Optimal Antimatroid Sorting

Benjamin Aram Berendsohn
Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

The classical *comparison-based sorting* problem asks us to find the underlying total ordering of a given set of elements, where we can only access the elements via comparisons. In this paper, we study a *restricted* version, where, as a hint, a set T of possible total orderings is given, usually in some compressed form.

Recently, an algorithm called *topological heapsort* with optimal running time was found for case where T is the set of topological orderings of a given directed acyclic graph, or, equivalently, T is the set of linear extensions of a partial ordering [Haeupler et al. 2024]. We show that a simple generalization of topological heapsort is applicable to a much broader class of restricted sorting problems, where T corresponds to a given *antimatroid*.

As a consequence, we obtain optimal algorithms for the following restricted sorting problems, where the allowed total orders are ...

- ... restricted by a given set of *monotone precedence formulas*;
- ... the *perfect elimination orders* of a given chordal graph; or
- ... the possible *vertex search orders* of a given connected rooted graph.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases sorting, working-set heap, greedy, antimatroid

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.104

1 Introduction

One of the fundamental problems in theoretical computer science is *sorting*. It has a wide variety of applications in practice [24] and is frequently used as an introduction to design and analysis of algorithms.

Perhaps the most intensely studied variant is *comparison-based sorting*, where one can only access the input elements via comparisons (otherwise, we assume the standard word RAM model). This excludes bucket sort, radix sort, and other integer-sorting techniques [10, 16, 15]. The well-known *information-theoretic bound (ITB)* states that comparison-based sorting requires $\log(n!) \approx n \log n$ comparisons in the worst case, and many algorithms asymptotically matching this bound are known.

In this paper, we study a restricted variant of comparison-based sorting. In addition to the n elements to be sorted, we are give a set T of possible total orders, with the guarantee that the actual order of the input is contained in T. Call this the *T-sorting problem.* Note that T may be very large compared to n, so it makes sense to consider variants of the problem where T is given in some compressed form. Regardless of that, the ITB for the T-sorting problem is $\log|T|$, so perhaps the first question to ask is: Can we solve the T-sorting problem with only $\mathcal{O}(\log|T|)$ comparisons?

In turns out that the answer is *yes* if $|T| \geq 2^{\Omega(n)}$. Fredman [9] showed how to solve the T-sorting problem with $\mathcal{O}(n + \log|T|)$ comparisons. This algorithm assumes that T is given explicitly and is thus entirely impractical. Fredman also showed that there exist sets T with $|T| \in \mathcal{O}(n)$ where the T-sorting problem requires $\Omega(n)$ comparisons, which means the ITB is not always tight.

Most later work on this problem has been focused on the special case where T is the set of linear extensions of a partial order P, given by its Hasse diagram. It is known that $\mathcal{O}(\log|T|)$ comparisons are sufficient, that is, the ITB is tight (even when $\log T \leq n$). After decades of research [21, 9, 29, 19, 2, 5, 33], eventually, two algorithms were found that are simultaneously comparison-optimal and running-time optimal [14, 32]. Both algorithms actually solve the slightly more general *DAG sorting* problem: Given is a directed acyclic graph G, and T is taken as the set of topological orderings of G. Note that each DAG G defines a partial order P (though G may contain some "unnecessary" edges), and T is the set of linear extensions of P.

Most important for our purposes is the *topological heapsort* algorithm of Haeupler, Hladík, Iacono, Rozhon, Tarjan, and Tětek [14]. This was the first DAG-sorting algorithm with optimal running time $\mathcal{O}(|V(G)| + |E(G)| + \log|T|)$, though it makes $\Theta(|V(G)| + \log|T|)$ comparisons, which is non-optimal if $|T|$ is small. Achieving comparison optimality requires further modifications, which we discuss later.

We briefly describe topological heapsort. Let G be the given DAG. At the beginning, identify the set S of sources (vertices of in-degree 0) of G, and insert them into a priority queue Q. In each following step, we first find and remove the minimum element x from Q. Then, we remove x from G. This may create several new sources, which we all insert into Q. Repeat until Q is empty. It is easy to see that this algorithm is *correct* if the underlying total order of the vertices is indeed a topological ordering of G. Haeupler et al. achieve optimal running time by using a special priority queue with the so-called *working-set property.*

Let us now try to generalize topological heapsort to solve the general T-sorting problem. Suppose that instead of the graph G, we are given a black-box data structure D that reports all elements that are the minimum of at least one ordering $\tau \in T$. After "removing" the first k elements $x_1, x_2, \dots, x_k$, let D report all elements y such that $x_1 x_2 \dots x_k y$ is a prefix of some ordering $\tau \in T$ (the *available* elements). Again, it is easy to see that topological heapsort with this data structure is *correct.* But is it optimal?

The main contribution of this paper (Section 3) is to show that topological heapsort has optimal running time if T corresponds to a class of structures called *antimatroids*, which are a generalization of partial orders, when we disregard the running time for the data structure D. We additionally show that, for several possible representations of antimatroids, we can implement D such that its total running time is linear in the size of the input, so the running time stays optimal (Section 4). Finally, we show how to additionally achieve comparison optimality (Section 5). The full version of the paper contains various omitted proofs, as well as a discussion of several generalizations of antimatroids where topological heapsort is *not* optimal. In the remainder of this section, we give an informal definition of antimatroids, and discuss some related work.

Antimatroids. Fix a set T of total orderings, and consider the black-box data structure D mentioned above. Recall that, for each sequence $x_1, x_2, \dots, x_k$ of $k \geq 0$ elements, D tells us the set of *available* elements y such that $x_1 x_2 \dots x_k y$ is a prefix of some $\tau \in T$. It is clear that any set T can be described in this way. Intuitively, T is an *antimatroid* if the following conditions apply. First, once an element becomes available, it stays available until it is chosen (availability is *monotone*); and second, availability of an element can only depend on the *set* of elements chosen so far (but not their order).

The first condition can be nicely related to topological heapsort: It means that our priority queue Q at no point contains unavailable elements. The second condition may seem more surprising, but it turns out to be also necessary for optimality of topological heapsort and tightness of the ITB (for details, see the full version of the paper).

Antimatroids were first studied by Dilworth [7] in lattice theory. Later, they were found to be special cases of so-called *greedoids* [25, 26], which are structures admitting a certain greedy optimization procedure. A common alternative definition of antimatroids treats them as *set systems*; this definition can be related to *matroids* (hence the name), which are also special cases of greedoids. We refer to the surveys of Björner and Ziegler [1] and Korte, Schrader, and Lovász [27] for more.

Related work. Optimal algorithms are known [17, 4, 29] for *merging* two sorted lists; this corresponds to a partial order consisting of exactly two disjoint chains. Merging can be generalized to *multiway merging*, where the input is split into any number of sorted lists; here, the ITB is known to be related to the *Shannon entropy* of the list lengths. Multiway merging is important for pratical sorting algortihms that exploit *runs* in a given unsorted sequence (see, e.g., Gelling, Nebel, Smith, and Wild [11]).

A different special case of the T-sorting problem concerns the case where the set T is the permutations that *avoid* a certain pattern π.[1] It is known that $T \leq c_\pi^n$ for some constant c_π depending on π, so the ITB is linear in this case. Opler [31] gave an optimal linear-time algorithm for the pattern-avoiding sorting problem, which works even when the pattern π is unknown. We refer to his paper for more information on that problem.

The $X + Y$*-sorting problem* is a restricted sorting problem where, interestingly, the ITB is *not* tight. Here, two sets $X, Y \subset \mathbb{R}$ of size k each are given, and we need to sort the set $\{x + y \mid x \in X, y \in Y\}$ of size $n = k^2$. If only comparisons of the form $x + y < x' + y'$ with $x, x' \in X$, $y, y' \in Y$ are allowed, then this is a restricted sorting problem by our definition. It is known that the number of possible orderings is only $k^{\Theta(k)}$ [9], which makes the ITB $\Omega(k \log k) = \Omega(\sqrt{n} \log n)$; however, Fredman [9] showed that $\Omega(k^2)$ comparisons are needed. It is currently unknown if an algorithm with optimal running time $\Theta(k^2)$ exists.

Finally, a question closely related to restricted sorting is finding *balanced pairs*, which are comparisons that split the set of possible total orders into two parts of approximately equal size. There is a body of work on balanced pairs in partial orders [21, 9, 29, 2, 3], and the concept has been generalized to antimatroids [8].

2 Preliminaries

All the following definitions are taken from Björner and Ziegler [1] with only slight changes.

Words and languages. Let an *alphabet* Σ be a set of *letters*. A *word* on Σ is a finite sequence of letters. We write words without commas as $\alpha = x_1 x_2 \dots x_n$, where $x_i \in \Sigma$, and we write $\alpha\beta$ for the concatenation of two words α and β. We denote the length of a word α by $|\alpha|$. The *support* $\tilde{\alpha}$ of a word α is the set of letters contained in it, and the empty word is denoted by ε.

A word α is *simple* if each letter occurs at most once in it (i.e., $|\tilde{\alpha}| = |\alpha|$), and α is a *permutation* of Σ if each letter occurs precisely once in it. The set of all simple words on Σ is denoted by Σ_s^*, and a *simple language* is a subset $L \subseteq \Sigma_\mathrm{s}^*$. We denote the set of permutations contained in a simple language by $\mathcal{P}(L)$, and we call L *full* if $|\mathcal{P}(L)| > 0$.

[1] A permutation τ of a set X *avoids* a permutation π of $[k]$ if τ contains no subsequence that is order-isomorphic to π, according to some fixed total order on X. This restricted sorting problem is usually formulated in a different, but equivalent way where X is given as a sequence that is known to avoid a pattern.

Partial orders. A partial order on a set Σ is denoted with a capital letter, like P, and we write $\prec_P$ for the actual partial order relation. Total orders are treated as permutations of the underlying set Σ. Whenever necessary, we write $\prec_\pi$ for the total order relation corresponding to the permutation π, i.e., we have $x \prec_\pi y$ if x precedes y in π. An *oracle* for a total order π is a black-box function that answers queries of the form $x \prec_\pi y$ in constant time.

Antimatroids. An *antimatroid* (sometimes called *antimatroid language*) is a simple language $A \subseteq \Sigma_{\mathrm{s}}^*$ that satisfies the following two axioms:

(i) For all $\alpha \in \Sigma_{\mathrm{s}}^*$ and $x \in \Sigma \setminus \tilde{\alpha}$, if $\alpha x \in A$, then $\alpha \in A$. (A is closed under taking prefixes.)

(ii) For each $\alpha, \beta \in A$ with $\tilde{\alpha} \not\subseteq \tilde{\beta}$, there is some $x \in \tilde{\alpha} \setminus \tilde{\beta}$ such that $\beta x \in A$.

Axiom (i) is called *accessibility* and implies that every word in L can be *built* by starting with ε and successively appending letters, without ever leaving L. Axiom (ii) ensures the two "availability properties" of antimatroids mentioned in the introduction.

In this paper, we only consider *full* antimatroids (recall that this means $\mathcal{P}(A) \neq \emptyset$).[2] A full antimatroid A is completely determined by $\mathcal{P}(A)$; in fact, A is exactly the set of prefixes of $\mathcal{P}(A)$. However, not every set of permutations can be described by an antimatroid; for example, any antimatroid on $\Sigma = \{x, y, z\}$ that contains xyz and zyx also contains xzy.

Throughout this paper, we use a different formalism, which is closer to the informal definition of antimatroids given in the introduction. For an alphabet Σ, a *precedence function* for a letter $x \in \Sigma$ is a function $p_x \colon 2^{\Sigma \setminus \{x\}} \to \{0,1\}$ that is *monotone*, i.e., we have $p_x(X) \leq p_x(Y)$ if $X \subseteq Y \subseteq \Sigma$. A *monotone precedence system (MPS)* on Σ is a collection $S = \{p_x\}_{x \in \Sigma}$ of precedence functions. The *language* of S is the set

$$\mathcal{L}(S) = \{x_1 x_2 \ldots x_k \mid \forall i \in [k] : p_{x_i}(\{x_1, x_2, \ldots, x_{i-1}\}) = 1\}.$$

The full version of the paper contains a proof that monotone precedence systems indeed characterize (full) antimatroids. In the following, we write $\mathcal{P}(S) = \mathcal{P}(\mathcal{L}(S))$ for convenience.

▶ **Example 1.** Let P be a partial order on a set Σ. Define $S = \{p_x\}_{x \in \Sigma}$ with $p_x(X) = 1$ if and only if $\{y \in \Sigma \mid y \prec_P x\} \subseteq X$. It is easy to see that $\mathcal{P}(S)$ is exactly the set of linear extensions of P. Thus, antimatroids generalize partial orders.

▶ **Example 2.** Let $\Sigma = \{a, b, c\}$, and consider the MPS S with $p_a(X) = p_b(X) = 1$ for all $X \subseteq \Sigma$, and $p_c(X) = 1$ iff $a \in X$ or $b \in X$. Then $\mathcal{P}(S) = \{abc, bac, acb, bca\}$. Since $\mathcal{P}(S)$ contains both acb and its reverse bca, but not all possible orderings, it is *not* the set of linear extensions of any partial order. Thus, antimatroids *strictly* generalize partial orders.

Priority queues. The central data structure of topological heapsort is the *working-set priority queue*. This is a priority queue (like the well-known *binary heap* [34]) with the so-called *working-set property*. Informally, this property ensures that extracting an element is fast if that element has been inserted only shortly before. We omit the precise definitions here, since they are not needed for our proofs, and refer to Haeupler et al. [14] for more details.

[2] Some authors define antimatroids to be always full [27].

3 Generalized topological heapsort

Let us now formally state the antimatroid-sorting meta-problem. Given is a set Σ, an antimatroid A on Σ, and a comparison oracle for a permutation $\pi \in \mathcal{P}(A)$. The task is to output π (or, equivalently, output Σ in sorted order according to the oracle comparisons $\prec_\pi$). Note the assumption $\pi \in \mathcal{P}(A)$ in particular implies that A is full.

We call this a meta-problem since the precise representation of A is unspecified. We discuss some possible representations in Section 4. For now, we give a meta-algorithm for the problem (Algorithm 1), using the characterization of A as a monotone precedence system S.

Algorithm 1 Topological heapsort for antimatroids.

```
**Input:** Set $\Sigma$, MPS $S = \{p_x\}_{x\in\Sigma}$, comparison oracle for some $\pi \in \mathcal{P}(S)$.
Initialize working-set priority queue $Q$ that contains each $x \in \Sigma$ with $p_x(\emptyset) = 1$
$\alpha \leftarrow \varepsilon$
while $Q$ is not empty do
    Delete the minimum from $Q$ and append it to $\alpha$
    For each $x \in \Sigma \setminus (\tilde{\alpha} \cup Q)$, add it to $Q$ if $p_x(\tilde{\alpha}) = 1$
return $\alpha$
```

We can already show correctness of our meta-algorithm regardless of implementation details.

▶ **Lemma 3.** *Topological heapsort is correct for every input* (Σ, S, π).

Proof. Let $S = \{p_x\}_{x\in\Sigma}$ and $\pi = x_1, x_2, \ldots, x_n$. We need to show that topological heapsort produces $\alpha = \pi$.

Since $\pi \in \mathcal{P}(S)$, we have $p_{x_i}(\{x_1, x_2, \ldots, x_{i-1}\}) = 1$ for each $i \in [n]$. In particular, this means that $p_{x_1}(\emptyset) = 1$, so x_1 is contained in Q at the start. Since also $x_1 \prec_\pi x_i$ for all $i > 1$, the first element added to α is indeed x_1. Suppose now that we have $\alpha = x_1 x_2 \ldots x_{i-1}$ at the start of some iteration. Since $p_{x_i}(\tilde{\alpha}) = 1$, we have $x_i \in Q$. Since $x_i \prec_\pi x_j$ for all $j > i$, the algorithm next appends x_i to α. By induction, the algorithm indeed returns π. ◀

Two parts of the algorithm are left unspecified: How to identify the initial elements in Q, and the elements inserted in each iteration. We now define an abstract data structure for this task. A *candidate data structure* C for a monotone precedence system $S = \{p_x\}_{x\in\Sigma}$ maintains a set $X \subseteq \Sigma$, initially $X = \emptyset$, and supports the following two operations:

- C.`init`(): Set $X \leftarrow \emptyset$ and report all $x \in \Sigma$ with $p_x(\emptyset) = 1$.
- C.`step`(x): Given $x \in \Sigma \setminus X$ with $p_x(X) = 1$, report all $y \in \Sigma$ with $p_y(X) = 0$ and $p_y(X \cup \{x\}) = 1$. Then, add x to X.

Observe that C allows at most $|\Sigma|$ calls to `step` after `init`, one for each $x \in \Sigma$. Further, parameters of `step` calls form a sequence $x_1 x_2 \ldots x_k \in \mathcal{L}(S)$. The *total time* for C, written $t^{\text{total}}(C)$, is the overall time for one call to `init` and a valid sequence of `step` calls, in the worst case.

Clearly, any correct candidate data structure can be used to finish the implementation of topological heapsort. The actual implementation will heavily depend on the given representation of the antimatroid; see Section 4.

$$\begin{array}{ccccc} a & b & c & d & e \\ \parallel & \parallel & \parallel & \parallel & \parallel \\ x_1 & x_2 & x_3 & x_4 & x_5 \end{array}$$

$$\begin{array}{cccccc} Q_0 & Q_1 & Q_2 & Q_3 & Q_4 & Q_5 \\ \parallel & \parallel & \parallel & \parallel & \parallel & \parallel \\ \{a,c\} & \{b,c\} & \{c\} & \{d,e\} & \{e\} & \emptyset \end{array}$$

Figure 1 An example transcript of topological heapsort for some antimatroid on $\Sigma = \{a, b, c, d, e\}$.

3.1 Running time analysis

Since we want our analysis to be independent of the candidate data structure, we split the work done by topological heapsort into two essential parts.

- The total time $t^{\text{total}}(C)$ spend by the candidate data structure, as defined above.
- The time required to initialize Q (given the initial elements), delete the minimum from Q, and insert given elements into Q in each step. Call this the *queue time* t^{Q}.

Since the loop performs at most $n = |\Sigma|$ iterations, the running time of topological heapsort is $t^{\text{total}}(C) + t^{\text{Q}} + \mathcal{O}(n)$.

Note that the queue time does *not* depend on the representation of S at all. In the following, we show that t^{Q} is always $\mathcal{O}(n + \log|\mathcal{P}(S)|)$, by reducing to the special case where S is a *partial order*, which has been solved by Haeupler et al. [14]. The number of comparisons can be as large as the queue time, which is not optimal if $\log(|\mathcal{P}(S)|) \ll n$. In Section 5, we improve this bound to the optimal $\mathcal{O}(\log|\mathcal{P}(S)|)$ by modifying the algorithm.

Partial orders. Let G be a directed acyclic graph with n vertices and m edges, and consider the set $\mathcal{T}(G)$ of topological orderings of G. Clearly, $\mathcal{T}(G)$ is the set of linear extensions of a partial order on $V(G)$. Thus, $\mathcal{T}(G)$ is an also antimatroid in $V(G)$ (see Example 1).

It is easy to implement a candidate data structure for $\mathcal{T}(G)$ via the standard topological sorting algorithm [18, 23], with total running time $\mathcal{O}(m + n)$. Haeupler et al. [14] showed that the queue time of topological heapsort is optimal for partial orders:

▶ **Lemma 4** (Haeupler et al. [14]). *Let S be a monotone precedence system corresponding to a partial order on a set Σ, and let $\pi \in \mathcal{P}(S)$ be a total order on Σ. Then, for topological heapsort with input (Σ, S, π), we have $t^{\text{Q}} \in \mathcal{O}(|\Sigma| + \log|\mathcal{P}(S)|)$.*

General antimatroids. We now generalize Lemma 4 to antimatroids. We need the following definition. The *transcript* of a run of topological heapsort on an input (Σ, S, π) is the sequence $(Q_0, Q_1, \ldots, Q_n)$, where Q_0 is the set of elements initially in the priority queue, and Q_i for $i \in [n]$ is the set of elements in the priority queue after the i-th step. Observe that, if $\pi = x_1 x_2 \ldots x_n$, then $x_i \in Q_{i-1}$, and $x_i \notin Q_j$ for each $j \geq i$. See Figure 1 for an example.

A crucial fact that we use in the proof below is that the queue time t^{Q} *only* depends on the transcript of the run. In particular, two runs on inputs (Σ, S, π) and (Σ, S', π) that have the same transcript also have the same queue time, even if $S \neq S'$.

▶ **Theorem 5.** *Let S be a monotone precedence system on Σ, and let $\pi \in \mathcal{P}(S)$ be a total order on Σ. Then, for topological heapsort with input (Σ, S, π), we have $t^{\text{Q}} \in \mathcal{O}(|\Sigma| + \log|\mathcal{P}(S)|$.*

Proof. Consider a run of topological heapsort on (Σ, S, π) with transcript $(Q_0, Q_1, \ldots, Q_n)$, and let $\pi = x_1 x_2 \ldots x_n$.

We now define a partial order P on Σ. For each $i \in [n]$, let $x_i \prec_P y$ for each $y \in \Sigma \setminus (Q_0 \cup Q_1 \cup \dots Q_{i-1})$. Informally, we have $x \prec_P y$ if x is deleted from the priority queue before y is inserted. Let S_P be the monotone precedence system corresponding to P. Clearly, $\pi \in \mathcal{P}(S_P)$. We claim the following:

(i) The transcript of running topological heapsort on (Σ, S_P, π) is again $(Q_0, Q_1, \dots, Q_n)$.
(ii) $\mathcal{P}(S_P) \subseteq \mathcal{P}(S)$.

We first explain how this implies the theorem. By Lemma 4, running topological heapsort on (Σ, S_P, π) has queue time $\mathcal{O}(|\Sigma| + \log|\mathcal{P}(S_P)|)$. By claim (i), running topological heapsort on (Σ, S, π) has the same queue time (since the transcript is the same), which, by claim (ii), is at most $\mathcal{O}(|\Sigma| + \log|\mathcal{P}(S)|)$, as required.

We now prove claim (i). Let $(Q'_0, Q'_1, \dots, Q'_n)$ be the transcript of running topological heapsort on (Σ, S_P, π). Each $y \in Q_0$ is a minimal element of P by definition, so $Q_0 \subseteq Q'_0$. On the other hand, each $y \notin Q_0$ satisfies $x_1 \prec_P y$ by definition and thus $y \notin Q'_0$, so $Q_0 = Q'_0$.

Now take some $j \in [n]$ and assume that by induction, we have $Q_i = Q'_i$ for all $i < j$. For each $y \in Q_j$, we have $\{z \in \Sigma \mid z \prec_P y\} \subseteq \{x_1, x_2, \dots, x_j\}$, and $y \notin \{x_1, x_2, \dots, x_j\}$, so $y \in Q'_j$ by definition. On the other hand, each $y \notin Q_j$ satisfies either (a) $y \in \{x_1, x_2, \dots, x_j\}$ or (b) $y \notin Q_0 \cup Q_1 \cup \dots \cup Q_j$. (a) directly implies $y \notin Q'_j$ (y is removed earlier). (b) implies $x_{j+1} \prec_P y$, which again implies $y \notin Q'_j$.

We finish the proof with claim (ii). Let $\alpha = x'_1 x'_2 \dots x'_n \in \mathcal{P}(S_P)$. We show that $\alpha \in \mathcal{P}(S)$, i.e., that for each $i \in [n]$, we have $p_{x'_i}(\{x'_1, x'_2, \dots, x'_{i-1}\}) = 1$.

Let $y \in \Sigma$, and let k be minimal such that $y \in Q_k$. Let $X_k = \{x_1, x_2, \dots, x_k\}$. First, by definition of Q_k, we have $p_y(X_k) = 1$. Second, we have $\{x \in \Sigma \mid x \prec_P y\} = X_k$ by minimality of k. Now suppose $y = x'_i$, and let $X'_{i-1} = \{x'_1, x'_2, \dots, x'_{i-1}\}$. Then $p'_y(X'_{i-1}) = 1$, implying $X'_{i-1} \supseteq X_k$. By monotonicity, we have $p_y(X'_{i-1}) \geq p_y(X_k) = 1$, as desired. ◀

Remark. The technique of constructing an auxiliary partial order is inspired by the work of Haeupler, Hladík, Rozhoň, Tarjan, and Tětek [12] (we discuss further connections in Section 4.2). In fact, the exact partial order P in the proof of Theorem 5 appears in the revised version[3] of the paper that introduced topological heapsort [14]. They observe that P is an *interval order*, which means $|\mathcal{P}(P)|$ is relatively easy to bound and can be related to the working-set bound.

The algorithm of van der Hoog, Rotenberg, and Rutschmann [32] is also analyzed with the help of interval orders, but seemingly cannot be easily generalized to antimatroids.

4 Antimatroid representations

In this section, we consider several representations of antimatroids (some only covering special cases), and discuss how to implement candidate data structures (CDSs) for these representations. If C is a CDS for an animatroid A with some representation, then we call C *efficient* if $t^{\text{total}}(C) \in \mathcal{O}(n)$, where n is the size of the representation (in machine words). Theorem 5 implies that topological heapsort with an efficient CDS has optimal running time.

4.1 Precedence formulas

We first consider an explicit representation of general full antimatroids based on monotone precedence systems. The *precedence formula representation* of a full antimatroid A on Σ is a collection $\{F_x\}_{x \in \Sigma}$, where F_x is a monotone boolean formula on the variable set $V_x = \Sigma \setminus \{x\}$. (A *monotone* boolean formula allows only the operators $\vee$ and $\wedge$ and the constants 0 and 1, notably prohibiting the negation $\neg$.)

[3] An earlier preprint [13] used a different proof technique.

Each formula F_x represents a precedence function p_x as follows: For $Y \subseteq \Sigma$, we let $p_x(Y)$ be the value of F_x when setting each $y \in Y$ to 1, and each $y \in X \setminus (Y \cup \{x\})$ to 0. Since any monotone function f can be represented as a monotone boolean formula, this representation is suitable for every antimatroid. In the following, we assume each formula is given as a linked list of symbols with each variable, operator, constant, or parenthesis in the formula occupying one machine word. Other representations, like parsing trees and circuits, are also possible.

An CDS C for $\{F_x\}_{x \in \Sigma}$ is implemented as follows. We need two preprocessing steps. First, for each formula, we simplify it. We repeatedly replace substrings $(1 \wedge x) \to x$, $(1 \vee x) \to 1$, $(0 \wedge x) \to 0$, and so on, in linear overall time. Afterwards, we have $F_x = 1$ for each x with $p_x(\emptyset) = 1$.

Second, compute a directed graph G, where $V(G) = \Sigma$, and $(x, y) \in E(G)$ if F_y contains the variable x. For each such edge, we store a list of pointers to each occurrence of x within F_y. Note that the size of G (including pointers) is at most the total size of all formulas.

The initial candidate set reported by `init`() is the set of sources in G. `step`(x) is implemented as follows: For each out-neighbor y of x, replace each occurrence of x in F_y with 1. Then, simplify F_y as outlined above, and report y if $F_y = 1$ after the simplification. Finally, remove x from the graph. To see that the data structure is correct, observe that after calling `step`(x) on each x in some set $X \subset \Sigma$, we have $F_y = 1$ for each y with $p_y(X) = 1$.

The total running time is linear, since each edge is visited once, and simplification is linear overall. Thus, the data structure is efficient, and so:

▶ **Theorem 6.** *Given an antimatroid A on Σ in a precedence formula representation of length n, we can solve the $\mathcal{P}(A)$-sorting problem in optimal time $\mathcal{O}(n + \log |\mathcal{P}(A)|)$.*

There are two representations of antimatroids that are similar to precedence formulas: *rooted set collections* and *alternative precedence structures* [27]. These roughly correspond to precedence formulas in conjuctive normal form and disjunctive normal form, respectively. A third related characterization with so-called *elementary ranking conditions* [30] is used in linguistics (see the full version of the paper for details). In the following, we discuss two more examples that only apply to subclasses of antimatroids.

4.2 Vertex search and distance orderings

Given a connected graph G and a designated root $r \in V(G)$, the *vertex search antimatroid* on (G, r), written $A^{\mathrm{VS}}_{G,r}$, contains the empty word ε as well as each simple word $v_1 v_2 \ldots v_k$ where $v_1 = r$ and each prefix $v_1 v_2 \ldots v_i$ induces a connected subgraph of G. The words in $A^{\mathrm{VS}}_{G,r}$ model each way of traversing the graph in a manner similar to BFS or DFS. The basic idea is easily extended to directed graphs; here we require that each prefix $v_1 v_2 \ldots v_i$ induces a subgraph of G where every vertex is reachable from $r = v_1$. (We assume that each vertex is reachable from r in G.)

It is easy to see that $A^{\mathrm{VS}}_{G,r}$ is indeed an antimatroid; for example, as a precedence formula for each vertex $v \neq r$, take $F_v = u_1 \vee u_2 \vee \cdots \vee u_k$, where $u_1, u_2, \ldots, u_k$ are the neighbors (resp. in-neighbors) of v. However, not every antimatroid is a vertex search antimatroid.

Again, the restricted sorting problem for vertex search antimatroids is solved in optimal time with the precedence formula representation given above.

The vertex search antimatroid plays a role in the *distance ordering* problem, a variant of single-source shortest path (SSSP) problem. Given is an edge-weighted directed graph (G, w) with a designated root r, and the task is to order the vertices by distance from r. Recently, Haeupler et al. [12] showed that Dijkstra's classical SSSP is *universally optimal* when equipped with a certain working-set heap (which is more powerful than the one presented in Section 2).

Universally optimal means, informally, that for every fixed graph, the algorihtm is worst-case optimal w.r.t. the weight function. In fact, they essentially match the information-theoretic lower bound with a running time of $\mathcal{O}(\log|T| + |V(G)| + |E(G)|)$, where T is the set of possible distance orderings for the input graph G with root r, with any weight function.

We now demonstrate a strong connection between antimatroid sorting and the distance ordering problem.[4] First off, the set T of possible distance orderings is precisely the vertex search antimatroid (see the full version of the paper for a proof). Thus, we can run topological heapsort to find a distance ordering with optimal running time – *if* we are allowed to directly compare two vertices. That means comparing the distances of two vertices to r, which we cannot do in the distance ordering problem (without computing the distances first).

Still, it can be seen that the *transcript* (see the proof of Theorem 5) of a run of Dijsktra's algorithm on an input (G, w, r) is the same as running topological heapsort on the corresponding vertex search antimatroid. This gives an alternative proof of universal optimality; details are found in the full version of the paper. It should be noted that Haupler et al. also give an algorithm that is universally optimal in both time and number of comparisons, which is harder (c.f. Section 5).

4.3 Elimination orderings

An undirected graph is *chordal* if has no induced cycles of length four or more. It is well-known that chordal graphs can be characterized by the existence of *perfect elimination orderings (PEOs).* A PEO is obtained by successively removing *simplicial* vertices, that is, vertices whose neighborhood form a clique.

The set of elimination orderings form an antimatroid, also known as the *simplicial vertex pruning* [1] or *simplicial shelling* [27] antimatroid. Given a graph G, we can define the simplicial vertex pruning antimatroid A_G^{SVP} via the MPS $\{p_v\}_{v \in V(G)}$ with

$$p_v(U) = \begin{cases} 1, & \text{if } v \text{ is simplicial in } G - U; \\ 0, & \text{otherwise.} \end{cases}$$

A candidate data structure for A_G is essentially a data structure that maintains the set of simplicial vertices in G under simplicial vertex deletion.

There are known static and fully-dynamic algorithms to compute simplicial vertices in *arbitrary* graphs [22, 28]. Since we only care about chordal graphs and only need vertex deletion, we can use a relatively simple data structure based on *clique trees.* Details are found in the full version of the paper. The total running time of removing all vertices is $\mathcal{O}(m + n)$. Thus, we can solve the corresponding restricted sorting problem in optimal time.

5 Comparison optimality

In this section, we modify the topological heapsort algorithm to only use an optimal $\mathcal{O}(\log|\mathcal{P}(S)|)$ comparisons, instead of $\mathcal{O}(|\Sigma| + \log|\mathcal{P}(S)|)$, while keeping the optimal running time. The special case of partial orders was again solved by Haeupler et al. [14]. Our algorithm is similar, but some non-trivial adaptation is required.

First of all, note that the additional $\mathcal{O}(n)$ term is only relevant when $|\mathcal{P}(S)| \le 2^{o(n)}$. In the partial order/DAG-sorting case, Haeupler et al. [14] showed that this condition implies the existence of a long directed path in the DAG. Observe that such a path is pre-sorted. We prove a corresponding fact for antimatroids (Section 5.1).

[4] Such a connection is not surprising, since the technique used in the analysis of Dijkstra's algorithm [12] inspired the original analysis of topological heapsort [14] and large parts of this paper.

Haeupler et al. then proceed roughly as follows. They first compute a set of *marked* vertices on the pre-sorted path. Intuitively, the marked vertices represent the interaction with the remainder of the graph: The removal of unmarked vertices does not affect "availability" of vertices outside of the path. This means that often, a large set of unmarked vertices can be removed at once, without any additional comparisons.

This strategy cannot be adapted to general antimatroids in a straight-forward way, since the availability constraints are too complex to preprecompute a small set of marked vertices. We instead use the following three-step algorithm (given is an antimatroid A on Σ):

1. Compute a long pre-sorted sequence β (Section 5.1).
2. Sort the set $\Gamma = \Sigma \setminus \tilde{\beta}$, using topological heapsort on the sub-antimatroid of A *induced* by Γ (Section 5.2). This gives us a sorted sequence γ.
3. Merge the two sequences β and γ, utilizing the antimatroid constraints (Section 5.3).

All three steps can be implemented efficiently using a candidate data structure for A. We obtain:

▶ **Theorem 7.** *Given a set Σ, a candidate data structure C for an antimatroid A on σ, and an oracle for a total order $\pi \in \mathcal{P}(A)$, we can sort Σ w.r.t. $\prec_\pi$ in time $\mathcal{O}(t^{\text{total}}(C) + \log |\mathcal{P}(A)|)$ and with $\mathcal{O}(\log |\mathcal{P}(A)|)$ comparisons.*

This means that each efficient candidate data structure presented in Section 4 yields a comparison-optimal sorting algorithm.

5.1 Bottlenecks in antimatroids

Let $S = \{p_x\}_{x \in \Sigma}$ be a monotone precedence system on Σ. The *layer sequence* of S is a partition of Σ into nonempty *layers* $L_1, L_2, \ldots, L_k$ inductively defined as follows:

- $L_1 = \{x \in \Sigma \mid p_x(\emptyset) = 1\}$.
- $L_i = \{x \in \Sigma \setminus \bar{L}_{i-1} \mid p_x(\bar{L}_{i-1}) = 1\}$, where $\bar{L}_{i-1} = L_1 \cup L_2 \cup \cdots \cup L_{i-1}$, for each $2 \leq i \leq k$.

Observe that a layer sequence exists if and only if $\mathcal{P}(S) \neq \emptyset$, and that a layer sequence is always unique. If a candidate data structure C is available, we can compute the layer sequence in $\mathcal{O}(t^{\text{total}}(C))$ time as follows: Call C.`init`() and add the reported elements to L_1. Then, call C.`step`(x) for each $x \in L_1$, add the reported elements to L_2, and so on.

▶ **Lemma 8.** *Let S be a monotone precedence system with k layers. Then $|\mathcal{P}(S)| \geq 2^{n-k}$.*

Proof. We essentially follow Haeupler et al. [14]. Let $L_1, L_2, \ldots, L_k$ be the layer sequence for S. For each $i \in [k]$, let α_i be an arbitrary permutation of L_i. We claim that the word $\alpha = \alpha_1\alpha_2 \ldots \alpha_k$ is contained in $\mathcal{P}(S)$. Indeed, if $x \in L_i$, then all $y \in \bar{L}_{i-1}$ precede x in α. Since $p_x(\bar{L}_{i-1}) = 1$ by definition of L_i, we have $\alpha \in \mathcal{P}(S)$.

There are $\prod_{i=1}^{k} |L_i|!$ ways of constructing α. Writing $n_i = |L_i|$ and $n = |\Sigma|$, we have

$$|\mathcal{P}(S)| \geq \prod_{i=1}^{k} n_i! \geq \prod_{i=1}^{k} 2^{n_i - 1} = 2^{n-k}. \qquad ◀$$

We now proceed to show how to find a long sequence $x_1x_2 \ldots x_k \in \Sigma_s^*$ such that for all $i \in [k-1]$, we have $x_i \prec_\alpha x_{i+1}$ for all $\alpha \in \mathcal{P}(A)$; in other words, the sequence is pre-sorted. In the DAG-sorting case, we can simply take a longest path in the DAG. Here, we need a different approach, which is also related to the technique of Haeupler et al. [14].

▶ **Lemma 9.** *Let $L_1, L_2, \ldots, L_k$ be the layers of a monotone precedence system S, and let $\pi \in \mathcal{P}(S)$. Then, for all $1 \leq i < j \leq k$ and $y \in L_j$, there exists an $x \in L_i$ such that $x \prec_\pi y$.*

Proof. Suppose the lemma does not hold for some pair $i < j$, i.e., there exists some $y' \in L_j$ such that $y' \prec_\pi x$ for all $x \in L_i$. Let $y = \min_\pi(\Sigma \setminus \bar{L}_i) = \min_\pi(L_{i+1} \cup L_{i+2} \cup \cdots \cup L_k)$. Since $y \preceq_\pi y'$, we have $y \prec_\pi x$ for all $x \in L_i$, implying that actually $y = \min_\pi(\Sigma \setminus \bar{L}_{i-1})$. This means that $p_y(\bar{L}_{i-1}) = 1$, implying that $y \in L_i$, a contradiction. ◀

An element $x \in \Sigma$ is a called a *bottleneck* if it is the only element in its layer. Let t be the number of bottlenecks and let k be the number of layers. Since each non-bottleneck layer contains at least two vertices, we have $n - t \geq 2(k - t)$, which implies $n - k \geq \frac{n-t}{2}$. By Lemma 8, we have

▶ **Corollary 10.** *Let S be an MPS with t bottleneck elements. Then $|\mathcal{P}(S)| \geq 2^{(n-t)/2}$.*

The *bottleneck sequence* $\beta = b_1 b_2 \dots b_t$ of S contains all bottlenecks, ordered by the layer they appear in. Lemma 9 implies that, for every permutation $\pi \in \mathcal{P}(S)$, we have $b_1 \prec_\pi b_2 \prec_\pi \cdots \prec_\pi b_t$. Overall, we obtain

▶ **Lemma 11.** *Let S be an MPS on Σ, and write $n = |\Sigma|$. Then there exists a sequence $\beta = b_1 b_2 \dots b_t$ that appears as a subsequence in every $\pi \in \mathcal{P}(S)$, and $|\mathcal{P}(S)| \geq 2^{(n-t)/2}$. Moreover, given a candidate data structure C for S, we can compute β in time $\mathcal{O}(t^{\text{total}}(C))$.*

5.2 Sorting subsets

In this section, we show how to sort the non-bottleneck elements $\Sigma \setminus \tilde{\beta}$ of an antimatroid A. Recall that a candidate data structure C is given, and we want to sort in overall time $\mathcal{O}(t^{\text{total}}(C) + \log|\mathcal{P}(A)|)$, with $\mathcal{O}(\log|\mathcal{P}(A)|)$ comparisons (see Theorem 7). We show how to sort any subset $\Gamma \subseteq \Sigma$ within this budget.

We first need some definitions. Let Σ be an alphabet. Given a word α and a subset $\Gamma \subseteq \Sigma$, the *restriction* of α to Γ, written $\alpha|_\Gamma$, is obtained by removing all letters in $\Sigma \setminus \Gamma$ from α. For a simple language $L \subseteq \Sigma_s^*$, let $L|_\Gamma = \{\alpha|_\Gamma \mid \alpha \in L\}$. It is known that if $A \subseteq \Sigma_s^*$ is an antimatroid, then the restriction $A|_\Gamma$ is also an antimatroid (called the *trace* [27, 1]).

Let us now assume we have a candidate data structure C_Γ for $A|_\Gamma$. Then we can use topological heapsort to sort Γ. The running time is $t^{\text{total}}(C_\Gamma) + \mathcal{O}(\log|\mathcal{P}(A|_\Gamma)|)$, and we need $\mathcal{O}(|\Gamma| + \log|\mathcal{P}(A|_\Gamma)|)$ comparisons. First, observe that $|\mathcal{P}(A|_\Gamma)| \leq |\mathcal{P}(A)|$, since the restriction operation surjectively maps $\mathcal{P}(A|_\Gamma)$ to $\mathcal{P}(A)$. Second, recall that $|\Gamma| \leq 2\log|\mathcal{P}(A)|$ if Γ is obtained by removing all bottleneck elements (Corollary 10). If we further assume that $t^{\text{total}}(C_\Gamma) \leq \mathcal{O}(t^{\text{total}}(C))$, then we are within the budget of Theorem 7.

It remains to build a candidate data structure C_Γ for $A|_\Gamma$ with $t^{\text{total}}(C_\Gamma) \leq \mathcal{O}(t^{\text{total}}(C))$. Suppose C is given. Then $C_\Gamma.\texttt{init}()$ is implemented as follows. First, call $C.\texttt{init}()$, and store the result in a set Y. Then, we repeat the following as long as there is an element $y \in Y \setminus \Gamma$: Discard y, call $C.\texttt{step}(y)$, and add all reported elements to Y. At the end, report the final set Y.

To implement $C_\Gamma.\texttt{step}(y)$, we similarly first call $C.\texttt{step}(y)$, store the result in a set Y, and perform the repeated replacement as above.

The (rather technical) correctness proof for C_Γ is found in the full version of the paper. The running time is clearly $t^{\text{total}}(C) + \mathcal{O}(|\Sigma|) = \mathcal{O}(t^{\text{total}}(C))$. Thus, we have:

▶ **Lemma 12.** *Given a set Σ, a subset $\Gamma \subseteq \Sigma$, a candidate data structure C for an antimatroid A on Σ, and an oracle for a total order $\pi \in \mathcal{P}(A)$, we can sort Γ w.r.t. $\prec_\pi$ in time $\mathcal{O}(t^{\text{total}}(C) + \log|\mathcal{P}(A)|)$ and with $\mathcal{O}(|\Gamma| + \log|\mathcal{P}(A)|)$ comparisons.*

5.3 Merging

We now want to optimally merge under antimatroid constraints:

▶ **Theorem 13.** *Let A be an antimatroid on Σ, let $\pi \in \mathcal{P}(A)$, and let Γ, Δ be a partition of Σ. Let $\gamma = \pi|_\Gamma$ and $\delta = \pi|_\Delta$. Let $R \subseteq \mathcal{P}(A)$ contain each permutation $\rho \in \mathcal{P}(A)$ such that $\rho|_\Gamma = \gamma$ and $\rho|_\Delta = \delta$.*

Given Σ, γ, δ, an oracle for π, and a candidate data structure C for A, we can compute π in time $\mathcal{O}(t^{\text{total}}(C) + \log|R|)$ and with $\mathcal{O}(|\Delta| + \log|R|)$ comparisons.

A proof of Theorem 13 is given in the full version of the paper. The algorithm used can be seen as a simplified version of Haeupler et al.'s *heapsort with lookahead* algorithm [14]; in fact, if the antimatroid A is given as a collection of precedence formulas, it simplifies to a partial order under the assumption that γ and δ are sorted. (However, our proof works with general candidate data structures.)

5.4 Putting things together

We now prove Theorem 7. First, we compute the bottleneck sequence β with Lemma 11. Then, we use Lemma 12 to sort the subset $\Gamma = \Sigma \setminus \tilde{\beta}$. In other words, if π is the underlying total order, we compute $\gamma = \pi|_\Gamma$. Finally, we merge β and γ using Theorem 13. The overall running time is $\mathcal{O}(t^{\text{total}}(C) + \log|\mathcal{P}(A)| + \log|R|)$, and we use $\mathcal{O}(\log|\mathcal{P}(A)| + |\Gamma| + \log|R|)$ comparisons, where R is as defined in Theorem 13. Note that $R \subseteq \mathcal{P}(A)$. Moreover, we have $|\Gamma| = n - |\beta| \leq 2\log|\mathcal{P}(A)|$ by Lemma 11. Thus, we need $\mathcal{O}(t^{\text{total}}(C) + \log|\mathcal{P}(A)|)$ time and $\mathcal{O}(\log|\mathcal{P}(A)|)$ comparisons, as desired.

6 Conclusion

In this paper, we have shown that the basic variant of *topological heapsort* nicely generalizes to the problem of sorting antimatroids, and its running time stays optimal. We also showed how the algorithm can be modified to be comparison-optimal in the antimatroid case, which in particular implies that the information-theoretic bound is tight for antimatroids. Since topological heapsort is not optimal for further generalizations like *greedoids* (see the full version of the paper), perhaps antimatroids are the most general structure that can be sorted "greedily". The question whether the information-theoretic bound for greedoids holds, and whether greedoids can be sorted efficiently in some other way, is left open.

For another interesting open question, let us consider an important class of antimatroids that was omitted from Section 4. *Convex shelling antimatroids* [6, 27, 1] are obtained by progressively removing points from the convex hull of a given point set. A candidate data structure for such an antimatroid would be a certain kind of *decremental convex hull* data structure. The total running time of this data structure cannot be linear as in the other examples, since computing the convex hull can take superlinear time [20]. Is there a candidate data structure for the convex shelling antimatroid that is fast enough to sort optimally?

References

1 Anders Björner and Günter M. Ziegler. Introduction to Greedoids. In Neil White, editor, *Matroid Applications*, volume 40 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 1992. `doi:10.1017/CBO9780511662041.009`.

2 G. R. Brightwell, S. Felsner, and W. T. Trotter. Balancing pairs and the cross product conjecture. *Order*, 12(4):327–349, 1995. `doi:10.1007/bf01110378`.

3 Graham R. Brightwell. Balanced pairs in partial orders. *Discret. Math.*, 201(1-3):25–52, 1999. doi:10.1016/S0012-365X(98)00311-2.

4 Mark R. Brown and Robert Endre Tarjan. A fast merging algorithm. *J. ACM*, 26(2):211–226, 1979. doi:10.1145/322123.322127.

5 Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M. Jungers, and J. Ian Munro. Sorting under partial information (without the ellipsoid algorithm). In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 359–368. ACM, 2010. doi:10.1145/1806689.1806740.

6 Brenda L. Dietrich. A circuit set characterization of antimatroids. *Journal of Combinatorial Theory, Series B*, 43(3):314–321, 1987. doi:10.1016/0095-8956(87)90007-4.

7 R. P. Dilworth. Lattices with unique irreducible decompositions. *Annals of Mathematics*, 41(4):771–777, 1940. doi:10.1007/978-1-4899-3558-8_10.

8 David Eppstein. Antimatroids and balanced pairs. *Order*, 31(1):81–99, 2014. doi:10.1007/S11083-013-9289-1.

9 Michael L. Fredman. How good is the information theory bound in sorting? *Theoretical Computer Science*, 1(4):355–361, 1976. doi:10.1016/0304-3975(76)90078-5.

10 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, December 1993. doi:10.1016/0022-0000(93)90040-4.

11 William Cawley Gelling, Markus E. Nebel, Benjamin Smith, and Sebastian Wild. Multiway powersort. In Gonzalo Navarro and Julian Shun, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2023, Florence, Italy, January 22-23, 2023*, pages 190–200. SIAM, 2023. doi:10.1137/1.9781611977561.CH16.

12 Bernhard Haeupler, Richard Hladík, Václav Rozhon, Robert E. Tarjan, and Jakub Tetek. Universal optimality of dijkstra via beyond-worst-case heaps. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2099–2130. IEEE, 2024. doi:10.1109/FOCS61266.2024.00125.

13 Bernhard Haeupler, Richard Hladík, John Iacono, Vaclav Rozhon, Robert Tarjan, and Jakub Tětek. Fast and simple sorting using partial information, 2024. arXiv:2404.04552v1.

14 Bernhard Haeupler, Richard Hladík, John Iacono, Václav Rozhoň, Robert E. Tarjan, and Jakub Tětek. Fast and simple sorting using partial information. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3953–3973. SIAM, 2025. doi:10.1137/1.9781611978322.134.

15 Yijie Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *J. Algorithms*, 50(1):96–105, January 2004. doi:10.1016/j.jalgor.2003.09.001.

16 Yijie Han and Mikkel Thorup. Integer sorting in $O(n\sqrt{\log \log n})$ expected time and linear space. In *Proceedings of the 43 rd Annual IEEE Symposium on Foundations of Computer Science (FOCS'02)*, pages 135–144. IEEE Comput. Soc, 2002. doi:10.1109/SFCS.2002.1181890.

17 Frank K. Hwang and Shen Lin. A simple algorithm for merging two disjoint linearly-ordered sets. *SIAM J. Comput.*, 1(1):31–39, 1972. doi:10.1137/0201004.

18 Arthur B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962. doi:10.1145/368996.369025.

19 Jeff Kahn and Jeong Han Kim. Entropy and sorting. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing - STOC '92*, STOC '92, pages 178–187. ACM, 1992. doi:10.1145/129712.129731.

20 David G. Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 15(1):287–299, 1986. doi:10.1137/0215021.

21 S. S. Kislitsyn. A finite partially ordered set and its corresponding set of permutations. *Mathematical Notes of the Academy of Sciences of the USSR*, 4(5):798â€“801, November 1968. doi:10.1007/bf01111312.

22 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000. doi:10.1016/S0020-0190(00)00047-8.

23 Donald E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, 1968.

24 Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1973.

25 B. Korte and L. Lovász. Mathematical structures underlying greedy algorithms. In Ferenc Gécseg, editor, *Fundamentals of Computation Theory*, volume 117 of *Lecture Notes in Computer Science*, pages 205–209, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg. doi:10.1007/3-540-10854-8_22.

26 Bernhard Korte and László Lovász. Greedoids - a structural framework for the greedy algorithm. In William R. Pulleyblank, editor, *Progress in Combinatorial Optimization*, pages 221–243. Elsevier, 1984. doi:10.1016/b978-0-12-566780-7.50019-2.

27 Bernhard Korte, Rainer Schrader, and László Lovász. *Greedoids*. Springer Berlin Heidelberg, 1991. doi:10.1007/978-3-642-58191-5.

28 Min Chih Lin, Francisco J. Soulignac, and Jayme Luiz Szwarcfiter. Arboricity, h-index, and dynamic algorithms. *Theor. Comput. Sci.*, 426:75–90, 2012. doi:10.1016/J.TCS.2011.12.006.

29 Nathan Linial. The information-theoretic bound is good for merging. *SIAM J. Comput.*, 13(4):795–801, 1984. doi:10.1137/0213049.

30 Nazarré Merchant and Jason Riggle. OT grammars, beyond partial orders: ERC sets and antimatroids. *Natural Language &amp; Linguistic Theory*, 34(1):241–269, August 2015. doi:10.1007/s11049-015-9297-5.

31 Michal Opler. An optimal algorithm for sorting pattern-avoiding sequences. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 689–699. IEEE, 2024. doi:10.1109/FOCS61266.2024.00049.

32 Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. Simpler optimal sorting from a directed acyclic graph. In Ioana Oriana Bercea and Rasmus Pagh, editors, *2025 Symposium on Simplicity in Algorithms (SOSA)*, pages 350–355. SIAM, January 2025. doi:10.1137/1.9781611978315.26.

33 Ivor van der Hoog and Daniel Rutschmann. Tight bounds for sorting under partial information. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*, pages 2243–2252. IEEE, 2024. doi:10.1109/FOCS61266.2024.00131.

34 J. W. J. Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, 1964.