

Computational Analysis of Greedy and Hash-Based Order Matching Algorithms for High-Frequency Trading

Dr.P.Yoganandhini^{1*}, I.Blessy¹

Department of Computer Science, Vellalar College for Women (Autonomous), Erode, India

*Corresponding Author

DOI: <https://doi.org/10.51583/IJLTEMAS.2026.150900125>

Received: 25 September 2026; Accepted: 30 September 2026; Published: 19 October 2026

ABSTRACT

Fast and reliable trading is crucial in the world of High Frequency Trading (HFT) where automated systems make multiple trades in mere milliseconds. Order-matching techniques may perform differently when applied to different sets of data and different algorithms that process the data. In this study a Data Science based HFT Order Matching Engine is developed in MATLAB which is able to numerically process and compare two approaches (Greedy and Hash Based) for order matching on the same trading dataset. Based on price and quantity of buy and sell order, 3,000 order records were selected and preprocessed. Both algorithms were tried with the same matching conditions: an order is compatible if its buy price is no less than the price of the sell order, and the quantity of the matched order is the smallest available for the buy order and/or the sell order. Performance metrics of the algorithms included execution time, memory consumption, throughput, average matching time, number of matches and efficiency. A mathematical modelling based performance dashboard has also been created for the presentation of the experimental results in graphical and tabular form. Both algorithms generated 1420 matches and an equal number of matchable items 5207.978 and efficiency 49.26%. The Greedy algorithm took 0.256752 seconds to run, performing 11684.2 orders per second and an average matching time of 0.00018081 seconds per match. The Hash Based algorithm took 2.751413 seconds, 1090.32 orders/s, and had an average matching time of 0.00193761 seconds. Based on these results, in the experimental environment and the tested data set, the Greedy approach took less execution time, memory, and average matching time, and had greater throughput. The study emphasizes the key role of computational performance analysis for assessing order matching methods for HFT applications.

Keywords: High-Frequency Trading, Order Matching Engine, Data Science, Greedy Algorithm, Hash-Based Algorithm, Order Book, MATLAB, Performance Analysis.

INTRODUCTION

High frequency trading (HFT) is an electronic trading method that trades a multitude of financial orders within a very short span of time. As automation is gaining importance in trading systems, the need for quick order processing, quick decision making and reliable trade execution is high. The order matching engine is the heart of an electronic trading system, where it takes buy and sell orders, and decides if suitable buy and sell orders are available to be executed based on the given set of trading rules. The effectiveness of the matching process can have a direct impact on computational performance and responsiveness of an HFT system. Therefore, the selection of suitable order matching algorithm and its application to the system are important for systems having a large number of orders in time-sensitive applications.

Incoming buy and sell orders are typically kept in an order matching engine that determines whether or not there are trades that can be made based on the price and quantity of the buy/sell orders. In this study a trade is only valid if the purchase price is greater than or equal to the price that is sold. The volume of shares to be traded for a valid trade is determined by the minimum number of purchases/sales available. If the quantity is

zero, the orders are not processed further, else quantity is subtracted from the associated orders and if the quantity is zero after the subtraction, the order is not further processed. These may seem simple operations but when the number of orders grows, the computational price can be high. Therefore, the organization and searching method of algorithm is a very important factor in the time of execution, memory usage and total throughput. In fact, there are a few methods of order matching algorithms.

In a Greedy approach, the orders are processed sequentially and it will attempt to discover the possible matches by comparing directly the price compatibility of the orders. It is relatively simple in structure, so that it is easy to be computationally analyzed and implemented. The Hash-based approach is on the contrary based on price-based keys to enable structured ordering and matching the order information. With such an organization, a certain amount of the duplicate searches can be avoided, but other operations such as updating and accessing the hash-based structure could impact computational speed. Whether an algorithm performs well or not is dependent not only on its conceptual searching algorithm, but also on having to demonstrate it in a real-world scenario with the same conditions.

Prior studies have considered algorithmic trading and order book processing, the market microstructure and computational issues of improving the execution of trades. It is useful to compare the Greedy and Hash-based matching approaches based on the same dataset for the orders, and the same set of matching conditions, when considering the case of the same orders. The time taken for execution, memory consumption and throughput give a more complete picture of the performance of an algorithm than any one of these metrics alone. The importance of data processing and experimental conditions is also significant as the structure of the data or the matching rules can have an impact on the conclusions to be drawn.

This computational analysis study aims to come up with a Data Science-based Trade Order Matching Engine in MATLAB. A realistic order dataset containing 3,000 records, was used for the experiment. In both algorithms, the same dataset was used and the price compatibility condition and quantity-matching rule were the same to make a controlled comparison. The tests were conducted on the base of total matches, the number of matched order, execution time, memory usage, throughput, average matching time and efficiency. The following experimental performance dashboard was also developed to graphically and tabularly present the results in MATLAB. The purpose of this study is to provide a computational structured comparison of two order matching methods and to explore the processing strategies of the algorithms internal to them, and the impact on the performance of an order matching system for an HFT.

Research Gap

Previous research has examined high-frequency trading, limit-order-book dynamics, market microstructure, and the computational requirements of electronic trading systems. These studies establish the importance of low-latency processing and efficient order-book operations. However, the computational behavior of different order-matching implementations can also depend on the data structures, searching procedures, and implementation environment used. In particular, a direct experimental comparison of Greedy and Hash-Based matching implementations under identical order data and matching conditions provides a useful way to distinguish differences caused by the matching strategy from those caused by implementation overhead. Therefore, this study focuses on a controlled computational comparison of the two implemented approaches using MATLAB, with execution time, memory consumption, throughput, matching output, and matching efficiency as the principal evaluation measures.

Data And Setup

The data set used in this study is taken from the Trade and Order Book data on Kaggle which are publicly available with the specified url: <https://www.kaggle.com/datasets/farhankardan/trade-and-order-book>. The data comprises financial market trade and order book data, which can be used to analyse order processing and order matching in an HFT setting. In this study, 3000 of these order records were extracted from the original data and cleaned up. The four attributes of interest to the analysis are namely price, volume, order side and order identification information are indicated in Table 1. Volume attribute was considered as the

quantity ordered and the difference between Buy/Sell order was carried out by using order side information.

Table 1. A detailed field description of attributes of the dataset is provided. Each order has a unique identifier that can be used for identification and tracking in the matching process. The type of order can be of buy or sell. The price of the order; used to check the compatibility of the price of the buy and sell orders. Volume/Quantity is the quantity of a product that is available for trading. In this project, the Volume field will be considered as the order quantity.

Table 1. Description of the Dataset Attributes

Column	Description
Order_ID	A unique identifier assigned to each order for identification and tracking during the matching process.
Order_Type/side	Indicates whether the order is a BUY or SELL order.
Price	Represents the price associated with the order and is used to determine price compatibility between buy and sell orders.
Volume/Quantity	Represents the quantity available for trading. In this project, the Volume field is treated as the order quantity.

METHODOLOGY

The proposed method is to build and implement an HFT Order Matching Engine on MATLAB. The system is designed to perform a set of orders to buy and sell and to search for matching orders within the system according to price and quantity. Efficiency of the matching algorithm is important for high-frequency trading systems, which make numerous trades in a very short period of time. As a first step of the proposed approach order Data Set is prepared. Each order has information such as Order ID, Price, Volume/Quantity and Order Side. Orders are separated into buy and sell orders and are put into an order book. Matching is performed in two different methods: Greedy Matching Algorithm, and Hash Based Matching Algorithm. Both algorithms are computed using the same dataset and same conditions for making a fair computational comparison. After match, the execution time and memory usage of both algorithms are analysed, as well as their throughput, the average time to match, the number of matches and the efficiency of the algorithm. These results are then used to determine which of the two is the more efficient, and to match orders at HFT.

Dataset Acquisition and Preprocessing

The first process is acquiring the datasets and pre-processing them. Acquiring of the datasets and pre-processing. The needed records and attributes are chosen during the preprocessing and the data is formatted in a uniform structure that is appropriate for processing in MATLAB. Buy and sell orders are identified using the order-type information. This provides the following data set: 1385 of buy orders and 1615 of sell orders for a total of 3,000 orders. The same preprocessed dataset is provided to both algorithms to ensure they are operating with the same input data.

Order Representation and Order Book Construction

Order representation and Order book construction contains the orders from the dataset. Each order is pre-processed and represented with the order identifier, type, price and quantity. The orders are divided into two groups: buy orders and sell orders. Buy orders are requests to buy an asset and sell orders are requests to sell an asset. In Greedy approach, buy orders are sorted in descending order of prices and selling orders are sorted in ascending order of prices. This way, the algorithm can process the most suitable price it can find first. The Hash-Based approach sorts orders by price, allowing to find similar price levels with price-based lookup. The order book is the work structure for the matching engine. The quantity of an order that is available for matching is adjusted with each successful transaction during matching. Once the remaining quantity of an order is depleted, the order will be eliminated from the active order book.

Order matching rule,

buy order and sell order are compatible if the buy price is equal or higher than the sell price:

$$\text{Buy Price} \geq \text{Sell Price} \quad (1)$$

If the orders are compatible, then the number of items that are filled is based on the minimum quantity on the two orders:

$$\text{Matched quantity} = \min(\text{Buy quantity, sell quantity}) \quad (2)$$

(1) checks if a buy and sell order may be matched and (2) makes sure that the minimum of quantity from the buy and sell quantity of the matched order is selected.

Greedy Order Matching Technique

An order matching approach that is greedy. The Greedy technique orders matching by considering the orders available one by one. The orders are sorted by price conditions and during the scanning process, compatible orders are chosen. The Greedy algorithm checks buy and sell orders in the proposed system based on the price compatibility condition specified in (1). If the price of the buy order is at or above the price of the sell order, then the two orders are deemed to be “matched”. The quantity to be executed is calculated with (2) which takes the smallest quantity of the buy and sell orders available. Once a match, any orders that are left are updated and matching continues until all available orders are either matched or have no remaining matches. The Greedy technique is relatively easy to implement and process in a sequential manner. It has an execution time that is lower than the Hash-based approach and higher throughput for the experimental data set used in this project. The current implementation primarily uses price ordering. If multiple orders have the same price, their processing order follows the ordering produced by the implemented sorting procedure. Because the present dataset and implementation do not explicitly establish a price-time-priority rule, the results should not be interpreted as a complete implementation of exchange-level price-time priority.

Hash-Based Order Matching Technique

The Hash-Based matching implementation organizes orders using price-based keys to support structured lookup during the matching process. BUY and SELL orders are separated before matching, and the price information is used to organize the corresponding order records. For each order, the implementation stores the information required for matching, including the order identifier, side, price, and remaining quantity. Orders having the same price level are handled within the same price key according to the data structure used by the MATLAB implementation. When a potential match is evaluated, the algorithm identifies a BUY and SELL price pair satisfying the (1) condition and For every compatible pair, the matched quantity is calculated by (2) condition. The remaining quantity of both orders is then updated. When an order reaches zero remaining quantity, it is removed from further matching; otherwise, it remains available for subsequent matches. The matching process continues until no compatible BUY and SELL orders remain.

High-Frequency Trading Order Matching

The input for HFT Order Matching Engine proposed is a set of n orders represented by:

$$S = \{s_1, s_2, \dots, s_n\} \quad (3)$$

Every order includes parameters that are crucial for the order, including Order ID, Price, Quantity/Volume and Side, the latter being Buy/Sell. The entire order set is broken into two categories:

$$S = B \cup A \quad (4)$$

where B is the set of Buy orders and A is the set of Sell orders. Matching takes place with the aim of finding

the right orders as a result of their prices. If there are any prices available that are willing to buy, then a Buy order can match a Sell order if there are enough buying prices that can be matched with the price. The amount of the match is calculated based on the smaller quantity of the two orders available for the match, for each successful match. The quantities are subsequently adjusted. A fully-realized order is taken off any further processing, while a partially matched order will be kept for the next transactions. This matching process will continue until the last Buy/Sell order is matched. The trade information, matched quantity and unexecuted orders generated in this step are used to measure the performance of the proposed matching algorithms.

High-Frequency Trading is a digitised trading method where a lot of financial orders are executed in a very brief time. Typically, an order will include an Order ID, Side (Buy or Sell), Price and Volume. Let O be the set of orders represented as:

$$O = \{O_1, O_2, O_3, \dots, O_n\} \quad (5)$$

where each order is represented as:

$$O_i = (ID_i, S_i, P_i, V_i)$$

where:

- The unique identifier of the order will be used as the ID_i .
- S_i = side of the order, Buy/Sell is based on the order definition in the dataset
- P_i = price of the order
- V_i = Volume/Order size of the i^{th} order
- n = Number of orders

The entire order set can be broken up into buy/sell orders as:

$$B = \{O_i \mid S_i = \text{Buy}\} \quad (6)$$

$$S = \{O_i \mid S_i = \text{Sell}\} \quad (7)$$

If the prices of a buy order and a sell order meet the matching requirement, they will be matched. Under a traditional price crossing situation, the following rules apply:

$$P_b \geq P_s \quad (8)$$

Here P_b is the purchase price and P_s is the sales price. The matched quantity for two compatible orders is equal to:

$$Q_m = \min(V_b, V_s) \quad (9)$$

The remaining quantities are adjusted after a match as follows:

$$V_b' = V_b - Q_m \quad (10)$$

$$V_s' = V_s - Q_m \quad (11)$$

The matching continues until no more matching buy/sell orders exist.

Computational Analysis of Matching Algorithms

The following performance measures are used in the proposed project.

Execution Time

The execution time is required time that an algorithm takes to process the input data set and complete the matching process.

$$\text{Execution Time} = T(\text{end}) - T(\text{start}) \quad (12)$$

The smaller the time taken to execute the program, the faster it can be processed.

Memory Usage

Memory usage is the size of memory allocated by the algorithm and its data structure, while running. Reducing the amount of memory used is good when the system has to handle a lot of market data.

$$\text{Memory Used} = \text{sum}([\text{mem_info.bytes}]) / (1024^2) \quad (13)$$

Throughput

Throughput represents the number of orders processed by the matching algorithm per unit of execution time.

It is calculated as:

$$\text{Throughput} = \text{Total Orders} / \text{Execution Time} \quad (14)$$

In the project the unit used is orders/second.

Suppose, for instance, 3,000 orders are handled in 0.5 seconds:

$$\text{Throughput} = 3000 / 0.5 = 6000$$

So, about 6,000 orders are handled per second by the system.

Average Matching Time

Average matching time is the average of the computational time that it takes to make each successful match:

$$\text{Average Matching Time} = \text{Execution Time} / \text{Total Matches} \quad (15)$$

The lower it is, the quicker it processes with each match.

Matched Quantity

Matched quantity is the volume of orders that matched successfully between orders from buyers and sellers compatible to each other.

$$\text{Matched Quantity} = \sum Q_{\text{match}} \quad (16)$$

This is particularly significant because if two algorithms have matching rules that are the same, they should give similar matching results.

Total Matches

Total matches are the number of buy-sell from the algorithm.

$$\text{Total Matches} = \sum I_k \quad (17)$$

Where:

- If a valid match for buy–sell occurs, then $I_k=1$.
- If no valid match is found, then $I_k=0$.

Matching Efficiency

The efficiency is defined as:

$$\text{Efficiency} = (\text{Matched Quantity} / \text{Maximum Matchable Quantity}) \times 100 \quad (18)$$

where :

$$\text{Maximum Matchable Quantity} = \min(\text{totalBuyQuantity}, \text{totalSellQuantity})$$

Matched Quantity represents the total quantity successfully executed by the matching algorithm and Maximum Matchable Quantity represents the maximum quantity that can be matched under the defined order compatibility and quantity constraints. This metric is used to evaluate the completeness of the matching output under the specified experimental rules.

RESULTS

Both the Greedy order matching algorithm and the Hash-Based order matching algorithm were tested with the same data set that comprised of 3,000 orders and matching condition was the same. The metrics used for the measurement of the performance were: total matches, quantity of matched elements, execution time, memory consumption, throughput, average matching time and efficiency. As can be seen in Table 2, the number of matched and matched quantity is the same for both algorithms, which indicates consistency in the matching results. But the Greedy algorithm is less time consuming, less memory consuming, with higher throughput and lower average matching time. The results indicate that the Greedy implementation required less execution time and average matching time and achieved higher throughput than the evaluated Hash-Based implementation under the tested experimental conditions. The two implementations produced the same number of matches and matched quantity under the defined matching rules. Therefore, the observed computational differences are associated with the implementations and data structures used in this experiment.

The performance of the Greedy and Hash-based order matching algorithms was evaluated using a dataset containing 1,000 orders for testing different dataset, comprising 462 BUY orders and 538 SELL orders. Both algorithms produced 477 total matches with a matched quantity of 1,185. The Greedy algorithm completed execution in 0.064794 s, while the Hash-based algorithm required 0.42875 s. The corresponding memory usage was 0.2676 MB for Greedy and 0.2684 MB for Hash. Using total orders divided by execution time, the throughput was approximately 15,433.9 orders/s for Greedy and 2,332.4 orders/s for Hash. The average matching time was 0.00013584 s for Greedy and 0.00089885 s for Hash. Both algorithms achieved the same matching efficiency of 49.903%, indicating that they produced the same matching output under the defined matching conditions. Overall, for this 1,000-order dataset and the tested MATLAB implementations, Greedy showed lower execution time and average matching time and higher throughput.

Table 2. Comparison of Greedy and Hash-Based Order Matching Algorithms

PERFORMANCE METRICS	GREEDY MATCHING	HASH-BASED MATCHING
Dataset	tick_orderbook Dataset	tick_orderbook Dataset
Total Orders	3,000	3,000
Buy Orders	1,385	1,385
Sell Orders	1,615	1,615
Total Matches	1,420	1,420
Matched Quantity	5,207.978	5,207.978
Execution Time(sec)	0.256752	2.751413

Memory Used(MB)	0.6109	0.6123
Throughput(Orders/sec)	11684.2	1090.32
Average Matching Time(sec)	0.00018081	0.00193761
Efficiency(%)	49.26	49.26

For the tested 3,000-order dataset and the MATLAB implementations evaluated in this study, the Greedy implementation recorded lower execution time and average matching time and higher throughput than the Hash-Based implementation. The two approaches produced the same total number of matches and matched quantity under the defined matching rules. The execution time of Greedy and Hash Based order matching algorithms are presented in figure 1(a). The Hash Based algorithm took 2.751413 seconds and the Greedy algorithm took 0.256752 seconds for the execution time. The graph shows the time difference between the two implementations with the same experimental setup. The memory consumption of the two order matching algorithms (Greedy and Hash-Based) is shown in Figure 1(b). Using the Greedy algorithm, 0.6109 MB was used and using the Hash Based algorithm, 0.6123 MB was used. As depicted in the graph, for the experimental setups tested, the memory usage was comparable for both algorithms.

Figure 1(c)shows the throughput obtained for the two order matching algorithms, Greedy and Hash-Based. The Greedy algorithm was able to achieve throughput of 11684.2 orders/s and the Hash-Based algorithm was able to achieve 1090.32 orders/s. The following graph is a plot of the processing rate difference between the two algorithms, in the same experimental setting.

Figure 2, presents the overall MATLAB performance dashboard of the HFT Order Matching Engine. It offers a single visualization of the performance metrics of the Greedy and Hash-Based algorithms, such as execution time, memory consumption, throughput, matched quantity and efficiency. The dashboard facilitates simple and comparative assessment of the experimental results. The experimental results showed that the Greedy algorithm performed better than the Hash Based algorithm in terms of computational performance based on the test set.

Figure 1. Comparison of (a) Execution Time , (b) Memory Usage and (c)Throughput

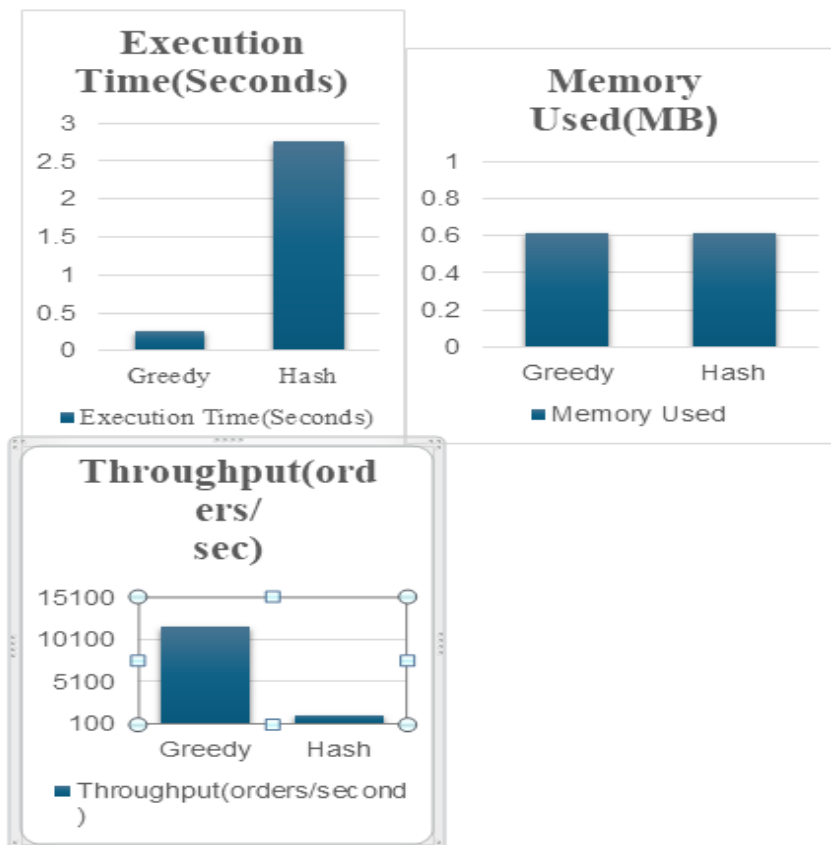
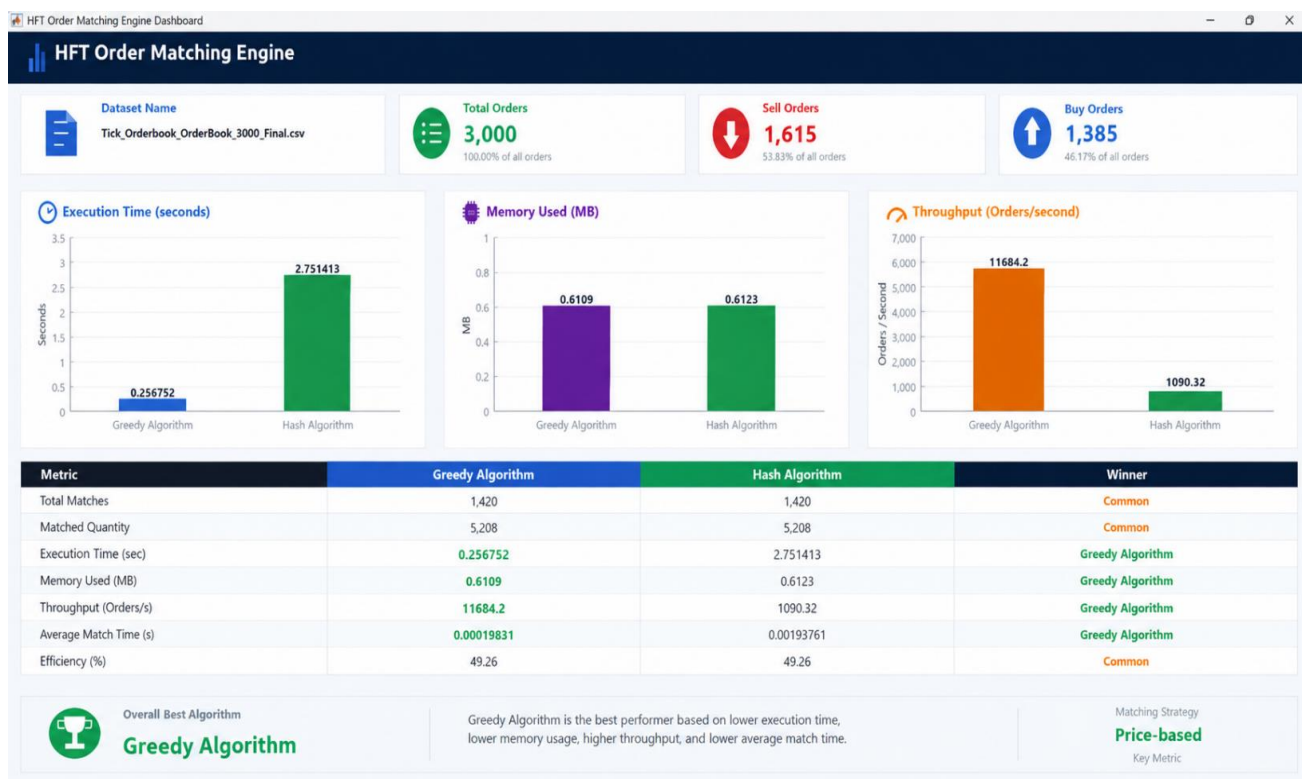


Figure 2. Overall Performance Dashboard



DISCUSSION

Both algorithms produced the same number of matches and matching efficiency under identical conditions using the same dataset of 3,000 orders and the same price compatibility rule. However, their computational performance differed. The Greedy algorithm completed the matching process in 0.256752 seconds, while the Hash-Based algorithm required 2.751413 seconds. Consequently, Greedy achieved a higher throughput of 11684.2 orders/s compared with 1090.32 orders/s for Hash-Based.

Memory usage was nearly identical, with Greedy using 0.6109 MB and Hash-Based using 0.6123 MB. The average matching time was also lower for Greedy at 0.00018081 seconds per match, compared with 0.00193761 seconds for Hash-Based. Both algorithms produced 1,420 total matches and a matched quantity of 5,207.978, with identical efficiency value of 49.26%.

The results indicate that the main difference between the approaches lies in computational performance rather than matching output. Although the Hash-Based approach provides structured organization based on price information, its additional processing resulted in higher execution time for the tested dataset. The MATLAB dashboard presents these performance measures through graphs, numerical values, and comparisons, making the differences easier to interpret.

Overall, for the tested 3,000-order dataset and experimental setup, the Greedy implementation recorded lower execution time, memory usage, and average matching time, along with higher throughput. These findings are specific to the algorithms, implementation, dataset, and experimental environment used and should not be generalized to all possible order-book workloads without further experimentation.

CONCLUSION

This study introduced a Data Science based HFT Order Matching Engine in MATLAB that compares Greedy, Hash Based Order Matching algorithm based on a common dataset of 3000 orders. When using both algorithms under the same matching conditions, they returned 1,420 matches and matched quantity of 5,207.978. The Greedy implementation took less time (0.256752 s), achieved higher throughput (11,684.2

orders/s), and had lower average matching time (0.00018081 s) than the Hash-Based implementation. Greedy had a little less memory usage, too. This shows how important algorithms are when it comes to computational performance when they are designed and implemented. These measures are easily visualized on the MATLAB dashboard and can be compared data-driven. The results are valid only for the data set and experimental setup, and serve as the foundation for future testing with a larger data set and optimized algorithms.

ACKNOWLEDGEMENT

The authors would like to thank Vellalar College for Women, Erode, for the academic support and facilities provided to carry out this study and are grateful to Kaggle for giving them the dataset used for the research.

REFERENCES

1. Aldridge I. High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems. 2nd ed. Hoboken, NJ: John Wiley & Sons; 2013. <https://doi.org/10.1002/9781119203803>.
2. Aquilina M, Budish E, O'Neill P. Quantifying the high-frequency trading “arms race”. The Quarterly Journal of Economics. 2022;137(1):493–564. <https://doi.org/10.1093/qje/qjab032>.
3. Avellaneda M, Stoikov S. High-frequency trading in a limit order book. Quantitative Finance. 2008;8(3):217–224. <https://doi.org/10.1080/14697680701381228>.
4. Bouchaud JP, Farmer JD, Lillo F. How markets slowly digest changes in supply and demand. In: Hens T, Schenk-Hoppé KR, editors. Handbook of Financial Markets: Dynamics and Evolution. Amsterdam: North-Holland; 2009. P. 57–160. <https://doi.org/10.1016/B978-012374258-2.50006-3>.
5. Cartea Á, Jaimungal S, Penalva J. Algorithmic and High-Frequency Trading. Cambridge: Cambridge University Press; 2015.
6. Cont R, Stoikov S, Talreja R. A stochastic model for order book dynamics. Operations Research. 2010;58(3):549–563. <https://doi.org/10.1287/opre.1090.0780>.
7. Gould MD, Porter MA, Williams S, McDonald M, Fenn DJ, Howison SD. Limit order books. Quantitative Finance. 2013;13(11):1709–1742. <https://doi.org/10.1080/14697688.2013.803148>.
8. Harris L. Trading and Exchanges: Market Microstructure for Practitioners. New York: Oxford University Press; 2003.
9. Hasbrouck J, Saar G. Low-latency trading. Journal of Financial Markets. 2013;16(4):646–679. <https://doi.org/10.1016/j.finmar.2013.05.003>.
10. Hendershott T, Jones CM, Menkveld AJ. Does algorithmic trading improve liquidity? The Journal of Finance. 2011;66(1):1–33. <https://doi.org/10.1111/j.1540-6261.2010.01624.x>.
11. Hendershott T, Riordan R. Algorithmic trading and the market for liquidity. Journal of Financial and Quantitative Analysis. 2013;48(4):1001–1024. <https://doi.org/10.1017/S0022109013000471>.
12. Kirilenko A, Kyle AS, Samadi M, Tüzel T. The Flash Crash: High-frequency trading in an electronic market. The Journal of Finance. 2017;72(3):967–998. <https://doi.org/10.1111/jofi.12498>.
13. Law B, Viens F. Market making under a weakly consistent limit order book model. High Frequency. 2020;2(3–4):215–238. <https://doi.org/10.1002/hf2.10050>.
14. Madhavan A. Market microstructure: A survey. Journal of Financial Markets. 2000;3(3):205–258. [https://doi.org/10.1016/S1386-4181\(00\)00007-0](https://doi.org/10.1016/S1386-4181(00)00007-0).
15. Menkveld AJ. High frequency trading and the new market makers. Journal of Financial Markets. 2013;16(4):712–740. <https://doi.org/10.1016/j.finmar.2013.06.006>.
16. Menkveld AJ. The economics of high-frequency trading: Taking stock. Annual Review of Financial Economics. 2016; 8:1–24. <https://doi.org/10.1146/annurev-financial-121415-033010>.
17. Sedgewick R, Wayne K. Algorithms. 4th ed. Boston: Addison-Wesley; 2011.
18. Gould MD, Porter MA, Howison SD. Quasi-centralized limit order books. Quantitative Finance. 2017;17(6):831–846. <https://doi.org/10.1080/14697688.2016.1247980>.