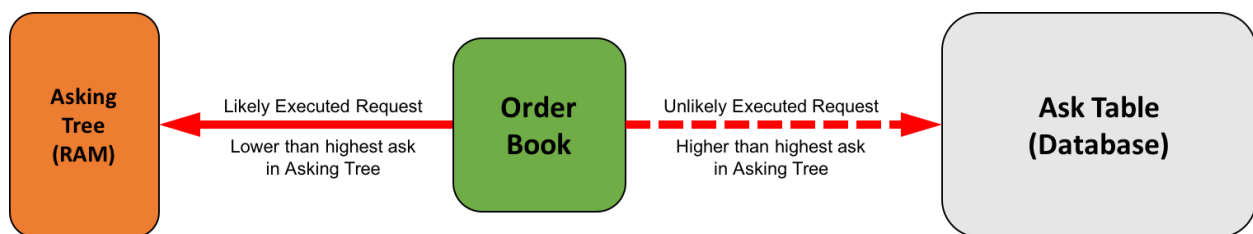
For a bidding request, it will be compared against the lowest bidding price on the bidding tree. If the price is higher(likely to be executed), the new request is inserted into the bidding tree(RAM), else, the new request will be stored in the database(refer to **Figure 4**)



**Figure 4:** Passing of new bidding request

### Storing of Asking Request

For an asking request, it will be compared against the highest asking price on the asking tree. If the price is lower(likely to be executed), the new request is inserted into the asking tree(RAM), else the new request will be stored in the database(refer to **Figure 5**)



**Figure 5:** Passing of new asking request

## Fetching Data from Database back into RAM

Due to the supply-and-demand of the free market environment of the cryptocurrency trading, the shift in the market price of Ethereum is largely motivated by imbalance of supply and demand in the market. As such, the nodes in the bid(ask) tree will be matched at a rate faster than the insertion rate.

Therefore, the depleting tree has to be replenished by fetching data from the database. For a bidding tree, the highest bid will be fetched from the bid table. Contrariwise, for an asking tree, the lowest bid will be fetched from the ask table.

## Current Price

The current price of Ethereum in the exchange would be determined by the historical mean of successful transaction within the last 24-hours(refer to **Equation 2**)

$$Current\ Price = \frac{\sum_i^{n_{24-hours}} Transaction_i}{n_{24-hours}} \quad (Equation\ 2)$$

## High and Low Price

The High and Low Price of Ethereum would be determined by the historical highest and lowest transacted price of the last 24-hours(refer to **Equation 3** and **Equation 4**)

$$High\ Price = \max \{Transaction_{24-hours}\} \quad (Equation\ 3)$$

$$Low\ Price = \min \{Transaction_{24-hours}\} \quad (Equation\ 4)$$

## **Future improvement**

The implementation of the order books has displayed features of an order book to facilitate requests/transactions in the exchange that includes, accepting request, matching request, and displaying the latest transactions/top-of-book(for both ask and bid).

However, there are additional considerations that could further enhance the order book that were not implemented due to the lack of technical knowledge in the field of cryptocurrency trading environment. In spite of the additional features proposed being uncomplicated to be implemented, the wheel should not be re-invented until further academic research is done to justify the addition of proposed features.

### **Book Depth**

The book dept could improve the implementation where the number of prices available in the electronic order book is fixed. Through fixing the book depth, the exchange can better make the market by limiting both the supply and demand, keeping the variation of the market within the capital of the market makers.

This feature further enhances the stability of a cryptocurrency like Ethereum that has a reputation of a stable cryptocurrency.

### **The 90-Percent Confidence Interval High and Low Price**

In addition, the implementation could give the investor an addition statistical referencing of 90% confidence interval, allowing the educated investors to better gauge the market trend. This is because each investor's' ask/bid request is believed to be independent, with high volume of request, and finite variances; by central limit theorem, we know that the sample shall follow an approximate of normal distribution. Hence, we could make use of readily available estimators to produce statistical interval estimate to for the investors.

The 90% confidence interval inform investors that if a bid(ask) request is submitted lower(higher) than the 90% high(low) price, the request is highly unlikely to be executed.



## Conclusion

In conclusion, the implementation provide investors with a convenient environment which allows them to track asking, bidding, and current prices in real time. The order driven model coupled with transparent matching system makes the electronic order book user-focused; allowing investors to make adjustment to their request in tandem with the market. The implemented matching system is highly transparent as it matches investors based on the highest bid-ask spread; this is advantageous for the investors as transactions are executed based on the requested price(the price they are willing to undertake) while maximizing the profits of the stock exchange.

On top of that, the Red-Black-Tree, in which matching is executed, minimized the search time, thus creating an efficient matching mechanism. Furthermore, to ensure the sustainability of the hosting server, a database(disk memory) is used in parallel to lighten the memory load on the RAM usage.

Refer to **Figure 1-A** in Appendix for the program flow infographics.

## Contribution

### Leader

Nicholas Yeow [T06902106] : Coding of Orderbook, Tree, Matching Function

### Members

Tan Kwan Fu [T06501101] : Report Writing, Co-coding of Database Function

Angela Yung [T06902101] : Co-coding of Database Function

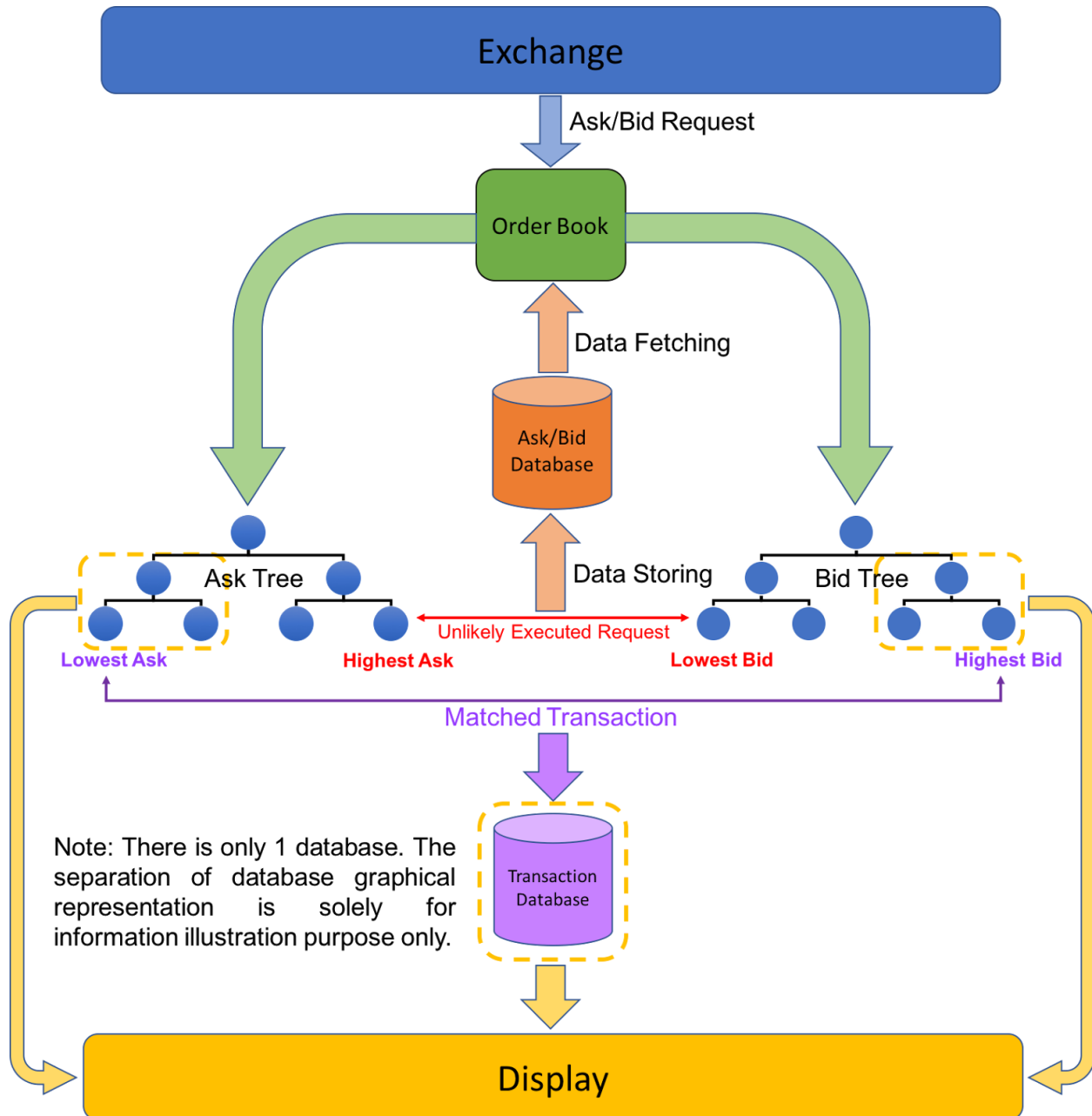
Yifan Li [T06901110] : Coding of Display Function

Deniz Sen [T06303125] : Report writing

Yao-Cheng Zhang [B04902047] : Researching, Report writing

## Appendix

### Program Flow Diagram



**Figure A-1:** Program Flow Infographics