Order Book Data Structures

John Jenq[1], **Priscilla Jenq**[2]
[1]Computer Science Department, Montclair State University, Montclair, NJ, USA
[2]Bank of America, New York, New York, USA

Abstract **-** *Order books are used by Exchanges to maintain sell and buy orders. In today's age, trading is done mostly over the internet and advancements in technology have rapidly increased the speed of the execution of orders. Orders arrive to stock exchanges like NYSE or NASDAQ in time increments as small as milliseconds or nanoseconds. E-trading is now done increasingly through trading algorithms. Trading may occur at a very high speed, and it is referred to as high frequency trading. Within seconds, a complicated algorithm needs to make the decision to buy, sell or hold stocks. Companies have reported profits of billions of dollars using sophisticated high frequency trading methodologies. In this report, we investigate data structures that can be used to implement order books and we determine whether they meet the speed requirement of a fast response time to handle orders.*

Keywords: Order book, short term Stock market prediction, high frequency trading, data structures

1 Introduction

A stock exchange is a platform that facilitates the exchange of stocks between buyers and sellers. In today's technological age, trading is done mostly over the Internet [1] and traders have mainly moved on from floor trading to *electronic trading* [3]. In fact, NASDAQ, the first electronic stock market in the world, is now the second largest stock exchange worldwide and, on average, trades more shares per day than any other U.S. market [2].

In the buying and selling of stocks, there are two different types of orders to consider: *market orders* and *limit orders*. A *market order* is executed at market value, while a *limit order* sets a specified price [9]. In this paper, we will only focus on limit orders, but note that the findings are also relevant to market orders, as they can be dealt with similarly. An order book [4] maintains orders from intended sellers and buyers. Strictly speaking, there are two books for each stock: one to maintain sellers' information and the other to maintain buyers' information. The books are maintained in such a way that the buyer who has the highest *bid price* will have the priority to buy if there is a match with an *ask price* on the sellers' side. In the case where there is more than one buyer with the same bid price, the time stamp will be used to resolve the conflict. The order which arrives earlier will be executed first. Similarly, the lowest sell price will put the seller at the top of the seller's order book. If there is more than one seller on the sellers' side, then the one with the earliest timestamp has the priority to sell. Thus, the order in which the prices arrive matters, as the price level with the earliest timestamp will have priority. The difference between these two top prices, the highest bid and lowest ask, is known as *spread*. If the difference between the last successful transaction price and the top selling price is smaller than the difference between the transaction price and the top buying price, this is called *seller's market*. A buyer's market is similarly defined, but with the converse being true. Figure 1 shows a seller's market.

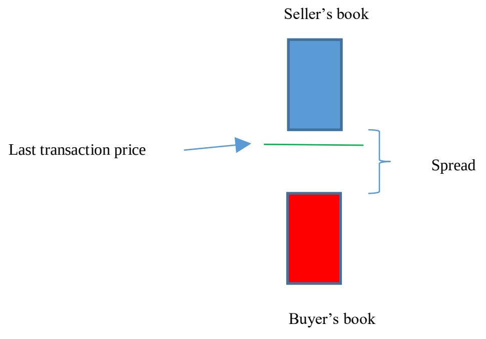

Figure 1. Order book spread and market trend

In this paper, we compare various data structures and discuss and analyze the pros and cons of each for implementing an order book. With different protocols, the requirements to manipulate the order book operations are different. We also post research questions about which data structures can facilitate the learning and predicting of the short term stock market.

2 Data structures for order book

The operations that we need to handle the business logic and to maintain an order book are: insertion, deletion, cancellation, and modification operations. The insertion operation allows a buy or sell order to be placed in the book. The deletion operation allows us to extract the top entry of a book so that we can execute the *transaction*, i.e., when the buy and sell price matches for a limit order. The deletion operation requires that we maintain a timestamp on each entry, where the entry with the earliest timestamp will deleted from the order book first. The timestamp can be represented as hour, minute, second and millisecond, etc. or it can be formatted as an unsigned integer that represents the total number of milliseconds or nanoseconds from midnight of a particular trade day. Note that it is possible that after execution of a transaction, a portion of the order may still remain in the order book. For example, if the quantity in the seller's book is greater than the quantity in the buyer's, and there are no other buyers for that buy price level, then the seller's quantity will be updated. It will still stay at the top of seller's book, but with the quantity modified to reflect the effects of the executed transaction. The cancellation operation allows an order to be removed from an order book based on its reference ID or price. Some FIX (Financial Information eXchange) protocols allow for simple cancellation using just the trade reference ID, which is unique for a particular trade day. Other messages will also provide the original price when doing the cancellation. If we implement the order book using a hash on the price level, then including price information in an order's cancel message is necessary, because without it, we cannot use hash to find the order.

Some FIX messages allow the modification operation, referred to as *cancel replace* in FIX terminology, to either stay in the same position in an order book or move to another position. This means that we may need to reorganize the order book after a modification message so that it will be placed in the right position. Again, it depends the data structure. If the data structure and algorithm don't maintain a strict order, then this is not an issue. However, if the data structure does maintain sorted order of some form, then the modified order will have to be moved to its proper place. Therefore, this is a point to consider when choosing a data structure.

To easily maintain sell and buy order prices, exchanges maintain *tick size* which "dictate the minimum standards at which the price of a particular security can move" [5]. In other words, a sell or buy order cannot have a random number of digits after the decimal point. For example, NYSE has a tick size of 1/16 (0.0625) dollars. This means that the price can only be incremented by multiples of 1/16. Thus, the prices of an order can be treated as discrete values.

2.1 Comparison of different data structures

In this section, we will investigate several data structures and analyze their time complexities according to their operations. We assume order prices are discrete but not continuous. For example, one can assume that one tick is 1/16 of a dollar. Note this value can be changed to some smaller value. All data structures support insert, delete, cancel, and modify operation of orders. We assume daily limits for high and low [8] so that the number of price levels of a particular stock or securities is bounded. But we will also briefly discuss the case when there are no daily limits enforced. Each FIX message contains a reference ID (*Ref-ID*) which is a unique ID used to identify a particular order. It is usually a combination of trading company ID and a unique sequence number generated by that company.

2.1.1 Method 1

We will allocate an array of pointers where each array element represents a price level. We can use price as the key to index into the order book. For each price level, we maintain a linked list. The head of linked list contains the order with the smallest timestamp. We can maintain a tail pointer to allow for fast insertion of new incoming orders to the tail of this list. Thus, it would take $O(1)$ time to insert. The deletion operation also takes $O(1)$ time because we delete an order from the head of a linked list. The complexity of cancelling of an order with the price information given depends on the length of the linked list of that price level. Cancellation of an order using reference ID (*Ref-ID*) alone will take $O(n)$ time because in the worst case, one may need to search all the entries of this data structure. Using price level information as a key, modifying an order with a new quantity while keeping the same timestamp has run time $O(n_i)$, where n_i is length of linked list in price level i. In the worst case the complexity will be $O(n)$. Similarly, for modification of an order with a new quantity and a new timestamp, the running time and worst case time would be the same as above. The only difference is that an extra deletion and insertion operation may be needed. The modification operation of requires one cancel operation, plus one insert operation, so the running time will be $O(n_i+1)$, where n_i is the length of the linked list of old price level. $O(1)$ is for insert into the new price level. Again, in the worst case, the time complexity is $O(n)$.

The complexities of these operation are summarized in **Table 1**. Note that other information may be maintained in this data structure like keeping track of how many distinct orders on a price level, the total number of shares in a given price level, etc.

The space complexity for this method would be $O(n+d)$, where n is the number of orders currently on the order book and d is the depth of price levels, which is the total number of possible prices. Because orders tend to cluster around certain price level ranges, the run time depends on the distribution of orders. If a buyer or seller sends many small orders with the same price in order to influence the market, this may fool others into falsely assuming that there are a lot of valid requests at a particular price level. These large quantity requests will cause search time to increase significantly. This is true especially since cancellation and modification operations will most likely follow, as these orders are not truly intended to be buy or sell requests, but rather as a means to influence the market. The other issue faced is that some stock might have very large price levels, like bitcoin [6], which in the year 2017 were a very hot stock and were actively traded. These large price levels may affect the run time of these operations.

Operation	Running Time complexity	Worst case
Insert	$O(1)$	$O(1)$
Delete	$O(1)$	$O(1)$
Cancel using price	$O(n_i)$, n_i is the length of chain in level i	$O(n)$
Cancel using *Ref-ID* alone	$O(n)$	$O(n)$
Modify quantity timestamp not changed	$O(n_i)$, n_i is length of chain in level i	$O(n)$
Modify quantity timestamp changed	$O(n_i+1)$, n_i is length of chain in level i	$O(n)$
Modify price	$O(n_i+1)$ one cancel + one insert	$O(n)$

Table 1. Operation complexity using data structure of method 1

The potential problem of this method is that if there are no daily limits set, and the market goes wild, there could be a large difference between the top and bottom order prices. One way to efficiently address this problem is to increase the table size. Another solution is to maintain a price level table with a maximum top price limit and minimum bottom price limit. For the order prices that fall outside the boundaries of these upper and lower limits, an extra array can be set up to hold these outlier orders. However, the problem with this solution is that the insert and delete operations will not be constant any longer.

2.1.2 Method 2

In method one, the most inefficient operation is cancelling with reference ID alone, which takes $O(n)$ time, because the proposed data structure does not track the reference ID. In the worst case scenario, one has to search every element to find the order. The second method investigated here is similar to method one, but in addition to maintaining a linked list, we will maintain a hash table for reference ID. This will improve the runtime of the cancellation operation using reference ID. The key of the hash function is the reference ID and the value is the address of the node in the order list. Although the worst case complexity remains the same, the average expected running time will be reduced because we are searching on a hash table. The average time complexity will be improved, provided that we use a good uniform hash function. The space complexity depends on the size of the hash table. If we can maintain the size of the hash table to be below 50% full (i.e., alpha value is 50%) with open addressing hashing, then the expected value of the number of probes is $E[probes] = \frac{1}{1-\propto} = \frac{1}{1-0.5} = 2$, which is constant time although the worst case complexity is still $O(n)$. Recall that d is defined as the depth of price levels and let s be the size of the *Ref-ID* hash table. The space complexity is $O(n+d+s)$. A summary of time complexity is shown in **Table 2**.

Due to the dynamic nature of stock trading and the requirement of an efficient implementation of trading operations, it is crucial to determine the size of the reference ID hash table. One of the problems of hashing with open addressing is that when the table is almost full, the time needed to search increases significantly. When the alpha value of the table exceeds 0.95, for example, the time to search an order will take longer due to a higher chance of collisions. The common practice to remedy this problem is to enlarge the hash table and rehash all the entries of the old table into the new table. However, the cost of the rehash and copy operations would be too expensive for order book handling. This is due to the fact that our application requires each order book operation to be processed in a timely fashion, which means that a long rehash and copy time to reorganize the hash table is not acceptable. Therefore, we need to predict the maximum number of orders that may be in the system at any time.

Operation	Running Time complexity	Worst case	Average case
Insert	*O(1)*	*O(n)*	*O(1)*
Delete	*O(1)*	*O(n)*	*O(1)*
Cancel using price	*O(n_i)* n_i is the length of chain in level *i*	*O(n)*	*O(n/levels)*
Cancel using *Ref-ID* only	*O(n)*	*O(n)*	*O(1)*, ∝< 0.5
Modify quantity timestamp not changed	*O(n_i)* n_i is the length of chain in level *i*	*O(n)*	*O(n/levels)*
Modify quantity timestamp changed	*O(n_i+ 1)*	*O(n)*	*O(n/levels)*
Modify price	*O(n_i+ 1)* one cancel + one insert	*O(n)*	*O(n/levels)*

Table 2. Operation complexity by maintaining *Ref-ID* using hashing with open addressing

2.1.3 Method 3

Another way to improve the efficiency of cancelling with reference ID—without worrying about the impact of the hash table alpha value and the intolerable cost of expanding the hash table's size—is to implement the reference ID hash table using hashing with linked list (chaining). The data structure is shown in Figure 2.

The complexity of this method's data structure to support order book operations is shown in **Table 3**. Although the worst case complexity for the cancel and modification operations remain *O(n)*, the run time complexity has significantly reduced.

Recall the definitions of *n*, *d* and *s* from the previous sections. The space required for this method would be *O(n+d+s)*, which is the same as the previous method. Instead of letting *d* be the size of the price array, some may be tempted to believe that a hash table of a predetermined size would be a better alternative to the price array for handling orders. Each order would be hashed into the price hash table using its price as the key, and if collisions occur, then chaining can be used. This approach can easily solve the problem that occurs when there are no daily limits set, or when the range of maximum and minimum price levels is large, or when there are outliers. However, the problem with this approach is that the insertion and deletion operations are not *O(1)* any longer. The reason is because an entry in the price array may contain orders of more than one price.

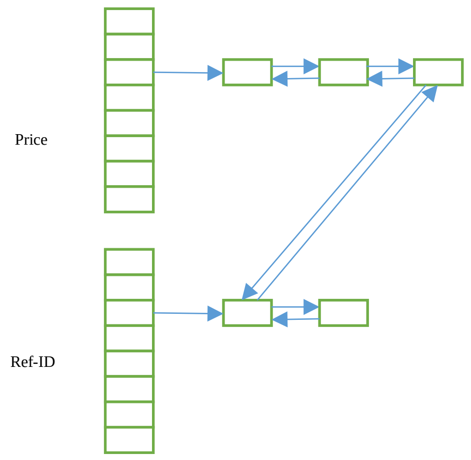

Figure 2 Maintain Reference ID using hashing with chaining

Operation	Running Time complexity	Worst case
Insert	*O(1)*	*O(1)*
Delete	*O(1)*	*O(1)*
Cancel using price	*O(n_i)* n_i :length of chain in price level *i*	*O(n)*
Cancel using *Ref-ID* only	*O(n_j)* n_j is the length of chain in level *j* of *Ref-ID* hash table	*O(n)*
Modify quantity timestamp not changed	*O(n_i)* n_i is the length of chain in price level *i*	*O(n)*
Modify quantity timestamp changed	*O(n_i+1)* n_i is the length of chain in price level *i*	*O(n)*
Modify price	*O(n_i+1)* : one cancel + one insert	*O(n)*

Table 3. Operation complexity by maintain *Ref-ID* using hash with chaining

2.1.4 Method 4

Another way to improve the cost of the cancellation with reference ID without worrying about the hash table *alpha* value can be done by replacing the hash table of the reference ID with a balanced search tree. A balanced search tree such as an AVL tree or B tree (a generalized binary search tree that self-balances) has time complexity of *O(logn)*. This method is similar to the method 3, but will create and maintain a balanced search tree data structure to maintain reference IDs. This search tree uses the reference ID as its key on the tree node. Each tree node contains a pointer to the linked list of the price table. Based on this set up, the insertion operation will become *O(logn)* rather the original *O(1)*. Similarly, the time complexity for deletion of the top order will also become *O(logn)*.

Operation	Running Time complexity	Worst case
Insert	*O(logn)*	*O(logn)*
Delete	*O(logn)*	*O(logn)*
Cancel using price	$O(logn+n_i)$ n_i is the length of chain in price level *i*	*O(logn)*
Cancel using *Ref-ID* only	*O(logn)*	*O(logn)*
Modify quantity timestamp not changed	$O(n_i)$ or *O(logn)* whichever is smaller n_i is the length of chain in level *i*	*O(logn)*
Modify quantity timestamp changed	$O(n_i)$ or *O(logn)* whichever is smaller n_i is the length of chain in level *i*	*O(logn)*
Modify price	$O(n_i)$ or *O(logn)* whichever is smaller n_i is the length of chain in level *i*	*O(logn)*

Table 4. Operation complexity by maintain *Ref-ID* using hash with balanced search tree

The cancel operation using price becomes $O(logn+n_i)$, where n_i is length of chain in price level *i* and *n* is the number of orders on the order book. Cancel with reference ID would become O(*logn*). Modifying the quantity without changing the timestamp will be $O(n_i)$ and modifying the quantity with the timestamp changed will be the smaller of $O(n_i)$ or *O(logn)*, provided that we maintain a combine count field, denoted as *ccn*, on the price table, where the *ccn* stores the total number of distinct orders on that price level. Comparing *ccn* with *log n*, where *n* is the total number of orders on the order book, we can then choose to search either the price level linked list or the balanced tree to find the order. So the worst case complexity will be just *O(logn)*, as shown in **Table 4**.

If both the reference ID and price for an order are presented in the message of the cancel order, then the complexity of cancellation will be the smaller of $O(n_i)$ or *O(logn)*. Thus, improving the modification and cancellation operations causes insertion and deletion from the top of list to slow down from *O(1)* to *O(logn)*.

2.2 Challenge to speed up further

Can we further improve the time complexity of the operations? In financial applications, speed is our major concern. Thus, we must look for ways to improve the operation speed without worrying about the cost of memory. Recall that Method 1 has the best time complexity for inserting and deleting from head of the linked list. Can we keep the runtime constant, while supporting other operations in *log(n)* time or better?

Let the price array be the same as in Method 1. We will change *Ref-ID* structure. Since we ignore the efficiency of memory usage and only care about speed, we can use an array to maintain all sender company IDs. Each element of this company array represents a potential sender company and will point to a table that stores the orders from that company. Note that the order is represented as a sequence number. The data structure is shown in Figure 3. Here, *CompID* is the sender company ID and *Seq#* is the sequence number generated by that company.

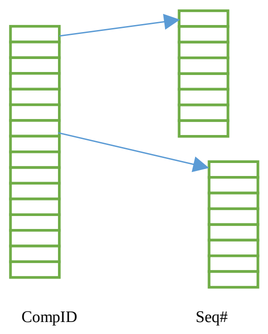

Figure 3. Partition *Ref-ID* into Company ID and Sequence Number

For example, if *CompID* has five digits and the sequence numbers of a session has 4 digits, then the size of the data structure to maintain Reference ID will have one billion entries. Assuming that the total number of company stock traded on this exchange is 3,000, then the total amount of memory needed is in the order of a trillion. If such a data structure is feasible, then the time complexity of the insert, delete, modify, and cancel operations will be *O(1)*, as outlined in Table 5.

Operation	Running Time complexity	Worst case
Insert	*O(1)*	*O(1)*
Delete	*O(1)*	*O(1)*
Cancel using price	*O(1)*	*O(1)*
Cancel using *Ref-ID* only	*O(1)*	*O(1)*
Modify quantity timestamp not changed	*O(1)*	*O(1)*
Modify quantity timestamp changed	*O(1)*	*O(1)*
Modify price	*O(1)*	*O(1)*

Table 5. If space complexity is not an issue

By reducing the sequence table to a factor of 10, the required space size becomes 300 Giga. In order to find the right entry on the sequence hash table, the required operation is $\frac{Seq\#}{10}$. Collisions can be handled either via open addressing or chaining, but chaining is a better choice, as explained in previous sections. If we choose the factor to be a power of two, the hashing can be easily done by shifting bits to the right to find the entry on the hash table.

3 Remarks

Currently, there are many different financial information exchange protocols [10] and they are continuously evolving. All new and updated protocols are published by exchanges. The purpose of these new or updated protocols is to reduce the communication time. Fairness to customers is also an important factor to consider. The basic messages sent by these financial information exchange engines, besides admin messages, are mainly to place orders, modify orders, replace orders, and remove orders in order book. We understand the order book implementations depends on the underlying communication protocol. An interesting research question is: can one propose a new protocol that is better than the current known protocols? It should reduce the bandwidth, and at the same time, better use the current technologies and help maintain an order book more efficiently. Another interesting question is: how do we use the shape and distribution of order books to predict the short term market? In other words, in addition to efficiently maintaining an order book, how can we embed information inferred from historical data into our order book data structure and also quickly retrieve it and use it to make better decisions in our trading algorithm? In order to facilitate analysis of the order books, we need to gather more information from this time series data, such as the last execution price and time when it was executed, and average execution price for the last second. Questions to consider would be: how many cancellations happened for each level over the last second? How many modifications occurred on each level over the last second? When did bursts of orders start occurring? What is the spread change of the book? How many orders were executed? What is the moving average of these orders? This information can be used to feed in machine learning programs such as ANN [7] simultaneously. Another interesting research topic is to investigate how to adopt parallel and distributed processing in the order book processing. For example, rather than increasing the amount of memory, how much can we speed up the execution of operations if parallel computations are involved? We know from the discussion in the previous section that the cost of trying to achieve a constant time complexity is expensive and most of the memory may be sitting idly and will rarely be used. Thus, the challenge is: can we perform operations in *O(1)* time using reasonable memory while multiple processors are employed?

4 Conclusions

In this report, we investigated and compared various data structures that can be used in the implementation of order book. If we need fast insertion and extraction of orders from an order book, using an array to store pointers to same price level orders may be a good choice. If the depth of the book is very high, i.e, the price range is large and there are a lot of orders, data structures such as a balanced tree may be better choice.

5 References

[1] https://en.wikipedia.org/wiki/Electronic_trading , Apr. 18, 2018
[2] https://en.wikipedia.org/wiki/NASDAQ , Apr. 18, 2018
[3] Jain, Pankaj K., 2005, “Financial market design and the equity premium: Electronic vs. floor trading,” Journal of Finance volume 60, issue 6, pp. 2955–2985. Also on

https://pdfs.semanticscholar.org/72b3/d51fe1db856c5146d0500d901ac1aacdadbf.pdf , Apr. 20, 2018
[4] https://en.wikipedia.org/wiki/Order_book_(trading) , Apr. 20, 2018
[5] https://www.investopedia.com/terms/t/tick-size.asp , Apr. 19, 2018
[6] https://dc-charts.com/depth_btc.php?ex=2&cu=0 , Apr. 20, 2018
[7] Priscilla Jenq and John Jenq, "Parallel Prediction of Stock Volatility", Vol 15, number 5, 2017, pp 70-73.
[8] https://corporatefinanceinstitute.com/resources/knowledge/trading-investing/daily-trading-limit/ , Apr. 20, 2018
[9] https://www.investopedia.com/ask/answers/100314/whats-difference-between-market-order-and-limit-order.asp , Apr. 18, 2018

[10] https://en.wikipedia.org/wiki/List_of_electronic_trading_protocols, Apr. 20, 2018