The Exchange Problem

Mohit Garg
Indian Institute of Science, Bengaluru, India

Suneel Sarswat
National Institute of Securities Markets, Mumbai, India

Abstract

Auctions are widely used in exchanges to match buy and sell requests. Once the buyers and sellers place their requests, the exchange determines how these requests are to be matched. The two most popular objectives used while determining the matching are maximizing volume *with dynamic pricing* and maximizing volume *at a uniform price*. In this work, we study the algorithmic complexity of the problems arising from these matching tasks.

For **dynamic-price matching**, we establish a lower bound of $\Omega(n \log n)$ on the running time, thereby proving that the currently best-known $O(n \log n)$ algorithm is time-optimal. In contrast, for **uniform-price matching**, we present a **linear-time** algorithm, improving upon previous methods that require $O(n \log n)$ time to match n requests.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Applied computing → Electronic commerce

Keywords and phrases Exchanges, Double Auctions, Matching Algorithms, Element Distinctness, Time Complexity

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.25

Funding *Mohit Garg*: Supported in part by a fellowship from the Walmart Center for Tech Excellence at IISc (CSR Grant WMGT-23-0001).

Acknowledgements The authors express their gratitude to Jaikumar Radhakrishnan for bringing the element distinctness problem to their attention and for engaging in insightful discussions that enriched their understanding of it. They are also thankful to him for helping improve the presentation of this work.

1 Introduction

We study problems of the following nature: The input is a list of trade requests from buyers and sellers of a particular product. Each request consists of a price and a quantity. The buyer's request, known as a *bid*, is represented by a pair (p, q), which indicates that the buyer offers to buy a maximum of q units of the product while paying at most p per unit. Similarly, a seller's request, known as an *ask*, is also a pair (p, q), which indicates that the seller offers to sell a maximum of q units of the product provided they receive at least p per unit. On receiving the input, we are required to generate a *matching*, which is a collection of *transactions*.[1] A transaction between a bid (p_b, q_b) and an ask (p_a, q_a) consists of a transaction price $p \in [p_a, p_b]$ and a transaction quantity $q \le \min\{q_b, q_a\}$, indicating that q units of the product change hands between the traders at price p per unit. For each trade request (p, q), the sum of the total transaction quantities of the transactions involving the request must be at most q. The volume of the matching is the total quantity that changes hands. The matching is *uniform* if all transactions occur at a common price. We consider the following two tasks.

[1] We use the term *matching*, which has been traditionally used for this object in the auction theory literature [10, 13, 12, 9, 4], while a term like *flow* might be more suitable for a computer science audience.

7th Conference on Advances in Financial Technologies (AFT 2025).
Editors: Zeta Avarikioti and Nicolas Christin; Article No. 25; pp. 25:1–25:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Task 1: Determine a matching with the largest volume.

Task 2: Determine a uniform matching with the largest volume.

The first task can be solved by formulating it as a max-flow problem. However, due to the underlying problem structure, simpler solutions are available for these tasks. In fact, two simple and almost identical algorithms can be used for these tasks. We describe them below.

Algorithms: These algorithms start with sorting the list of bids in decreasing order of their prices. Next, the list of asks is sorted. For the first task, the sell requests are sorted in decreasing order of their prices. Whereas, for the second task, the sell requests are sorted in increasing order of their prices.
After the sorting step, both algorithms work in linear time as follows. The bid on top of its sorted list (p_b, q_b) is matched with the ask on top of its sorted list (p_a, q_a) if they are compatible, i.e., $p_b \geq p_a$. In this case, a transaction between them is established with quantity $q = \min(q_b, q_a)$; a quantity of q is reduced from their existing quantities of q_b and q_a; finally, the 0 quantity request(s) are deleted from the lists. If the requests are incompatible, the topmost ask in the sorted list of asks is deleted. The above steps are then repeatedly applied until one of the lists becomes empty. Finally, for the first task, for each matched pair: $\langle (p_b, q_b), (p_a, q_a) \rangle$ any transaction price in the interval $[p_a, p_b]$ can be put. For the second task, let the last matched pair in the above process be $\langle (p_b, q_b), (p_a, q_a) \rangle$. Then, for all matched pairs any number in the interval $[p_a, p_b]$ can be chosen as the common transaction price.

Clearly, the above algorithms take time $O(n \log n)$ in the comparison model. We ask the following question.

Do these algorithms have the optimal running time for the above tasks?

The problems we consider in this work are slight generalizations of the above tasks where bids and asks have additional parameters, and the objectives incorporate a notion of *fairness.* These problems arise out of *call auctions*, which form the backbone of many modern trading platforms, most notably stock exchanges, commodities markets, and various electronic marketplaces. Numerous works have focused on the mechanism-design and game-theoretic aspects of such auctions (e.g., see [3] and the citations therein), but despite the fundamental importance and widespread usage of these auctions, the precise computational complexity of determining maximum-volume matchings – either in uniform-price or dynamic-price settings – has remained relatively unexplored, with only a few works that include a runtime analysis of their algorithms ([12, 10, 13]. For the above tasks, we prove the following.

▶ **Theorem 1.** *Any comparison-based algorithm for Task 1 requires $\Omega(n \log n)$ time in the worst-case, where n is the number of trade requests.*

▶ **Theorem 2.** *There exists a comparison-based algorithm for Task 2 that runs in $O(n)$ time, where n is the number of trade requests.*

These findings reveal an unexpected dichotomy. While **Task 1** intrinsically requires sorting, **Task 2** admits a more efficient linear-time solution. This result has not appeared previously, despite its importance to many exchanges' opening and closing routines.

To describe the general problems that we consider, previous results, and our contributions formally, we need the following preliminaries.

1.1 Preliminaries

We now formally describe call auctions which are slight generalizations of the tasks introduced above.

Setup, Definitions, and Notation

A call auction is used at an exchange to match the bids and asks of traders for a particular product. Bids and asks are collectively termed orders. Orders are collected by the exchange for a fixed duration of time, at the end of which the collected orders are simultaneously matched to produce transactions. Each transaction between a bid and an ask consists of a transaction price and a transaction quantity. Each order consists of four attributes that are natural numbers: a unique identification number (id), a unique timestamp, a limit price, and a quantity that is at least one.[2] $\mathsf{timestamp}(w)$ represents the time the order w was received. For an order w, the limit price $\mathsf{price}(w)$ is the minimum (if w is an ask) or maximum (if w is a bid) possible transaction price of a transaction involving w. For an order w, the quantity $\mathsf{qty}(w)$ represents the total quantity offered for trade, i.e., the sum of transaction quantities of transactions involving w in the matching must be at most $\mathsf{qty}(w)$. Given a collection of orders with distinct ids, i.e., given a set of bids and asks with distinct ids, a matching is a collection of transactions that satisfy the natural constraints. Next, we define a few important terms and a matching formally.

Given a bid b and an ask a, we say that b and a are **tradable** if $\mathsf{price}(b) \geq \mathsf{price}(a)$. For a set of transactions M and an order w, $\mathsf{Qty}(w, M)$ denotes the sum of transaction quantities of transactions in M that involve w, and $\mathsf{Vol}(M)$ denotes the sum of all transaction quantities of the transactions in M. With slight abuse of notation for a set of asks or a set of bids Ω, we define $\mathsf{Vol}(\Omega)$ to be the sum of the quantities of the orders in Ω.

A set of transactions M is a **matching** over a list of bids B and asks A with distinct ids if (i) For each transaction $m \in M$, the bid and ask of m come from B and A, respectively. (ii) For each transaction $m \in M$, the bid b of m is tradable with the ask a of m. (iii) For each order $w \in B \cup A$, $\mathsf{Qty}(w, M) \leq \mathsf{qty}(w)$.

Fairness and Competitiveness

In the problems we study, the matching produced needs to be fair, which essentially states that more competitive orders (based on price-time priority) have to be given preference in the matching. To formally define a fair matching, we first introduce the notion of competitiveness. For bids b_1 and b_2 we say b_1 is more **competitive** than b_2, denoted by $b_1 \succ b_2$, iff $\mathsf{price}(b_1) > \mathsf{price}(b_2)$ or $(\mathsf{price}(b_1) = \mathsf{price}(b_2)$ and $\mathsf{timestamp}(b_1) < \mathsf{timestamp}(b_2))$. Similarly, for asks a_1 and a_2, we say a_1 is more competitive than a_2, denoted by $a_1 \succ a_2$, iff $\mathsf{price}(a_1) < \mathsf{price}(a_2)$ or $(\mathsf{price}(a_1) = \mathsf{price}(a_2)$ and $\mathsf{timestamp}(a_1) < \mathsf{timestamp}(a_2))$. We say a matching M is fair on bids if a bid b participates in M, then all bids that are more competitive than b must be fully traded in M. Formally, M is fair on bids iff for all pairs of bids b_1 and b_2 such that $b_1 \succ b_2$, $\mathsf{Qty}(b_2, M) \geq 1 \implies \mathsf{Qty}(b_1, M) = \mathsf{qty}(b_1)$. Similarly, a matching M is fair on asks iff for all pairs of asks a_1 and a_2 such that $a_1 \succ a_2$, $\mathsf{Qty}(a_2, M) \geq 1 \implies \mathsf{Qty}(a_1, M) = \mathsf{qty}(a_1)$. A matching is **fair** if it is fair on bids as well as fair on asks.

[2] Later, the orders given as input to the algorithms, are all assumed to have distinct ids and distinct timestamps.

Uniform vs. Dynamic Price

There are two main types of call auctions: uniform-price and dynamic-price. In uniform price call auctions, the exchange is supposed to output a fair matching with maximum volume subject to the constraint that all transaction prices in the matching are identical. Customarily, the transaction price of a uniform price matching is referred to as the *equilibrium price*, and the process of discovering this price is called *price discovery.* In contrast, for dynamic price call auctions, the exchange is supposed to output a fair matching with maximum volume, where the transaction prices need not be the same. It must be noted that dropping the requirement of fairness from a dynamic price matching can only make the problem easier; the following result states that for any matching, there exists a fair matching of the same volume.

▶ **Theorem 3** (Proved in [9, 4]). *Given a set of bids B, a set of asks A, and a matching M over (B, A), one can find a fair matching M' over (B, A) such that*
(a) $\mathsf{Vol}(M) = \mathsf{Vol}(M')$*, and*
(b) *if M is uniform, then M' is uniform.*

Furthermore, [9, 4] provides an $O(n \log n)$-time algorithm to compute the matching M'.

▶ Remark 4. It is clear from the above theorem, that unlike uniformity, ensuring fairness does not change the volume of a matching. More precisely, given sets of bids B and asks A, consider the following matchings over (B, A). Let M_1 be a largest volume matching, M_2 be a fair matching with the largest volume, M_3 be a uniform matching with the largest volume, and finally let M_4 be a uniform and fair matching with the largest volume. Then,

$$\mathsf{Vol}(M_1) = \mathsf{Vol}(M_2) \geq \mathsf{Vol}(M_3) = \mathsf{Vol}(M_4).$$

Nevertheless, computing M_1 can be easier than computing M_2, and similarly computing M_3 can be easier than computing M_4. More precisely, computing M_1 reduces to computing M_2, and computing M_3 reduces to computing M_4.

Two Problems

The two matching problems that we study in this work can be stated as follows.

- **Problem 1: Dynamic-Price Matching.** Given a set of bids B and a set of asks A, find a fair matching over (B, A) of maximum volume.
- **Problem 2: Uniform-Price Matching.** Given a set of bids B and a set of asks A, find a fair and uniform matching over (B, A) of maximum volume.

Two models

We obtain our lower bound results in the following models of computation.

- **The Comparison Model:** We assume that the prices of the orders in the input are not given to us explicitly. Instead, two prices can be compared to each other via an oracle in unit time.
- **The Binary Query Model:** We will also derive a lower bound in a more general model, namely the binary query model, where the oracle can be made to evaluate an arbitrary function $f : \mathbb{N} \times \mathbb{N} \rightarrow \{0, 1\}$ on two prices in unit time. Note that the query function can be different for different queries.

▶ Remark 5. Our upper bound results do adhere to the more restricted comparison model.

We are now ready to state the previous and new results.

1.2 Earlier Works

We first briefly summarize the past results that were obtained for Problems 1 and 2. In the following, n represents the number of orders in the input, i.e., $n = |A| + |B|$.

- First, in [12] it was shown that Problem 2 can be solved in $O(n \log n)$ time.
- Next, in [13] it was shown that Problem 1 can be solved in $O(n^2)$ time under the assumption that all orders have unit quantity. The authors mention that for multi-unit orders, one can simply break an order into multiple single-unit orders and use their algorithm. But, this would then result in overall complexity $O(Q^2)$, where Q is the sum of the total demand and total supply, i.e., $Q = \mathsf{Vol}(A) + \mathsf{Vol}(B)$.
- This was improved in [10] where it was shown that Problem 1 can be solved in $O(n \log n)$ time for single-unit orders.
- Finally, inspired from previous works, in [9, 4] algorithms for both Problems 1 and 2 were presented that run in $O(n \log n)$ time each for arbitrary multi-unit orders. However, the authors did not analyze the time complexity of their algorithms, as they were more focused on formalizing these algorithms in a theorem prover. Their algorithms are general versions of the two algorithms we saw on page 2, where instead of sorting the lists of bids and asks by their prices, they are sorted by their competitiveness. After this modification, the first algorithm yields a maximum volume matching (note that Problem 1 requires the output matching to be also fair), whereas the second algorithm yields a uniform and fair matching with maximum volume (precisely solving Problem 2). For solving Problem 1, in addition to the above algorithm, they use an $O(n \log n)$ time algorithm that takes an arbitrary matching and outputs a fair matching of the same volume (this is always possible, see Theorem 3). It is straightforward to see that their algorithms indeed run in $O(n \log n)$ time.

1.3 Our Contribution

Now we describe the results obtained in this work.

Tight Lower Bound for Dynamic-Price Matching. For Problem 1, we show that any comparison-based algorithm for computing a maximum-volume matching (even ignoring fairness) requires $\Omega(n \log n)$ time in the worst case. We also show a stronger $\Omega(n\sqrt{\log n})$ bound in the broader binary query model. This confirms that the widely used $O(n \log n)$ (which also works for determining a maximum volume matching) sorting-based methods ([10, 9, 4]) are asymptotically optimal in the comparison model.

▶ **Theorem 6** (Result 1)**.** *Any algorithm that takes as input a set of bids B and a set of asks A and computes a matching over (B, A) with the maximum volume has a worst-case running time of $\Omega(n \log n)$ in the comparison model and $\Omega(n\sqrt{\log n})$ in the binary query model, where $n = |B| + |A|$.*

The proof of Theorem 6 also applies to Theorem 1. In this proof, we obtain the desired lower bound by reducing the element distinctness problem over a small domain to the maximum volume matching problem.[3] In order to prove the correctness of our reduction, we crucially employ a stronger inequality than the demand-supply inequality of [10]. This

[3] An entropy-based argument similar to the one used for sorting in [11] also yields a lower bound for the maximum matching problem in the comparison model, but it fails to deliver any non-trivial lower bound in the binary query model.

stronger inequality, which we prove, can be of independent interest. Our result then follows from applying known lower bounds on the time complexity of the element distinctness problem [2, 7]. We include a proof of the lower bound in the comparison model making the argument in [7] more precise as it seems to be missing some important details.

Tight Linear-Time Algorithm for Uniform-Price Matching. For Problem 2, contrary to the dynamic case, we design a new algorithm that achieves a uniform-price matching in $O(n)$ time. This is an improvement over the previous algorithms [12, 13, 9, 4] that take $O(n \log n)$ time to match n requests.

▶ **Theorem 7** (Result 2). *There exists an algorithm that takes as input a set of bids B and a set of asks A, and outputs a uniform and fair matching with maximum volume in $O(n)$ time, where $n = |B| + |A|$.*

Theorem 7 immediately implies Theorem 2, as ensuring fairness does not reduce the size of a uniform matching, as observed earlier. Note that obtaining a linear time algorithm in the volume of the input orders is simpler than Result 2. Our improvement for uniform-price matching, roughly speaking, comes from eliminating the sorting step in the algorithm of [9, 4] as described earlier. Instead, we compute the "medians" of the bids and asks in linear time which bisects the sets of bids and asks into two "equal halves" each. Depending on whether the medians are tradable or not, we can either get rid of half of the bids and asks, or match half of the input in linear time, thereby reducing the problem size by half in either case. This results in a linear time algorithm. A number of interesting challenges are encountered while implementing the above idea. In particular, when we discard "half" of the orders, they might still be matchable with the other "half" that we recurse on; prima facie, it is not obvious why such orders can be safely discarded.

1.4 Practical Significance

Call auctions are ubiquitous in financial markets. Major stock exchanges – including the New York Stock Exchange (NYSE), NASDAQ, and various global exchanges – often rely on uniform-price call auctions at market open and market close to establish a single clearing price. The main advantage is its simplicity and impartiality: every matched trader pays/receives the same price. Similar rules govern IPO allocations and some commodity trading platforms. Our work thus not only answers fundamental complexity questions – shedding light on the intrinsic difficulty of maximum-volume matching – but also yields a more efficient uniform-price procedure. In high-frequency or large-volume trading environments, any reduction in algorithmic overhead can be consequential.

Moreover, these results highlight an intellectually appealing paradox: while dynamic-price matching intrinsically requires sorting (or an equivalent $O(n \log n)$ process) under the comparison model, uniform-price matching admits a more efficient linear-time solution. This dichotomy had never been formally established, despite the centrality of call auctions in marketplace design.

This work offers a fundamental yet previously unexamined algorithmic advance for widely used exchange procedures, and we believe these insights will be of significant interest to researchers and practitioners in auction theory, matching markets, operations research, and financial engineering alike.

1.5 Further Related Work

In [5, 6], continuous auctions were studied, that form a class of double auctions that complement the call auctions. The authors presented an algorithm that implements the continuous auction and runs in $O(n \log n)$ time for processing n *instructions* (requests/orders). They also showed that their algorithm is time-optimal by reducing the task of sorting to continuous auctions.

Organization of the Rest of the Paper

Theorems 1 and 6 are proved in Section 2. Next, in Section 3, Theorem 7 is proved. Finally, we conclude the paper in Section 4 with some concluding remarks.

2 Lower Bound for Maximum-Volume Matching

In this section we prove Theorem 6 (note that the proof also works for Theorem 1), which we restate.

▶ **Theorem 6** (Result 1). *Any algorithm that takes as input a set of bids B and a set of asks A and computes a matching over (B, A) with the maximum volume has a worst-case running time of $\Omega(n \log n)$ in the comparison model and $\Omega(n\sqrt{\log n})$ in the binary query model, where $n = |B| + |A|$.*

In our proof, we will show a reduction from the following problem to the maximum-volume matching problem.

Element Distinctness Problem (on small domain). Given an input $(x_1, \cdots, x_n)$, where each $x_i \in [n]$, check whether there are distinct indices i and j such $x_i = x_j$.[4]

Our result then immediately follows from the fact that the element distinctness problem on a small domain requires time $\Omega(n \log n)$ in the comparison model [7] (for which we include a proof in the next subsection) and time $\Omega(n\sqrt{\log n})$ in the binary query model [2].

Furthermore, our reduction will rely on a stronger version of the demand-supply inequality of [10], which states that the volume of any matching is upper bounded by the sum of the total demand and the total supply at any price.

▶ **Theorem 8** (demand-supply inequality). *If M is a matching over a set of bids B and a set of asks A, then for all numbers p, we have*

$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B_{\geq p}) + \mathsf{Vol}(A_{\leq p})$$

where $B_{\geq p} \subseteq B$ consists of bids whose limit prices are at least p, $A_{\leq p} \subseteq A$ consists of asks whose limit prices are at most p.

Our strengthening, which is proved in Section 2.2, comes from observing that the above inequality holds even if we replace $\mathsf{Vol}(B_{\geq p})$ by $\mathsf{Vol}(B_{>p})$ or $\mathsf{Vol}(A_{\leq p})$ by $\mathsf{Vol}(A_{<p})$, and can be stated as follows.

[4] For $n \geq 1$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$.

▶ **Theorem 9** (strong demand-supply inequality). *If M is a matching over a set of bids B and a set of asks A, then for all numbers p, we have*

$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(A_{<p}) + \min(\mathsf{Vol}(B_{=p}), \mathsf{Vol}(A_{=p})),$$

where $B_{>p} \subseteq B$ consists of bids whose limit prices are greater than p, $A_{<p} \subseteq A$ consists of asks whose limit prices are less than p, and $B_{=p} \subseteq B$ (or $A_{=p} \subseteq A$) consists of bids (or asks) whose limit prices are exactly p.

We are now ready to do our proof.

Proof of Theorem 6. Let MM be an algorithm for the maximum matching problem. We reduce the element distinctness problem on a small domain to MM in linear time.

The reduction. Given an instance $X = (x_1, \cdots, x_n)$ for the element distinctness problem, we construct two sets of orders $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$ and $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ such that the quantity of each order in $\Omega \cup \Lambda$ is set to 1. We set the prices as follows. $\mathsf{price}(\omega_i) = i$ and $\mathsf{price}(\lambda_i) = x_i$. Observe that the prices of orders in Ω are all distinct. Next, we run the maximum matching algorithm MM twice on these inputs by first treating Ω as the set of bids and then treating Ω as the set of asks to obtain two matchings M_1 and M_2, respectively; $M_1 = \mathsf{MM}(\Omega, \Lambda)$ and $M_2 = \mathsf{MM}(\Lambda, \Omega)$.

We now claim that if elements of X are all distinct then $\mathsf{Vol}(M_1) = \mathsf{Vol}(M_2) = n$, otherwise $\mathsf{Vol}(M_1) < n$ or $\mathsf{Vol}(M_2) < n$. Note that if we show this claim the reduction is complete, as we can then solve the element distinctness problem mentioned above in time $O(n)$ plus twice the time taken by MM on $2n$ orders.

It is easy to see that if the x_i's are all distinct, then $\mathsf{Vol}(M_1) = \mathsf{Vol}(M_2) = n$; the bid with price i is matched with the ask with price i.

Now, we show that if the x_i's are not distinct, then $\mathsf{Vol}(M_1) \leq n-1$ or $\mathsf{Vol}(M_2) \leq n-1$. From the pigeon-hole principle, if two elements are repeating in X, then one of the elements from $[n]$ must be missing from X. Let the smallest missing element in X be m and the smallest repeating element in X be r.

For a set of orders Ω, we define $\Omega_{=t} := \{\omega \in \Omega \mid \mathsf{price}(\omega) = t\}$, $\Omega_{<t} := \{\omega \in \Omega \mid \mathsf{price}(\omega) < t\}$, and $\Omega_{>t} := \{\omega \in \Omega \mid \mathsf{price}(\omega) > t\}$. Now, observe that $\mathsf{Vol}(\Lambda_{=m}) = 0$ (since m is missing from X which is the set of prices of Λ), which implies $\min(\mathsf{Vol}(\Omega_{=m}), \mathsf{Vol}(\Lambda_{=m})) = 0$.

We now consider two cases: $m < r$ or $m > r$.

Case: $m < r$. In this case we will show that $\mathsf{Vol}(M_1) < n$. Observe that the number of elements in X that are at most m is precisely $m-1$, since no element smaller than m is repeating and m itself is missing. Here, the input x_i's are set to be the ask prices, i.e., Λ is the set of asks, and Ω is the set of bids. Therefore, $\mathsf{Vol}(\Lambda_{<m}) = |\{\lambda \in \Lambda \mid \mathsf{price}(\lambda) < m\}| = m-1$ and $\mathsf{Vol}(\Omega_{>m}) = n-m$. Hence, from Theorem 9,

$$\begin{aligned}\mathsf{Vol}(M_1) &\leq \mathsf{Vol}(\Omega_{>m}) + \mathsf{Vol}(\Lambda_{<m}) + \min(\mathsf{Vol}(\Omega_{=m}), \mathsf{Vol}(\Lambda_{=m})) \\ &= (n-m) + (m-1) + 0 = n-1.\end{aligned}$$

Case: $m > r$. In this case we will show that $|M_2| < n$. Note that no element in X is missing below m but some elements are repeating. Thus, the number of elements that are greater than m are at most $n-m$. Hence, similar to the above case, applying Theorem 9, we have $\mathsf{Vol}(M_2) \leq \mathsf{Vol}(\Lambda_{>m}) + \mathsf{Vol}(\Omega_{<m}) + \min(\mathsf{Vol}(\Lambda_{=m}), \mathsf{Vol}(\Omega_{=m})) \leq (n-m) + (m-1) + 0 = n-1$. ◀

2.1 Lower Bound for Element Distinctness on Small Domain

We now show a lower bound on the number of comparisons needed to solve the element distinctness problem on a small domain. This is a known result, but the proof that we could find has gaps, so we decided to include a complete proof.

▶ **Theorem 10.** *Any algorithm that takes as input $(x_1, \cdots, x_n)$, where each $x_i \in [n]$, and can decide whether there exist i and j such that $i \neq j$ and $x_i = x_j$, takes $\Omega(n \log n)$ time in the comparison model.*

Proof. Any algorithm for element distinctness can be seen as a decision tree where each internal node is labeled by a comparison $X_i : X_j$, for some $i \neq j$, and each leaf is labeled either Yes or No. On a given input $(x_1, \cdots, x_n)$ the decision tree is applied as follows. We start at the root node, traverse a path to a leaf, and declare the label of the leaf as the output. On encountering a node labeled $X_i : X_j$, if $x_i \leq x_j$, we take its left child as the next node in the path, otherwise, we take its right child. Inputs corresponding to permutations on $[n]$ must receive the answer Yes, and inputs where $x_i = x_j$ for some $i \neq j$ must receive the answer No.

We fix a decision tree T for the element distinctness problem and prove that its height is at least $\Omega(n \log n)$. We make the following claim.

▷ Claim 11. Each permutation on $[n]$ ends up in a distinct leaf of T.

This immediately implies that the number of leaves is at least $n!$ and hence the height of the tree is at least $\Omega(\log n!) = \Omega(n \log n)$. We now turn to proving the above claim.

For the sake of contradiction we assume two distinct permutations $\sigma = (\sigma_1, \cdots, \sigma_n)$ and $\tau = (\tau_1, \cdots, \tau_n)$ end up at the same leaf L of T. To obtain a contradiction we will find an input $(S_1, \cdots, S_n)$ where $S_i = S_j$ for some $i \neq j$ which also ends up in the leaf L.

We define a poset $P(L)$ on the set of symbols $X = \{X_1, \cdots, X_n\}$. In our poset when X_i and X_j are related, we will write $X_i \leq X_j$ and call it a constraint. For each node labeled $X_i : X_j$ in the path from the root to the leaf L in the tree T, we have a constraint of the form $X_i \leq X_j$ or $X_j \leq X_i$ (in fact, $X_j < X_i$ suffices, but we use the weaker form) which must be satisfied for a computation to take this path. Let C be the set of all such constraints. We define $C(L) = \bigcap\{\hat{C} \mid C \subseteq \hat{C} \text{ such that } (X, \hat{C}) \text{ is a poset}\}$; notice that this is a non-empty intersection, as we have at least one poset, namely the total order $X_{\sigma(1)} \leq X_{\sigma(2)} \leq \cdots \leq X_{\sigma(n)}$ which satisfies all constraints in C as σ ends up in the leaf L. We define $P(L) := (X, C(L))$.

We say an input $(x_1, \cdots, x_n)$ *respects* a poset P if $x_i \leq x_j$ whenever $X_i \leq X_j$ is a constraint in P. The following proposition is easy to prove.

▶ **Proposition 12.** *An input reaches the leaf L iff it respects the poset $P(L)$.*

Next, observe that since σ and τ are two distinct permutations that respect $P(L)$, the length of a largest chain in $P(L)$ is at most $n - 1$.[5] Mirsky's theorem [8] then implies that there is an antichain decomposition of $P(L)$ with at most $n - 1$ antichains. Since X has n elements, there must be an antichain in $P(L)$ that has at least two elements. Let A be a maximal antichain in $P(L)$ that has at least two elements. We can write X as a disjoint union of three sets $X = Q \uplus A \uplus R$, where Q consists of elements below A and R consists of elements

[5] At this point, the proof in [7] instantly concludes that there must be two elements in the poset that are not related and fixes an input which respects the poset and where these two coordinates are equal, to complete the proof, without specifying why such an input exists. We fill such gaps in the proof.

above A in the poset $P(L)$: $Q = \{X_i \in X \setminus A \mid$ there exists $X_j \in A$ such that $X_i \leq X_j\}$ and $R = \{X_i \in X \setminus A \mid$ there exists $X_j \in A$ such that $X_j \leq X_i\}$ (observe that an element cannot be both above and below A). The constraints in $C(L)$ restricted to Q and restricted to R give rise to the posets: P_Q and P_R. Now we define the poset (X, D) such that $C(L) \subseteq D$ as follows. Take a linear extension of P_Q: $X_{t_1} \leq \cdots \leq X_{t_q}$ and add all the implied constraints to D. Do not add any constraints between the elements of the antichain $A = \{X_{t_{q+1}}, X_{t_{q+2}}, \cdots, X_{t_{q+a}}\}$. Take a linear extension of P_R: $X_{t_{q+a+1}} \leq \cdots \leq X_{t_n}$ and add all the implied constraints to D. Furthermore, add the following constraints to D: For each $X_i \in X$, add $X_i \leq X_i$. For each $X_i \in Q$ and $X_j \in A \cup R$, add $X_i \leq X_j$. For each $X_i \in A$ and each $X_j \in R$, add $X_i \leq X_j$. Clearly, $C(L) \subseteq D$. Now fix the input S such that $S_{t_1} = 1, S_{t_2} = 2, \cdots, S_{t_q} = q$, $S_{t_{q+1}} = S_{t_{q+2}} = \cdots = S_{t_{q+a}} = q + 1$, $S_{t_{q+a+1}} = q + 2, S_{t_{q+a+2}} = q + 3, \cdots, S_{t_n} = n - (a - 1)$. Now, S clearly respects the poset (X, D). Thus, it also respects the poset $P(L)$ (as $C(L) \subseteq D$) and reaches the leaf L. Since $a = |A| \geq 2$, S is a No instance which reaches L, a contradiction. ◀

2.2 Stronger Demand-Supply Inequality

In this subsection, we prove Theorem 9.

▶ **Theorem 9** (strong demand-supply inequality). *If M is a matching over a set of bids B and a set of asks A, then for all numbers p, we have*

$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(A_{<p}) + \min(\mathsf{Vol}(B_{=p}), \mathsf{Vol}(A_{=p})),$$

where $B_{>p} \subseteq B$ consists of bids whose limit prices are greater than p, $A_{<p} \subseteq A$ consists of asks whose limit prices are less than p, and $B_{=p} \subseteq B$ (or $A_{=p} \subseteq A$) consists of bids (or asks) whose limit prices are exactly p.

Proof of Theorem 9. First observe that the volume of any matching is upper bounded by the volume of all bids as well as the volume of all asks, i.e., if M is a matching over (B, A), then

$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B) \quad \text{and} \quad \mathsf{Vol}(M) \leq \mathsf{Vol}(A).$$

To prove Theorem 9, it suffices to prove the following two inequalities.

$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(A_{<p}) + \mathsf{Vol}(B_{=p});$$
$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(A_{<p}) + \mathsf{Vol}(A_{=p}).$$

To prove the first inequality we partition the matching M into two sets: $M_1 = \{(b, a, q, p') \in M \mid \mathsf{price}(b) \geq p\}$ consisting of all transactions in M whose participating bid price is at least p and $M_2 = \{(b, a, q, p') \in M \mid \mathsf{price}(b) < p\}$ consisting of all transactions in M whose participating bid price is strictly less than p. Thus, $\mathsf{Vol}(M) = \mathsf{Vol}(M_1) + \mathsf{Vol}(M_2)$.

It is easy to see that M_1 is a matching over sets of bids $B_{\geq p}$ and asks A, and hence from the above observation, $\mathsf{Vol}(M_1) \leq \mathsf{Vol}(B_{\geq p}) = \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(B_{=p})$.

Next, we prove that M_2 is a matching over sets of bids B and asks $A_{<p}$. Consider a transaction $m = (b, a, q, p')$ from M_2, which is between a bid b and an ask a. Since $m \in M$, $\mathsf{price}(b) \geq \mathsf{price}(a)$, and from the definition of M_2, we have $\mathsf{price}(b) < p$. This implies $\mathsf{price}(a) < p$, i.e., asks of M_2 come from $A_{<p}$. Hence, M_2 is a matching over $(B, A_{<p})$, and applying the above observation again, we have $\mathsf{Vol}(M_2) \leq \mathsf{Vol}(A_{<p})$.

Combining, we have $\mathsf{Vol}(M) = \mathsf{Vol}(M_1) + \mathsf{Vol}(M_2) \leq \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(B_{=p}) + \mathsf{Vol}(A_{<p})$, which completes the proof of first inequality.

Similarly, we can prove the second inequality by first partitioning the matching M into $M_1 = \{(b, a, q, p') \in M \mid \mathsf{price}(b) > p\}$ and $M_2 = \{(b, a, q, p') \in M \mid \mathsf{price}(b) \le p\}$, noticing that M_1 is a matching over $(B_{>p}, A)$ and M_2 is matching over $(B, A_{\le p})$, and finally, using the observation again to obtain the desired inequality. ◀

3 A Linear-Time Algorithm for Uniform-Price Matching

In this section, we prove Theorem 7, which we first restate.

▶ **Theorem 7** (Result 2). *There exists an algorithm that takes as input a set of bids B and a set of asks A, and outputs a uniform and fair matching with maximum volume in $O(n)$ time, where $n = |B| + |A|$.*

We begin by first describing the previous $O(n \log n)$ algorithm as described in [9, 4], which we will critically use in establishing the correctness of our linear time algorithm.

We may assume that the input lists of asks A and bids B are such that $\mathsf{Vol}(A) = \mathsf{Vol}(B)$. This can be achieved by adding a dummy order. Say $\mathsf{Vol}(A) < \mathsf{Vol}(B)$, then add a dummy ask with $\mathsf{price} = \infty$ and $\mathsf{qty} = \mathsf{Vol}(B) - \mathsf{Vol}(A)$. If $\mathsf{Vol}(A) > \mathsf{Vol}(B)$, then add a dummy bid with $\mathsf{price} = -1$ and $\mathsf{qty} = \mathsf{Vol}(A) - \mathsf{Vol}(B)$. Strictly speaking, according to our definition, we cannot set the price to be ∞ or -1, but our definitions can be adjusted to allow for this.

3.1 Previous Algorithm

We describe the algorithm in [9, 4], which we denote by UM. The transaction prices that are set by UM are not guaranteed to be uniform initially. After the matching is produced, a linear time algorithm is employed to put a uniform transaction price for all the transactions. For example, the maximum of the limit prices of the asks that participate in the output matching can be set as the transaction price of all the transactions in the output matching. Consequently, in the algorithms presented below, we will not worry about putting a transaction price in the transactions.

Given the list of bids B and asks A, UM first sorts the lists based on their respective competitiveness with the most competitive order on the top. It then invokes Match on the sorted lists B, A, and an empty matching M. Throughout the algorithm, M can only grow, and at the end of the algorithm, M will contain a desired uniform-price matching.

Algorithm 1 Uniform Matching UM.

```
Function UM(Bids B, Asks A):
    Sort the lists B and A based on their respective competitiveness;
    return Match(B, A, ∅);
end
```

Match on (B, A, M) picks the most competitive bid b and the most competitive ask a from B and A respectively, and checks if they are tradable. If they are not tradable, it returns M, and the algorithm terminates.

Otherwise, if they are tradable, it adds a transaction between b and a with transaction quantity $q = \min\{\mathsf{qty}(a), \mathsf{qty}(b)\}$ to M, reduces the quantity of a and b by q in the lists B and A. Note that at least one of b or a must be fully exhausted and removed completely from B or A. It then recursively calls Match on (B, A, M).

Clearly UM takes $O(n \log n)$ time, whereas Match takes $O(n)$ time, where $n = |A| + |B|$.

The algorithm can be simply described as first sorting the lists B and A based on competitiveness, and then doing a top-down greedy matching as long as the current bid and ask are tradable.

Algorithm 2 The Match subroutine.

```
Function Match(Bids B, Asks A, Matching M) // Initially, M = ∅.:
    if |B| = 0  or  |A| = 0 then
        return M;
    b ← pop(B);
    a ← pop(A);
    if price(b) < price(a) then
        push(B, b);
        return Match(B, A, M);
    // Otherwise, b and a are tradable
    q ← min{qty(a), qty(b)};
    push(M, {(id(b), id(a), q, price(a))});
    if qty(b) − q > 0 then
        push(B, (id(b), timestamp(b), qty(b) − q, price(b)));
    if qty(a) − q > 0 then
        push(A, (id(a), timestamp(a), qty(a) − q, price(a)));
    return Match(B, A, M);
end
```

The following result states that the above result is correct.

▶ **Theorem 13** (Proved in [9, 4]). UM(B, A) *outputs a uniform price matching.*

We give a brief intuition of the proof of the above result. Let $M = \texttt{UM}(B, A)$. To see that M is a fair matching is trivial, as it starts matching the bids and asks in the decreasing order of competitiveness. Similarly, it is easy to convince oneself that M has a uniform price; any price between the limit prices of the last paired bid and ask is acceptable to all orders that participate in the matching; this again follows from the fact that matching is done top down in the order of decreasing competitiveness. Finally, to see that M has maximum volume needs some work: fix an optimal matching OPT. We can gradually transform OPT into M without altering its volume. The illuminating case to consider is when the most competitive orders b and a are fully traded in M. Note that the transaction quantity between b and a in M is $q = \min\{\mathsf{qty}(b), \mathsf{qty}(a)\}$. Since OPT is fair, b and a must also be fully traded in OPT. In particular $\mathsf{Qty}(b, \mathsf{OPT}) \geq q$ and $\mathsf{Qty}(a, \mathsf{OPT}) \geq q$. Now, if the transaction quantity between b and a in OPT is strictly less than q, then OPT can be modified by making the transaction quantity between b and a equal to q. Let us say b is matched with a' and a is matched with b' in OPT, then we can reduce the quantity of these transactions by a unit quantity, increase the transaction quantity between b and a by unit quantity, and increase the transaction quantity between b' and a' by a unit quantity (this is possible since every matched bid is tradable with every matched ask as all the transaction prices are identical). Note that in the end, $\mathsf{Vol}(\mathsf{OPT})$ remains the same. We can repeat this surgery over and over again till the matched quantity between b and a in OPT becomes equal to q. We remove this transaction from both OPT and M and apply the same argument repeatedly till $M = \mathsf{OPT}$.

3.2 Useful Lemmas and Subroutines

Before we proceed to describe our improved algorithm, we establish certain lemmas and subroutines that will be useful in describing our algorithm and proving its correctness.

We first need some definitions.

▶ **Definition 14** (Split, Range). *For a set of bids or a set of asks* Ω *and an element* $\omega \in \Omega$, $\mathsf{Split}(\Omega, \omega)$ *returns a partition of* $\Omega = (\Omega_{\succeq}, \Omega_{\prec})$, *where* $\Omega_{\succeq} = \{x \in \Omega \mid x \succeq \omega\}$ *and* $\Omega_{\prec} = \{x \in \Omega \mid x \prec \omega\}$. *Thus,* Split *splits* Ω *into two parts, one containing orders that are at least as competitive as* ω, *and the other containing orders that are less competitive than* ω. *Clearly,* Split *can be implemented in linear time.*

We now define the range of an order in a set of orders Ω *which are all bids or all asks. Let* ω *be an order in* Ω. *Let* $\mathsf{Split}(\Omega, \omega) = (\Omega_{\succeq}, \Omega_{\prec})$. *Let* $\Omega_{\succ} = \Omega_{\succeq} \setminus \{\omega\}$. $\mathsf{Range}_{\Omega}(\omega) = \{x \in \mathbb{N} \mid \mathsf{Vol}(\Omega_{\succ}) < x \leq \mathsf{Vol}(\Omega_{\succeq})\}$. *If all orders in* Ω *have unit quantities, then the* Range_{Ω} *of the* i^{th} *most competitive order is the singleton* $\{i\}$. *In general, range of the most competitive order* ω_1 *is the set* $\{1, \cdots, \mathsf{qty}(\omega_1)\}$. *The range of the next most competitive order* ω_2 *is* $\{\mathsf{qty}(\omega_1) + 1, \cdots, \mathsf{qty}(\omega_1) + \mathsf{qty}(\omega_2)\}$, *and so on. Thus, the ranges of orders in* Ω *partition* $[\mathsf{Vol}(\Omega)]$.

Observe the following fact about the previous algorithm `UM`.

▶ **Lemma 15.** *If a bid* $b \in B$ *and an ask* $a \in A$ *are matched in the matching output by* $\mathtt{UM}(B, A)$, *then* $\mathsf{Range}_B(b) \cap \mathsf{Range}_A(a) \neq \emptyset$.

This is trivial to see when all orders in $B \cup A$ have unit quantities since `UM` matches the i^{th} most competitive bid with the i^{th} most competitive ask for all $i \leq \mathsf{Vol}(M)$, where $M = \mathtt{UM}(B, A)$, and the Range of the i^{th} most competitive bid (and ask) is the singleton $\{i\}$. To see why the general statement is true observe that whenever $\mathtt{Match}(\hat{B}, \hat{A}, \hat{M})$ is called it potentially matches the most competitive bid $b \in \hat{B}$ with the most competitive ask $a \in \hat{A}$, and $\mathsf{Vol}(\hat{M}) + 1 \in \mathsf{Range}_B(b) \cap \mathsf{Range}_A(a)$.

Our main workhorse is the Select subroutine which works in linear time by employing the classical algorithm of [1]. Given a list of orders (all bids or all asks) Ω and a number $t \leq |\Omega|$, $\mathsf{Select}(\Omega, t)$ outputs the t^{th} most competitive order in Ω.

We also use a weighted version of Select called SelectQ which takes as input Ω and a quantity $q \leq \mathsf{Vol}(\Omega)$ and outputs the unique element $\omega \in \Omega$ such that $q \in \mathsf{Range}_{\Omega}(\omega)$. SelectQ can be implemented in linear time using the Select algorithm as subroutine as described below.

Algorithm 3 Weighted Selection where weights are quantities.

```
Function SelectQ(Orders Ω, q):
    ω ← Select(Ω, ⌈|Ω|/2⌉);
    (Ω_⪰, Ω_≺) ← Split(Ω, ω);
    if Vol(Ω_⪰) − qty(ω) < q ≤ Vol(Ω_⪰) then
        return ω;
    if q ≤ Vol(Ω_⪰) − qty(ω) then
        return SelectQ(Ω_⪰, q);
    if q > Vol(Ω_⪰) then
        return SelectQ(Ω_≺, q − Vol(Ω_⪰));
end
```

We get the following recurrence relationship on the running time of SelectQ: $T(n) \leq T(\lceil n/2 \rceil + 1) + O(n)$, which yields $T(n) = O(n)$, where $n = |\Omega|$.

The next subroutine we consider is $\mathsf{Split}(\Omega, \omega)$, which partitions Ω into two parts consisting of orders in Ω which are at least as competitive as ω, and orders which are strictly less competitive than ω. We now want to define a subroutine SplitQ which takes as input a list of orders Ω and a quantity $q \leq \mathsf{Vol}(\Omega)$. We want to "partition" Ω into two parts so that the volume of the more competitive part is precisely q. For this, we first find the element $\omega \in \Omega$ such that $q \in \mathsf{Range}_\Omega(\omega)$. We then $\mathsf{Split}(\Omega, \omega)$ to obtain $(\Omega_\succeq, \Omega_\prec)$. If $\mathsf{Vol}(\Omega_\succeq) = q$, then we are immediately done. Else $\mathsf{Vol}(\Omega_\succeq) > q$ and $\mathsf{Vol}(\Omega_\succeq \setminus \{\omega\}) < q$. Thus we must break a part of ω and put it in $\Omega_\prec$ so that $\mathsf{Vol}(\Omega_\succeq)$ is precisely q. This can be achieved in linear time by the following subroutine $\mathsf{SplitQ}(\Omega, q)$. Apart from outputting the partition as described above, it also returns the order ω. SplitQ clearly runs in linear time.

Algorithm 4 Splitting an order list by a particular quantity.

```
Function SplitQ(Orders Ω, quantity q) // Promise:  q ≤ Vol(Ω):
    ω ← SelectQ(Ω, q);
    (Ω_⪰, Ω_≺) ← Split(Ω, ω);
    Ω_⪯ ← Ω_≺;
    q_extra ← Vol(Ω_⪰) − q;
    if q_extra > 0 then
        Ω_⪰ ← (Ω_⪰ \ {ω}) ∪ {(id(ω), timestamp(ω), qty(ω) − q_extra, price(ω))};
        Ω_⪯ ← Ω_≺ ∪ {(id(ω), timestamp(ω), q_extra, price(ω))};
    return (ω, Ω_⪰, Ω_⪯);
end
```

We are now ready to state another lemma regarding the previous algorithm UM.

► **Lemma 16.** *Let $\hat{b} \in B$ be the t^{th} most competitive bid that gets completely traded in $M = \mathtt{UM}(B, A)$. Let $\mathsf{Split}(B, \hat{b}) = (B_\succeq, B_\prec)$ and $q = \mathsf{Vol}(B_\succeq)$. Let $\mathsf{SplitQ}(A, q) = (\hat{a}, A_\succeq, A_\preceq)$. Then, M can be partitioned into (M_1, M_2) such that $M_1 = \mathtt{UM}(B_\succeq, A_\succeq)$ and $M_2 = \mathtt{UM}(B_\prec, A_\preceq)$.*

The above statement is easy to check. Since $\hat{b}$ gets completely traded in M, all orders in $B_\succeq$ must get traded with the most competitive asks in A whose total quantity is q, which is precisely the set $A_\succeq$. Let M_1 consist of the transactions produced while matching orders in $B_\succeq$. Then M_2 will be obtained by running the UM on $(B_\prec, A_\preceq)$.

Note that a symmetric statement holds where we start with the assumption that the t^{th} most competitive ask gets traded in M.

We are now ready to describe our improved algorithm.

3.3 Improved Algorithm

Our improved algorithm Uniform takes as input a list of bids B and a list of asks A. It immediately invokes $\mathsf{Uniform}_{\mathsf{bid}}$ on $(B, A, M = \emptyset)$. M will grow into the final matching output by the algorithm.

Algorithm 5 Efficient Uniform Algorithm.

```
Function Uniform(Bids B, Asks A):
    Uniform_bid(B, A, ∅);
end
```

There are two main subroutines of our algorithm $\mathsf{Uniform}_{\mathsf{bid}}$ and $\mathsf{Uniform}_{\mathsf{ask}}$ which are symmetric in nature and they alternatively call each other. This is done to ensure that in two successive return calls, the problem size, i.e., $|B| + |A|$, reduces by a factor of two, as $\mathsf{Uniform}_{\mathsf{bid}}$ halves the number of the bids, whereas $\mathsf{Uniform}_{\mathsf{ask}}$ halves the number of the asks. So just understanding $\mathsf{Uniform}_{\mathsf{bid}}$ will be sufficient for understanding our algorithm.

On receiving (B, A, M), $\mathsf{Uniform}_{\mathsf{bid}}$ first finds the median bid b by invoking $\mathsf{Select}(B, \lceil \frac{B}{2} \rceil)$. It then splits B into two halves $(B_{\succeq}, B_{\prec})$ based on the median bid b by invoking $\mathsf{Split}(B, b)$. Let $q = \mathsf{Vol}(B_{\succeq})$. It then finds the element $a \in A$ such that $q \in \mathsf{Range}_A(a)$ by calling $\mathtt{SplitQ}(A,q)$ and splits A into its most competitive and least competitive asks $(A_{\succeq}, A_{\preceq})$ such that $\mathsf{Vol}(A_{\succeq}) = q$ by applying $\mathsf{SplitQ}(A, q)$.

After that, the algorithm checks if b and a are tradable. Two cases arise. If b and a are tradable, then every order in $B_{\succeq}$ and $A_{\succeq}$ are tradable and they will exhaustively be matched to each other to produce a matching of volume q in linear time using the `Match` subroutine. $\mathsf{Uniform}_{\mathsf{bid}}$ adds this Vol q matching to M and recursively calls $\mathsf{Uniform}_{\mathsf{ask}}$ on $(B_{\prec}, A_{\preceq}, M)$. Intuitively, the previous algorithm would also match all orders in $B_{\succeq}$ and $A_{\succeq}$ exhaustively to each other to produce a matching of volume q and proceed to matching orders in $B_{\prec}$ and $A_{\preceq}$.

In the case b and a are not tradable, then $\mathsf{Uniform}_{\mathsf{bid}}$ discards $(B_{\prec}, A_{\preceq})$ and calls $\mathsf{Uniform}_{\mathsf{ask}}$ on $(B_{\succeq}, A_{\succeq}, M)$. Intuitively, the previous algorithm will halt (i.e., produce its last transaction) before it even comes down to examining orders in $B_{\succeq}$ and $A_{\succeq}$.

Algorithm 6 Uniform Matching by bisecting bids.

```
Function Uniform_bid(Bids B, Asks A, Matching M):
    if |B| = 0 or |A| = 0 or (B = {b} and A = {a} and price(a) > price(b)) then
        return M;
    b ← Select(B, ⌈|B|/2⌉);
    (B_⪰, B_≺) ← Split(B, b);
    (a, A_⪰, A_⪯) ← SplitQ(A, Vol(B_⪰));
    if price(a) ≤ price(b) then
        M ← M ∪ Match(B_⪰, A_⪰);
        return Uniform_ask(B_≺, A_⪯, M);
    // Otherwise, b and a are not tradable
    return Uniform_ask(B_⪰, A_⪰, M);
end
```

Algorithm 7 Uniform Matching by bisecting asks.

```
Function Uniform_ask(Bids B, Asks A, Matching M):
    if |B| = 0 or |A| = 0 or (B = {b} and A = {a} and price(a) > price(b)) then
        return M;
    a ← Select(A, ⌈|A|/2⌉);
    (A_⪰, A_≺) ← Split(A, a);
    (b, B_⪰, B_⪯) ← SplitQ(B, Vol(A_⪰));
    if price(a) ≤ price(b) then
        M ← M ∪ Match(B_⪰, A_⪰);
        return Uniform_bid(B_⪯, A_≺, M);
    // Otherwise, b and a are not tradable
    return Uniform_bid(B_⪰, A_⪰, M);
end
```

$\mathsf{Uniform}_{\mathsf{ask}}$ is similar to $\mathsf{Uniform}_{\mathsf{bid}}$. Observe that, as opposed to the earlier $O(n \log n)$-time algorithm, the `Match` subroutine is not being provided sorted lists of bids and asks, but instead it is given a list of bids and asks of equal volume where each bid and each ask are tradable, and consequently it produces an exhaustive matching. Having described our algorithm $\mathsf{Uniform}$, we now turn to prove its correctness. The main theorem that establishes the correctness is as follows.

▶ **Theorem 17.** *Given a list of bids B and a list of asks A, let* $\mathsf{OPT} = \mathtt{UM}(B, A)$ *and* $M = \mathsf{Uniform}(B, A)$*. Then, for each order* $w \in B \cup A$, $\mathsf{Qty}(w, \mathsf{OPT}) = \mathsf{Qty}(w, M)$*.*

Once we prove the above theorem, the correctness follows from the following proposition and Theorem 13.

▶ **Proposition 18.** *If M_1 is a uniform price matching over (B, A) and M_2 is a matching over (B, A) such that for all $w \in B \cup A$,* $\mathsf{Qty}(w, M_1) = \mathsf{Qty}(w, M_2)$*, then M_2 is a uniform price matching.*

The proposition is obvious: the volumes of M_1 and M_2 must be the same from the condition above. Also, since the same orders participate in both M_1 and M_2, transactions in M_2 can be assigned the same uniform price that is in M_1. Finally, fairness also follows immediately since the more competitive orders are fully traded in M_1, they must be fully traded in M_2 as $\mathsf{Qty}(w, M_1) = \mathsf{Qty}(w, M_2)$ for all orders w.

We now turn to proving Theorem 17.

Proof of Theorem 17. For a list of orders Ω, we use $\Omega\downarrow$ to denote the list obtained by sorting Ω by decreasing competitiveness.

We make the following claim.

▷ Claim 19. Let B be a list of bids, and A be a list of asks, with $\mathsf{Vol}(B) = \mathsf{Vol}(A)$, and let M be a matching over (B, A). Let $M_1 = \mathtt{Match}(B\downarrow, A\downarrow, M)$, $M_2 = \mathsf{Uniform}_{\mathsf{bid}}(B, A, M)$, and $M_3 = \mathsf{Uniform}_{\mathsf{ask}}(B, A, M)$. Then, for all $\omega \in B \cup A$, $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_2) = \mathsf{Qty}(\omega, M_3)$.

If the claim is true, then Theorem 17 follows immediately by observing that without loss of generality we had assumed that our inputs B and A are such that $\mathsf{Vol}(B) = \mathsf{Vol}(A)$ (which was achieved by adding a dummy untradable order), $\mathtt{UM}(B, A) = \mathtt{Match}(B\downarrow, A\downarrow, \emptyset)$, and $\mathsf{Uniform}(B, A) = \mathsf{Uniform}_{\mathsf{bid}}(B, A, \emptyset)$.

We now prove the claim by induction on $|B| + |A|$.

We focus on showing the following part: for all $\omega \in B \cup A$, $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_2)$. A symmetric argument will yield $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_3)$.

The base cases include $|B| = |A| = 0$ and $|B| = |A| = 1$. The proof in these cases follows easily.

Thus, we are left with cases where $|B| \geq 1$, $|A| \geq 1$, and $|B| + |A| \geq 3$. Now, we argue that it suffices to consider cases where $|B| \geq 2$. Let us analyze what happens when we run $\mathsf{Uniform}_{\mathsf{bid}}$ on (B, A, M), where $|B| = 1$ and $|A| \geq 2$. $\mathsf{Uniform}_{\mathsf{bid}}$ will compute $B_{\succeq} = B$ and $A_{\succeq} = A$ (as B is a singleton and $\mathsf{Vol}(B) = \mathsf{Vol}(A)$). $\mathsf{Uniform}_{\mathsf{bid}}$ checks whether the "median" bid b (the only bid in B) and the ask a (the least competitive ask in A) are tradable or not. If they are tradable, then each order in $\omega \in B \cup A$ will be exhaustively matched. Since it is a uniform price matching, every bid-ask pair is tradable, so $\mathtt{Match}$ on $(B{\downarrow}, A{\downarrow}, M)$ will also match every order in $B \cup A$ exhaustively, and the claim follows easily. If they are not tradable, then $\mathsf{Uniform}_{\mathsf{bid}}$ will return $\mathsf{Uniform}_{\mathsf{ask}}(B, A, M)$, i.e., $\mathsf{Uniform}_{\mathsf{bid}}(B, A, M) = \mathsf{Uniform}_{\mathsf{ask}}(B, A, M)$, and in this case $|A| \geq 2$ and will be handled when we apply the symmetric argument to prove $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_3)$.

Thus, we may assume that $|B| \geq 2$ and this will imply that both $B_{\succeq}$ and $B_{\prec}$ will turn out to be proper subsets of B when we run $\mathsf{Uniform}_{\mathsf{bid}}$ on (B, A, M).

We fix sets B, A, and M such that $|B| \geq 2$. Also, $M_1 = \mathtt{Match}(B{\downarrow}, A{\downarrow}, M)$ and $M_2 = \mathsf{Uniform}_{\mathsf{bid}}(B, A, M)$. We need to show that for all $\omega \in B \cup A$, $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_2)$.

In the proof below, we will be using the following facts that hold for arbitrary B', A', and M'.

- $\mathtt{UM}(B', A') = \mathtt{Match}(B'{\downarrow}, A'{\downarrow}, \emptyset)$;
- $\mathtt{Match}(B', A', M') = M' \cup \mathtt{Match}(B', A', \emptyset)$;
- $\mathsf{Uniform}_{\mathsf{bid}}(B', A', M') = M' \cup \mathsf{Uniform}_{\mathsf{bid}}(B', A', \emptyset)$;
- $\mathsf{Uniform}_{\mathsf{ask}}(B', A', M') = M' \cup \mathsf{Uniform}_{\mathsf{ask}}(B', A', \emptyset)$, where the union is a disjoint union (which is a list concatenation operation when thinking of the matchings as lists).

$\mathsf{Uniform}_{\mathsf{bid}}$ on (B, A, M) first finds the "median" bid b, and ask a obtained by running $\mathtt{SplitQ}(A, \mathsf{Vol}(B_{\succeq}))$, and the partitions $(B_{\succeq}, B_{\prec})$ of B, $(A_{\succeq}, A_{\preceq})$ of A. Let $q = \mathsf{Vol}(B_{\succeq}) = \mathsf{Vol}(A_{\succeq})$. $\mathsf{Uniform}_{\mathsf{bid}}$ then checks whether b and a are tradable which gives rise to two cases.

Case: b and a are tradable. In this case, $B_{\succeq}$ is matched completely with $A_{\succeq}$ to produce a matching M' with quantity q and the final output matching is

$$M_2 = \mathsf{Uniform}_{\mathsf{ask}}(B_{\prec}, A_{\preceq}, M \cup M') = M \cup M' \cup \mathsf{Uniform}_{\mathsf{ask}}(B_{\prec}, A_{\preceq}, \emptyset).$$

Also, $\mathtt{Match}(B{\downarrow}, A{\downarrow}, M) = M \cup \mathtt{Match}(B{\downarrow}, A{\downarrow}, \emptyset) = M \cup \mathtt{UM}(B, A)$. We now invoke Lemma 16 by setting $\hat{b}$ to b to argue that the matching output by $\mathtt{UM}$ on (B, A) is $\mathtt{UM}(B_{\succeq}, A_{\succeq}) \cup \mathtt{UM}(B_{\prec}, A_{\preceq})$. Thus,

$$M_1 = M \cup \mathtt{UM}(B_{\succeq}, A_{\succeq}) \cup \mathtt{UM}(B_{\prec}, A_{\preceq}) = M \cup \mathtt{UM}(B_{\succeq}, A_{\succeq}) \cup \mathtt{Match}(B_{\prec}{\downarrow}, A_{\preceq}{\downarrow}, \emptyset).$$

Note that we have expressed both M_1 and M_2 as a disjoint union (list concatenation) of three sets. Fix an $\omega \in A \cup B$. Now, $\mathsf{Qty}(\omega, M) = \mathsf{Qty}(\omega, M)$ (trivially), $\mathsf{Qty}(\omega, M') = \mathsf{Qty}(\omega, \mathtt{UM}(B_{\succeq}, A_{\succeq}))$ (as M' is obtained by exhaustively matching all orders in $(B_{\succeq}, A_{\succeq})$), and $\mathsf{Qty}(\omega, \mathsf{Uniform}_{\mathsf{ask}}(B_{\prec}, A_{\preceq}, \emptyset)) = \mathsf{Qty}(\omega, \mathtt{Match}(B_{\prec}{\downarrow}, A_{\preceq}{\downarrow}, \emptyset))$ (from induction). Thus, we have $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_2)$.

Case: b and a are not tradable. $\mathsf{Uniform_{bid}}$ completely discards $B_\prec$ and $A_\preceq$ and outputs the matching

$$M_2 = \mathsf{Uniform_{ask}}(B_\succeq, A_\succeq, M) = M \cup \mathsf{Uniform_{ask}}(B_\succeq, A_\succeq, \emptyset).$$

Also, $M_1 = \mathtt{Match}(B\downarrow, A\downarrow, M) = M \cup \mathtt{Match}(B\downarrow, A\downarrow, \emptyset)$. Observe that no bid in $B_\prec$ is tradable with any ask in $A_\preceq$, as bids in $B_\prec$ are strictly less competitive than b and asks in $A_\preceq$ are at most as competitive as a, and b and a are not tradable. We further claim that $\mathtt{Match}(B\downarrow, A\downarrow, \emptyset) = \mathtt{UM}(B, A)$ does not match any orders from $B_\prec \cup A_\preceq$. To see this, we invoke Lemma 15. Note that except for bid b and ask a, the respective ranges of orders in $B_\succeq$ and $A_\succeq$ have numbers strictly less than q and the respective ranges of orders in $B_\prec$ and $A_\preceq$ have numbers that are all strictly greater than q, as $q \in \mathsf{Range}_A(a) \cap \mathsf{Range}_B(b)$. Thus, any potential matches between $B_\prec$ and $A_\succeq$ or between $A_\preceq$ and $B_\succeq$ can only happen between b and a, but they are not tradable (as per the case). Thus, we conclude that the $\mathtt{UM}(B, A) = \mathtt{UM}(B_\succeq, A_\succeq)$. Therefore,

$$M_1 = M \cup \mathtt{Match}(B_\succeq\downarrow, A_\succeq\downarrow, \emptyset).$$

From induction, arguing as before, we get for all orders $\omega \in A \cup B$, $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_2)$, and we are done. ◀

We now analyze the running time of $\mathsf{Uniform}$. Let $T(n)$ represent the running time of $\mathsf{Uniform}$, where the number of orders $|B| + |A| = n$.

$\mathsf{Uniform}$ calls $\mathsf{Uniform_{bid}}$, which in turn calls $\mathsf{Uniform_{ask}}$ after decreasing the number of the bids by a factor of two. $\mathsf{Uniform_{ask}}$ then calls $\mathsf{Uniform_{bid}}$ again after decreasing the number of the asks by a factor of two. So after two successive returns, we can see that the number of bids and asks decreases by a factor of two. Also, since all the subroutines take linear time, by simple inspection, we conclude

$T(n) \le T(\lceil \frac{n}{2} \rceil + 1) + cn$, where c is an absolute constant. Thus, clearly $T(n) = O(n)$.

This completes the proof of Theorem 7.

4 Conclusions

The problems we consider are clearly of fundamental interest and we achieve asymptotically tight results for them using elementary techniques. Surprisingly, despite their fundamental nature and wide practical applicability, prior to this work, the complexity aspects of such problems were not deeply studied. The following natural questions arise from our work.

- In this work, the most classical exchange model is assumed; there are execution principles other than price-time priority (like pro-rata matching) which are also being employed in the real world. These alternative principles present opportunities for studying algorithmic complexity beyond the traditional price-time priority model.
- Furthermore, it might be interesting to consider similar problems in the context of decentralized exchanges.
- Finally, bridging the gap between the upper and lower bounds on the time complexity of Problem 1 in the binary query model remains open.

References

1 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

2 Ravi B. Boppana. The decision-tree complexity of element distinctness. *Inf. Process. Lett.*, 52(6):329–331, 1994. doi:10.1016/0020-0190(94)00154-5.

3 Kaustubh Deshmukh, Andrew V. Goldberg, Jason D. Hartline, and Anna R. Karlin. Truthful and competitive double auctions. In Rolf H. Möhring and Rajeev Raman, editors, *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*, volume 2461 of *Lecture Notes in Computer Science*, pages 361–373, Rome, Italy, 2002. Springer. doi:10.1007/3-540-45749-6_34.

4 Mohit Garg, N. Raja, Suneel Sarswat, and Abhishek Kr Singh. Double auctions: Formalization and automated checkers. *J. Autom. Reason.*, 69(3):17, 2025. doi:10.1007/S10817-025-09732-X.

5 Mohit Garg and Suneel Sarswat. The design and regulation of exchanges: A formal approach. In Anuj Dawar and Venkatesan Guruswami, editors, *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022, December 18-20, 2022, IIT Madras, Chennai, India*, volume 250 of *LIPIcs*, pages 39:1–39:21, Chennai, India, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2022.39.

6 Mohit Garg and Suneel Sarswat. Efficient and verified continuous double auctions. In Nikolaj S. Bjørner, Marijn Heule, and Andrei Voronkov, editors, *LPAR 2024 Complementary Volume, Port Louis, Mauritius, May 26-31, 2024*, volume 18 of *Kalpa Publications in Computing*, pages 1–13, Port Louis, Mauritius, 2024. EasyChair. doi:10.29007/92JT.

7 Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching.* Monographs in Theoretical Computer Science. An EATCS Series. Springer Berlin, Heidelberg, Berlin, Heidelberg, 1 edition, 1984. doi:10.1007/978-3-642-69672-5.

8 L. Mirsky. A dual of dilworth's decomposition theorem. *The American Mathematical Monthly*, 78(8):876–877, 1971. URL: http://www.jstor.org/stable/2316481.

9 Raja Natarajan, Suneel Sarswat, and Abhishek Kr Singh. Verified double sided auctions for financial markets. In Liron Cohen and Cezary Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference)*, volume 193 of *LIPIcs*, pages 28:1–28:18, Rome, Italy, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.ITP.2021.28.

10 Jinzhong Niu and Simon Parsons. Maximizing matching in double-sided auctions. In Maria L. Gini, Onn Shehory, Takayuki Ito, and Catholijn M. Jonker, editors, *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 1283–1284, Saint Paul, MN, USA, 2013. IFAAMAS. URL: http://dl.acm.org/citation.cfm?id=2485184.

11 Jaikumar Radhakrishnan. Entropy and counting. *Computational mathematics, modelling and algorithms*, 146, 2003.

12 Peter R. Wurman, William E. Walsh, and Michael P. Wellman. Flexible double auctions for electronic commerce: theory and implementation. *Decis. Support Syst.*, 24(1):17–27, 1998. doi:10.1016/S0167-9236(98)00060-8.

13 Dengji Zhao, Dongmo Zhang, Md Khan, and Laurent Perrussel. Maximal matching for double auction. In Jiuyong Li, editor, *AI 2010: Advances in Artificial Intelligence - 23rd Australasian Joint Conference, Adelaide, Australia, December 7-10, 2010. Proceedings*, volume 6464 of *Lecture Notes in Computer Science*, pages 516–525, Adelaide, Australia, 2010. Springer. doi:10.1007/978-3-642-17432-2_52.