hexagon:2611.00196v2

An Optimal Deterministic Adversary for Sorting

Nicholas Nash
ORCID: 0009-0003-0475-7852

8 November 2026

Abstract

This paper describes an optimal *deterministic*, polynomial-time adversary for ordinary sorting. For any correct sorting algorithm, the adversary forces that algorithm to make $(1 - o(1))n \log_2 n$ comparisons. The best previously known deterministic polynomial-time adversary forced $\frac{37}{64} n \log_2 n$ comparisons. The adversary maintains a family of orders consistent with the answers so far, using a hierarchy of directed acyclic graphs. Bounds on vertex degrees and directed path lengths allow comparison answers to be chosen through deterministic calculations on small graph neighborhoods.

Methodological disclosure

The adversary and its analysis are due to AI (GPT 6 Astra). The human author shaped the exposition and abstractions, and reviewed the paper *except for the appendices*. A Lean formalization is available here.

1 Introduction

We consider comparison sorting of n distinct labelled keys.[1] A sorting algorithm (or *sorter*, for short) chooses each pair using the answers to its earlier comparisons and must be correct for every strict total order on the labels. A sorting adversary supplies those answers as the comparisons arrive. The sorting adversary is *legal* if every finite sequence of its answers is consistent with at least one total order. The adversary knows n and sees the comparisons, but it need not inspect the sorter that generates them.

The classical *information-theoretic lower bound* (ITLB) shows that any sorter must perform $\lceil \log(n!) \rceil = n \log n - O(n)$ comparisons in the worst case [1, Section 5.3.1]. This bound guarantees the existence of a hard input for every correct sorter. Here the problem is to produce hard answers by one *deterministic, polynomial-time* adversary algorithm.

Kaplan, Zamir, and Zwick [7, Section 7] ask whether an efficient deterministic adversary can force $(1-\varepsilon)n \log n$ comparisons for every $\varepsilon > 0$ and all sufficiently large n. The following theorem answers their question.

Theorem 1.1. *There is a deterministic legal adversary, initialized using only n, that uses polynomially many bit operations per query and forces every correct sorter to make at least $(1 - o(1))n \log n$ comparisons.*

Proof. See Section 5. □

[1] Throughout, log denotes $\log_2$, and ln the natural logarithm. We assume $n \geq 2$.

We refer to an adversary satisfying Theorem 1.1 as an *optimal* adversary. Without the restriction to polynomial-time, a deterministic optimal adversary is immediate. With randomness, simple, polynomial-time adversaries achieve optimality in both expectation and with high probability, just by generating random permutations.

Deterministic adversaries without the polynomial-time restriction If an adversary need not operate in polynomial-time, an optimal deterministic adversary is a simple book-keeping exercise. The adversary must simply keep all total orders consistent with the answers so far. Each comparison partitions them into two classes. The adversary answers so as to retain a larger class. After q comparisons, at least $n!/2^q$ orders remain, forcing $q \geq \log(n!)$. This rule is deterministic, but enumerating its classes may require factorial work. The total orders consistent with the answers are the *linear extensions* of the resulting partial order. Counting these extensions exactly is #P-complete [10].

Deterministic adversaries in polynomial-time Kahn and Kim [12] already give a deterministic polynomial-time adversary for sorting with partial information. Starting from any given partial order P, it answers comparisons online and forces $\frac{1}{2}\log e(P)$ comparisons, where $e(P)$ is the number of linear extensions of P, and thus $\frac{1}{2}n\log n - O(n)$ for ordinary sorting. We note that, for ordinary sorting, the much simpler adversary of Brodal et al [6] achieves the same guarantee.

With randomness If randomness is allowed, an optimal, polynomial-time adversary is a simple matter of generating a single uniformly random permutation. After choosing such a permutation the adversary simply answers from the ranks in that permutation. For every fixed correct sorter, the usual comparison-tree argument gives at least $\log(n!)$ comparisons in expectation and $\log(n!) - O(\log n)$ comparisons with probability $1 - o(1)$.

Randomized approximation of the number of linear extensions. Randomization also makes it possible to estimate how many linear extensions remain after each possible answer. Given the current partial order P, assign each label a real coordinate in $(0, 1)$ and form the region

$$K(P) = \{X \in (0,1)^n : X_i < X_j \text{ whenever } i <_P j\}.$$

Each allowed total order occupies volume $1/n!$ in the unit cube. Hence $\mathrm{vol}(K(P)) = e(P)/n!$ [9]. The two possible answers to a comparison cut this region into two parts, and their volumes are proportional to their counts of allowed orders. Dyer, Frieze, and Kannan [11] approximate these volumes in randomized polynomial time, so randomness also lets the adversary estimate which answer retains more orders.

Organization of this paper. To motivate our adversary, we describe the earlier adversaries of Brodal et al. [6], and Kaplan et al [8] in Section 2. Section 3 describes how the adversary records comparisons in directed acyclic graphs and uses calculations on small graph neighborhoods to choose its answers. The key guarantees are stated there, with proofs deferred. Section 4 proves the comparison lower bound using the local guarantees stated in Section 3. Section 5 then spells out how to choose graph bounds as a function of n to obtain polynomial running time. Section 6 states some immediate corollaries for sorting under partial information, as well as for efficient partial order production.

Lastly, the appendices prove the properties of *order-restriction DAGs* (defined in Section 3.1) and the consistency of the tree updates, and give the deterministic local calculations. Note that the paper, *except for the appendices*, has been reviewed by the author. The proofs in the appendices remain raw AI-generated work.

2 Related Work

2.1 Sorting with partial information

Fredman [2] studied how closely sorting can approach the information-theoretic lower bound when some orders have already been excluded. For any nonempty family Γ of permutations of n items, he showed that the unknown order can be identified using at most $\log|\Gamma| + 2n$ comparisons. In particular, if the available information is a partial order P, this gives $\log e(P) + 2n$ comparisons, where $e(P)$ counts its linear extensions. The comparison bound is within an additive $2n$ of the ITLB, but constructing the strategy requires enumerative information about the possible orders and does not give a polynomial-time algorithm for general partial orders [12, p. 390].

Kahn and Kim [12] later gave a deterministic polynomial-time sorting algorithm using $O(\log e(P))$ comparisons. They also constructed a deterministic polynomial-time adversary forcing $\Omega(\log e(P))$ comparisons. More recent work gives much faster algorithms for sorting from supplied comparisons [21, 22]. Our adversary faces the corresponding task of choosing answers efficiently while forcing asymptotically the full ITLB for ordinary sorting.

2.2 Binary-tree adversaries

Atallah and Kosaraju [3] gave an efficient sorting adversary with a lower bound of $\frac{1}{2}n\log n$ comparisons. Related adversaries arose in slightly different problems studied by Borodin, Guibas, Lynch, and Yao [4], Richards and Vaidya [5], and Brodal, Chaudhuri, and Radhakrishnan [6]. The binary-tree rule of Borodin et al. and Brodal et al. provides the starting point for our construction.

Place every item at the root of an ordered binary tree. Items in disjoint subtrees have the left-to-right order of those subtrees. A comparison changes the placement as follows.

- If both items occupy the same node, move one to its left child and the other to its right child, and answer accordingly.
- If one item is at an ancestor of the other, move the higher item to the child whose subtree does not contain the lower item.
- If their nodes are unrelated by ancestry, answer from their left-to-right order, without moving either item.

For a concrete example, start with A, B, C, D at the root. Answering $A < B$ moves A to the left child and B to the right, while C, D remain at the root. If the sorter next compares A and C, the ancestor rule moves C to the right child and gives $A < C$. Comparing D and B then moves D to the left child and gives $D < B$. The two children now contain $\{A, D\}$ and $\{B, C\}$. A subsequent comparison of D with C is answered $D < C$ without moving either item. The orders within $\{A, D\}$ and within $\{B, C\}$ remain unresolved. Figure 1 shows the placements.

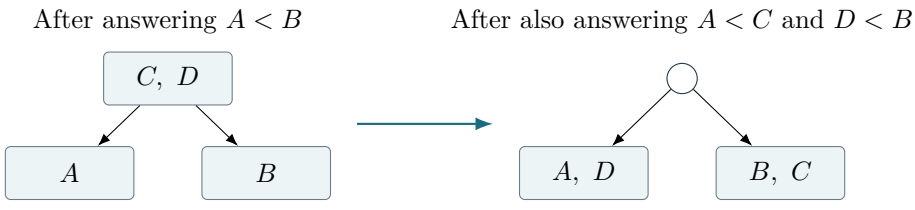


Figure 1: The binary-tree adversary. The first comparison separates two items at the same node. The next two comparisons each move an item from an ancestor into the opposite child. In the final state, comparing D with C requires no movement.

Each comparison increases the sum of item depths by at most two. At completion, occupied nodes must be distinct and unrelated by ancestry: otherwise two represented items can occur

in either relative order, leaving an unresolved comparison. Associate the root with $(0, 1)$, and associate each child with the corresponding half of its parent's interval. A node of depth d_i therefore represents an interval of length 2^{-d_i}. At completion these intervals are disjoint, giving

$$\sum_i 2^{-d_i} \le 1, \qquad 2^{-\frac{1}{n}\sum_i d_i} \le \frac{1}{n}\sum_i 2^{-d_i} \le \frac{1}{n}.$$

Convexity gives the bound on $2^{-\frac{1}{n}\sum_i d_i}$. Hence $\sum_i d_i \ge n \log n$, and at least $\frac{1}{2} n \log n$ comparisons are necessary [6, Section 4.1].

The tree representation can settle relations that have never been queried. For example, start again with four items at the root and answer $A < B$ and $C < D$. Placing A, C in the left child and B, D in the right child also commits the adversary to $A < D$ and $C < B$. Neither relation follows from those two answers alone.

2.3 The adversary of Kaplan, Zamir, and Zwick

Kaplan, Zamir, and Zwick [8] use a four-child tree. Intermediate states correspond to consecutive ranges of children, so an item can narrow its range before entering a particular child. See Figure 2.

A key idea of this adversary is that a comparison between two items need not immediately cause them to descend in the tree structure. Instead, pairs of items whose relative order is fixed can remain in the same node.

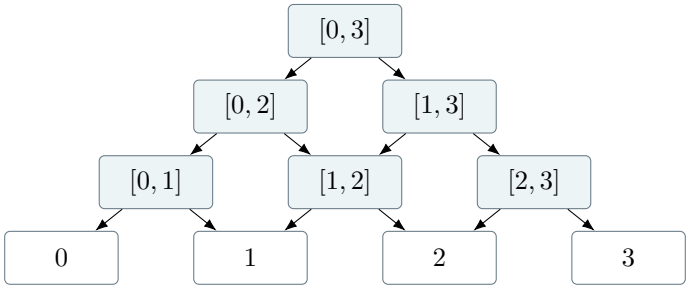


Figure 2: One step of the four-child tree in the adversary of Kaplan, Zamir, and Zwick [8, Section 5]. An intermediate node records a range of possible children. An item at $[1, 3]$ can still enter any of children 1, 2, or 3.

Their analysis measures the sorters progress in terms of a suitably defined potential, our analysis is similar, although more involved. The resulting lower bound of this adversary is $\frac{37}{64} n \log n$ comparisons, improving the earlier $\frac{1}{2} n \log n$ bound.

3 The new optimal adversary

Kaplan, Zamir, and Zwick [8, Section 6] suggest using continuous intervals and retaining more complex dependencies between items. We follow this direction by allowing DAGs of items at each tree node. The main tasks are to choose answers using bounded neighborhoods and to show that restoring the degree and path bounds requires few moves overall.

3.1 Order-restriction DAGs

An *order-restriction DAG* D consists of a directed acyclic graph on a finite set V, together with an open interval $(\ell_x, r_x) \subseteq (0, 1)$ for each vertex x. Its *feasible assignments* are the points in

$$K(D) = \{X \in \mathbb{R}^V : \ell_x < X_x < r_x \text{ for every } x, \quad X_x < X_y \text{ for every arc } x \to y\}. \tag{1}$$

Assignments with distinct coordinates determine relative orders of the vertices. For example, the arcs $a \to b$ and $a \to c$, with all three intervals equal to $(0, 1)$, allow both $a < b < c$ and $a < c < b$. Write $V(D) = \operatorname{vol} K(D)$, taking $V(D) = 1$ when V is empty.

We call the intervals *canonical* when each is the full range of its coordinate in feasible assignments.

3.2 A tree of order-restriction DAGs

The core data structure of our adversary is a rooted tree. Each node v stores an order-restriction DAG D_v, whose vertices are the items recorded there. Figure 3 shows an example.

Each item has a record (x, ℓ_x, r_x) stored at exactly one node. The label x identifies the key. The variable X_x describes its position among the items and child anchors at that node. We call the item *free* there while this position remains variable. The interval endpoints are *dyadic rationals*, meaning that their denominators are powers of two. The stored intervals are canonical.

An arc $x \to y$ in D_v records a comparison answered $x < y$ between items stored at v. A comparison between two free items contributes at most one arc.

Initially the tree consists of just the root, containing all records with bounds $\ell_x = 0$ and $r_x = 1$. The position variables X_x are unassigned, and the root's graph has no arcs. Thus every assignment in $(0, 1)^n$ is feasible, and every total order can be obtained by assigning distinct positions.

Each child is attached to a fixed position $c \in (0, 1)$ in its parent, called its *anchor*. A node has at most one child at each anchor. For any feasible assignment at the parent, an item whose position is below c precedes the entire child subtree, and an item whose position is above c follows it. Subtrees of different children are ordered by their anchors. All items of each child subtree therefore occur consecutively in the resulting total order. The child's order-restriction DAG describes the possible orders within this block, using position variables in $(0, 1)$.

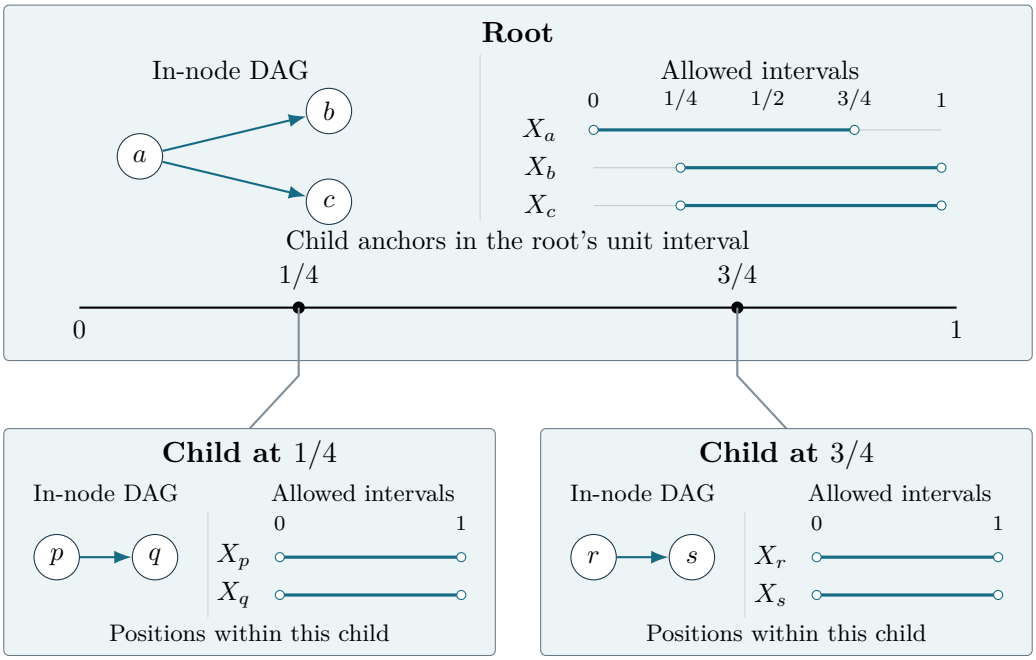


Figure 3: A state of the adversary. Each large box is a tree node, containing an order-restriction DAG, drawn as its arcs and allowed intervals.

In the root of Figure 3, the in-node DAG records $a < b$ and $a < c$. To read an order from this state, choose feasible positions at each node, avoiding ties with other items or child anchors. List

the items and anchors from left to right, and recursively replace each anchor by its child's order. For example, root positions $(X_a, X_b, X_c) = (1/3, 1/2, 7/8)$ give

$$p < q < a < b < r < s < c.$$

Other feasible choices give other orders. The *transcript* $\mathcal{T}$ is the sequence of answers so far. Our invariant is that every represented order satisfies the transcript. The tree need not represent every order that satisfies it.

3.3 Answering a comparison

Algorithm 1 gives the comparison procedure. Its main choice is delegated to `Select`, which encapsulates the key algorithmic result of this paper.

Algorithm 1: `Compare`(A, B): answer a comparison query.

Input: Distinct item labels A, B and the current adversary state.
Output: An answer $A < B$ or $B < A$, with the state updated.
if *fixed ranks have been chosen* **then**
  return their order of A, B
if $|\mathcal{T}| = n \lceil \log n \rceil$ **then**
  Form the DAG of answers in $\mathcal{T}$ and choose a topological order, breaking ties by label
  Store its ranks and **return** their order of A, B
while *true* **do**
  $v \leftarrow$ the root
  while *A and B lie in the same child subtree of v* **do**
    $v \leftarrow$ that child
  if *A and B lie in different child subtrees of v* **then**
    answer $\leftarrow$ the left-to-right order of their child anchors
    break
  $\alpha, \beta \leftarrow$ the coordinates or child anchors representing A, B at v
  $r \leftarrow$ `Select`(D_v, α, β)
  if *r is an answer* **then**
    answer $\leftarrow$ the corresponding order of A, B
    break
  $(x, c) \leftarrow r$
  `RefineAt`(x, c)
if *at least one queried item is stored at v* **then**
  Store *answer* as an arc in the graph at v or as an interval bound
  `Normalize`(D_v)
Append *answer* to $\mathcal{T}$
`Repair`(v)
return *answer*

3.3.1 Choosing an answer

`Select` uses deterministic estimates of the fractions of $K(D)$ retained by the two answers. It obtains these estimates by examining bounded neighborhoods of the queried vertices in the DAG. The size of each neighborhood depends on vertex degree (the sum of its in-degree and out-degree) and the length of the longest directed path in the DAG. We use one integer k to specify these bounds in Lemma 3.1. Larger k allows higher degrees and longer paths, while the guaranteed retained fraction approaches one half.

Algorithm 2 gives the `Select` algorithm. It calls `Enclose`, given in Algorithm 3. `Enclose` returns a list whose entries give one interval I_x for each queried coordinate x.

Algorithm 2: `Select`(D, α, β): choose an answer or a position for refinement.

Input: A canonical order-restriction DAG D and a query between two distinct coordinates or between a coordinate and a dyadic constant.
Output: An answer $\alpha < \beta$ or $\beta < \alpha$, or a pair (x, c).
$T \leftarrow$ the queried coordinates
$\mathcal{L} \leftarrow$ `Enclose`(D, T)
In every entry, use $[t, t]$ for a queried constant t
$p \leftarrow$ the fraction of entries with $\max I_\alpha < \min I_\beta$
$q \leftarrow$ the fraction of entries with $\max I_\beta < \min I_\alpha$
if $\max(p, q) \geq 1/2 - 1/(100k)$ **then**
  if $p \geq q$ **then return** $\alpha < \beta$
  return $\beta < \alpha$
$x \leftarrow$ the smallest label in T
$W_x \leftarrow r_x - \ell_x$
$h \leftarrow$ the largest power of two at most $W_x/2$
$c \leftarrow h\lceil(\ell_x + W_x/4)/h\rceil$
return (x, c)

To state the guarantee provided by `Select`, let f_x be the density of X_x under a uniform feasible assignment in $K(D)$. Explicitly,

$$f_x(c) = \frac{\mathrm{vol}_{|V|-1}(K(D) \cap \{X_x = c\})}{V(D)}. \tag{2}$$

The numerator is the volume of the section $X_x = c$, measured in the remaining $|V|-1$ coordinates.

The following lemma expresses the key idea: either we answer while retaining nearly half the feasible volume, mirroring the ideal adversary, or we identify a refinement that greatly increases the product of the node volumes (Lemma 3.3).

Lemma 3.1 (Local selection)**.** *Let $k \geq 64$ be an integer and D an order-restriction DAG with canonical dyadic intervals, $V(D) > 0$, degree at most k, and directed paths of at most 2^{k+1} vertices. Algorithm 2 returns one of the following:*

1. *An answer retaining at least $1/2 - 1/(100k)$ of $V(D)$.*
2. *A queried coordinate x and a dyadic $c \in (\ell_x, r_x)$ with $f_x(c) > 2^k$.*

The calculation is deterministic and uses only the graph and interval bounds within a distance of the queried coordinates depending only on k.

Proof. See Appendices B and E. □

Proof idea for Lemma 3.1. We sketch the enclosing-interval construction behind `Select`'s probability estimates. Pass through the vertices in topological order, changing one position at a time. Its permitted range is determined by the stored interval and the neighbors' positions. Split this range into S equal pieces, choose one, and move the position to the same relative point in that piece. For example, choosing the second quarter of $(0, 1)$ sends a position u to $(1 + u)/4$.

Run this procedure on two copies of the position vector, starting one at all lower interval endpoints and the other at all upper endpoints. At each step, use the same piece number in both copies, each within its own permitted range.

For the proof, also consider a position vector X, initially uniform in $K(D)$. At each update, choose a piece number independently and uniformly, and use it in all three vectors. The update

preserves coordinatewise order, so X stays between the two endpoint copies (Figure 4). It also preserves the uniform distribution on $K(D)$: conditional on the other positions, it sends a uniform position into a uniformly chosen equal piece. This use of enclosing copies is related to sampling algorithms for linear extensions [20, Section 4.2].

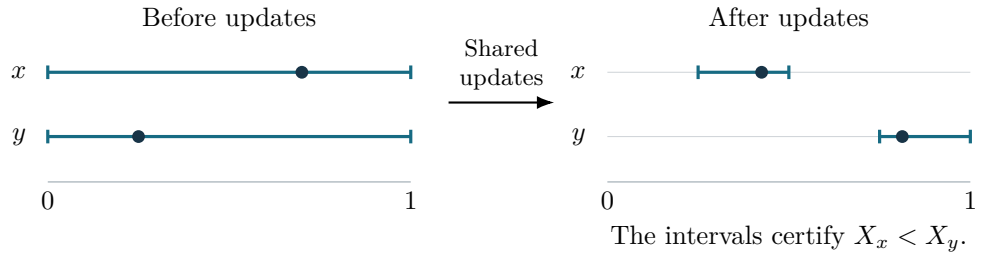


Figure 4: Updating two unconstrained coordinates using four pieces. All three copies choose the second piece for x and the fourth for y. The blue interval endpoints show the two enclosing positions for each coordinate. The dots show one feasible assignment between them. The resulting disjoint intervals certify $X_x < X_y$.

Disjoint enclosing intervals determine the queried order, so narrow intervals make the probability estimates useful. At a vertex with no incoming arcs, choosing the first piece moves both endpoint copies towards their common lower bound and reduces their separation. The proof tracks how this shrinking passes through neighboring vertices. The degree and path bounds control how many passes are needed to make the queried intervals narrow for most choice sequences.

Within one pass, dependence on newly updated positions follows directed paths. After a bounded number of passes, only a bounded neighborhood can affect a queried interval. `Enclose` enumerates the update choices there, and `Select` counts the entries that certify each comparison answer. The density argument for the refinement alternative is left to the full proof. □

Keeping nearly half the volume at each answer is the geometric counterpart of keeping at least half the possible total orders. Section 4 relates the node volumes to the number of orders still possible. We use this count to prove that every correct sorter must make at least $(1 - o(1))n \log n$ comparisons.

Algorithm 3: `Enclose`(D, T): compute enclosing intervals for the queried coordinates.

Input: An order-restriction DAG D and a set T of one or two queried coordinates.
Data: Fixed neighborhood radius ρ, number of repetitions t, and power of two S giving the number of update choices, chosen in Appendix D.
Output: A list of interval families $(I_x)_{x \in T}$.

$R \leftarrow$ vertices within distance ρ of T, ignoring arc directions
$F \leftarrow$ vertices at distance strictly less than ρ from T
$\pi \leftarrow$ a topological order of the DAG induced by R, breaking ties by label
$\mathcal{L} \leftarrow$ the empty list
foreach *assignment* $j_{z,s} \in \{0, \dots, S-1\}$ *for* $z \in F$ *and* $1 \le s \le t$ **do**
  $A_z \leftarrow \ell_z$, $B_z \leftarrow r_z$ for every $z \in R$
  for $s = 1, \dots, t$ **do**
    foreach $z \in F$ *in the order* π **do**
      foreach $Y \in \{A, B\}$ **do**
        $L \leftarrow \max(\{\ell_z\} \cup \{Y_u : u \to z\})$
        $U \leftarrow \min(\{r_z\} \cup \{Y_u : z \to u\})$
        $Y_z \leftarrow \big(Y_z + (S - 1 - j_{z,s})L + j_{z,s}U\big)/S$
  Append $([A_x, B_x])_{x \in T}$ to $\mathcal{L}$
return $\mathcal{L}$

3.3.2 Moving an item into a child

Instead of answering the query, `Select` may return a pair (x, c) directing `Compare` to move item x into the child anchored at c, creating that child if necessary. We now describe this move and its effect on feasible volume.

Algorithm 4 fixes $X_x = c$ in the parent's order-restriction DAG. Its call to `Normalize` (Algorithm 5) makes the remaining intervals canonical without changing their feasible region (Lemma A.1).

Algorithm 4: `Restrict`(D, x, c): fix and remove a coordinate.

Input: A canonical order-restriction DAG D with $V(D) > 0$, a vertex x, and $c \in (\ell_x, r_x)$.
foreach *arc* $y \to x$ **do**
  $r_y \leftarrow \min(r_y, c)$
foreach *arc* $x \to z$ **do**
  $\ell_z \leftarrow \max(\ell_z, c)$
Delete x and its incident arcs
`Normalize`(D)
return D

Algorithm 5: `Normalize`(D): propagate the interval bounds.

Input: An order-restriction DAG D.
foreach *vertex* x *in topological order* **do**
  $\ell_x \leftarrow \max(\{\ell_x\} \cup \{\ell_y : y \to x\})$
foreach *vertex* x *in reverse topological order* **do**
  $r_x \leftarrow \min(\{r_x\} \cup \{r_y : x \to y\})$

Lemma 3.2 (Coordinate restriction)**.** *Let D be canonical with $V(D) > 0$ and $c \in (\ell_x, r_x)$. Then* `Restrict`(D, x, c) *represents the section $X_x = c$ of $K(D)$, and*

$$V(\mathrm{Restrict}(D, x, c)) = V(D) f_x(c) > 0. \tag{3}$$

Proof. See Appendix A. □

Algorithm 6 uses this operation in the tree. Figure 5 illustrates a move into an existing child.

Algorithm 6: `RefineAt`(x, c): move an item into a child.

Input: A free item x, an interior anchor c, and the current adversary state.
Output: The updated state, with x stored in a child of its former node.
$v \leftarrow$ the node storing x
$D_v \leftarrow$ `Restrict`(D_v, x, c)
if *v has no child anchored at c* **then**
  Create a child of v at c, with an empty order-restriction DAG
$u \leftarrow$ the child of v anchored at c
Insert x into D_u as an isolated vertex with record $(x, 0, 1)$

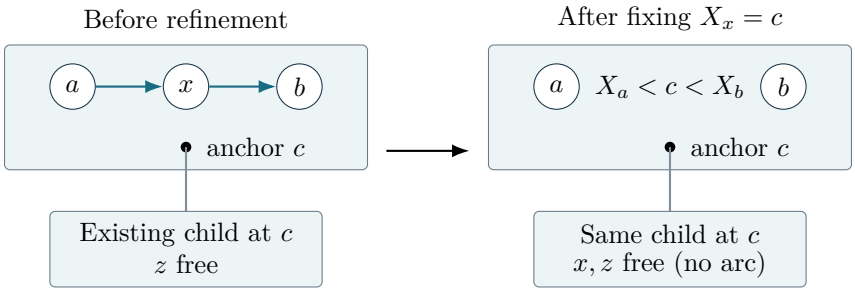


Figure 5: The two arc inequalities become bounds on the remaining parent coordinates. The new coordinate of x orders it within the child, where both $x < z$ and $z < x$ remain possible. The anchor c determines where the child block lies among the parent items. The child's constraints determine the possible orders within that block.

At an interior anchor, the old constraints allow x immediately before or after the existing child block. Both placements satisfy the transcript, so no earlier answer compares x with an item in that block. Thus x can enter the child without contradicting any earlier answer (Appendix E). This move can allow orders previously excluded by the tree, while preserving every answer in the transcript.

When `Select` returns (x, c), `RefineAt` moves x into the child anchored at c. To measure the resulting change in volume across the tree, write $K_v = K(D_v)$, $V_v = V(D_v)$, and define

$$Z = \prod_v V_v. \tag{4}$$

Initially $Z = 1$, and positive node volumes give $0 < Z \leq 1$. Each item has one coordinate, so the product region always has dimension n. This volume is not by itself the fraction of total orders consistent with the transcript, as we will see in Section 4. The following identity expresses the change in Z using the density f_x of X_x at the parent before refinement.

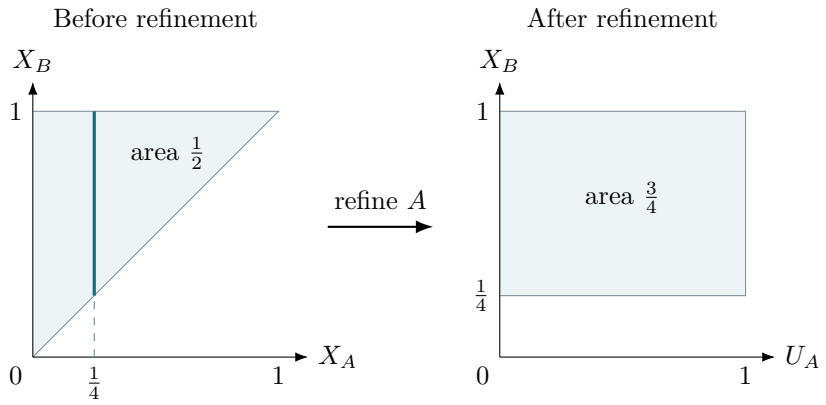


Figure 6: The highlighted section of the triangle has length 3/4. Replacing the fixed parent coordinate by a fresh unit coordinate in the child gives the rectangle on the right. Its area is 3/2 times the original area.

Lemma 3.3 (Refinement identity)**.** *Suppose every node region has positive volume. Let* Z_{before} *and* Z_{after} *be the product volumes before and after refining* x *at an interior anchor* c*. Then*

$$\frac{Z_{\text{after}}}{Z_{\text{before}}} = f_x(c). \tag{5}$$

Proof. See Appendix E. □

Fixing $X_x = c$ leaves a cross-section of the parent's feasible region with volume $V(D)f_x(c)$. The item x then leaves the parent and receives a fresh coordinate in the child. This coordinate ranges independently over $(0, 1)$, contributing a factor of one to the child's volume. All other node volumes are unchanged, so the product volume is multiplied by $f_x(c)$.

Figure 6 provides an example of the refinement operation. The parent region $0 < X_A < X_B < 1$ has area 1/2. Fixing $X_A = 1/4$ leaves the interval $X_B \in (1/4, 1)$ of length 3/4. The new child coordinate $U_A \in (0, 1)$ turns this interval into a rectangle of area 3/4. Refinement therefore multiplies Z by 3/2. Both regions represent the order $A < B$. Replacing the fixed parent coordinate by a fresh child coordinate gives the larger area.

We call the change

$$g = \log \frac{Z_{\text{after}}}{Z_{\text{before}}} = \log f_x(c) \tag{6}$$

the refinement's *volume gain*. Lemma 3.1 guarantees that a refinement requested by `Select` gains more than k in $\log Z$, whereas an answer loses at most $-\log(1/2 - 1/(100k)) = 1 + O(1/k)$. Thus an inconclusive call to `Select` actually restores volume as it moves an item. Since $Z \le 1$, these increases must be offset by losses from answers and repairs.

3.3.3 Keeping the in-node DAGs small

To keep the DAGs within the bounds of Lemma 3.1, Algorithm 7 maintains degree below k and directed paths with fewer than

$$H = 2^k \tag{7}$$

vertices between comparisons. After one arc is inserted, the degree is at most k and every directed path has fewer than $2H = 2^{k+1}$ vertices. Further insertions could exceed these bounds, so `Repair` restores the stricter bounds before the next comparison.

A repair selects items to move into children so that the parent's DAG again satisfies the degree and path-length bounds. Its routine `ChooseAnchor`(D, x) chooses an interior position for the specified item x. Lemma 3.4 gives the guarantee needed for these choices.

Algorithm 7: `Repair`(v): restore the graph bounds after an answer.

Input: The node v at which the answer was determined.
if *the graph at v has a directed path on H vertices* **then**
  $L \leftarrow$ the lexicographically first H-vertex path, in path order
  foreach $x \in L$ *in path order* **do**
    $c \leftarrow$ `ChooseAnchor`(D_v, x)
    `RefineAt`(x, c)
else if *a vertex at v has degree k* **then**
  $x \leftarrow$ the smallest label of degree k
  $c \leftarrow$ `ChooseAnchor`(D_v, x)
  `RefineAt`(x, c)

One repair suffices: every new path on H vertices contains the inserted arc, and only its endpoints can newly reach degree k. Either repair removes an endpoint of that arc, leaving a subgraph of the graph before the answer.

Lemma 3.4 (Anchor selection)**.** *Under the assumptions on D in Lemma 3.1, for any vertex x,* `ChooseAnchor`(D, x) *returns a dyadic $c \in (\ell_x, r_x)$. The routine is deterministic and uses only a neighborhood of x whose radius depends on k.*

Every position c returned by `Select` *or* `ChooseAnchor`*, written in reduced form as $c = (2i+1)/2^{m+1}$ with $m \geq 0$, satisfies*

$$\log f_x(c) \geq (1 - 1/k)m - 4. \tag{8}$$

Proof. See Appendices B and E. □

Since $m \geq 0$, each repair move loses at most four in $\log Z$. For a path repair, the dependence on m yields a stronger lower bound on the total change in $\log Z$. Section 4.1 uses this bound to limit the number of refinements.

3.4 From local choices to the comparison bound

Lemmas 3.1 and 3.4 supply answers and anchors for individual order-restriction DAGs. We also need these choices to preserve consistency with all earlier answers when items move between nodes. Write $Q(n) = n\lceil \log n \rceil$ for the cutoff on the transcript length $|\mathcal{T}|$ in Algorithm 1.

Proposition 3.5 (Consistency and termination)**.** *During the first $Q(n)$ comparisons, the state has positive feasible volume, and every represented order satisfies the transcript. Each call terminates with at most $n+2$ refinements. The strict degree and path bounds hold between calls.*

Proof. See Appendix E. □

To see Proposition 3.5's $n+2$ bound, consider one call to `Compare`. Before it answers its query, only the two queried items can move. Once moved, an item is isolated with interval $(0, 1)$, so its density is one. `Select` requires density greater than 2^k to move it again (Lemma 3.1), which prevents another move before the answer. `Repair` then moves at most n items.

We next sketch why refinements are rare over a sequence of comparisons, by tracking increases and decreases in $\log Z$. By Lemmas 3.1 and 3.3, each answer loses at most $1 + O(1/k)$ in $\log Z$, while a move requested by `Select` gains more than k. The anchor bound (8) limits each degree repair's loss to four. A path repair performs $H = 2^k$ refinements at once, but their anchors must be distinct. The same anchor bound therefore makes the whole repair profitable, with a total increase of at least $kH/4$ in $\log Z$, as Section 4.1 proves.

Degree repairs occur at most once per k comparisons on average, since each removes k comparison arcs and each comparison inserts at most one. Because $\log Z$ starts at zero and never exceeds zero, the gains from the other moves require enough comparisons to offset them. We thus obtain the following lemma.

Lemma 3.6 (Refinement count)**.** *After $q \le Q(n)$ completed comparison calls, the total number P of refinements is at most Cq/k for an absolute constant C.*

Proof. See Section 4.1. □

Finally, the refinement count and the anchor bound allow us to count the total orders consistent with the transcript. Let T_q be their number after q answers.

Theorem 3.7 (Orders remaining after comparisons)**.** *For every integer $k \ge 64$, the adversary with parameter k is deterministic and legal. There is a nonnegative $\delta_k = O(\log k/k)$, independent of n and the query sequence, such that after any q answers*

$$\log T_q \ge \log(n!) - (1 + \delta_k)q. \tag{9}$$

Proof. See Section 4. □

When a correct sorter stops, $T_q = 1$, so the theorem forces at least $\log(n!)/(1+\delta_k)$ comparisons. Section 5 chooses k as a function of n to obtain the optimal asymptotic bound in polynomial time.

4 Counting the orders that remain

We prove Theorem 3.7 in two steps. First, we compare the increases in $\log Z$ from moves requested by `Select` and from path repairs with the losses from answers and degree repairs. This proves the refinement count in Lemma 3.6. We then count orders consistent with the transcript using the volumes and the anchors in the stored tree.

4.1 Refinements are rare

Write $\eta = 1/(100k)$ for the error in the retained fraction in Lemma 3.1. Recording an answer between two anchors leaves Z unchanged. Every other answer retains at least $1/2 - \eta$ of its node's volume, so its loss in $\log Z$ is at most

$$a_\eta = -\log(1/2 - \eta) = 1 + O(1/k). \tag{10}$$

Let G be the sum of the refinement gains through q completed calls to `Compare`. Since Z starts at one and never exceeds one,

$$G \le \log Z + a_\eta q \le a_\eta q. \tag{11}$$

Proof of Lemma 3.6. Let D count degree repairs. Each removes k arcs, and each call inserts at most one arc. Thus $D \le q/k$. By (8), each such move gains at least -4. A refinement requested by `Select` has $f_x(c) > 2^k$ by Lemma 3.1, so its gain $\log f_x(c)$ is greater than k, by (6).

In a path repair, refining x_i at c_i changes the arc to x_{i+1} into the bound $X_{x_{i+1}} > c_i$. The next anchor therefore satisfies $c_{i+1} > c_i$. Thus these $H = 2^k$ anchors are distinct. The anchors with reduced denominator 2^{m+1} are $(2j+1)/2^{m+1}$ for $0 \le j < 2^m$. Thus only $\sum_{m=0}^{k-2} 2^m = 2^{k-1} - 1 < H/2$ possible anchors have $m < k - 1$. At least $H/2$ of the selected anchors therefore have $m \ge k - 1$. By (8), every refinement contributes at least -4 to $\log Z$, and each of these

$H/2$ refinements contributes at least $(1-1/k)(k-1)$ in addition. The total increase from the H refinements in this path repair is therefore at least

$$\underbrace{(1-1/k)\frac{H}{2}(k-1)}_{\substack{\text{extra gain from } H/2 \text{ anchors} \\ \text{with } m \geq k-1}} + \underbrace{(-4H)}_{\substack{-4 \text{ from all } H \text{ anchors,} \\ \text{including those with } m<k-1}}$$

$$= H\left(\frac{k}{2} - 5 + \frac{1}{2k}\right) \geq \frac{kH}{4}.$$

Every path repair in a completed call to `Compare` contributes all H refinements to P, the total number of refinements after q completed calls. Path repairs contribute at least $k/4$ per refinement on average. Refinements requested by `Select` gain more than k (Lemma 3.1 and (6)), and each of the D degree repairs gains at least -4 by (8). Summing these contributions and applying (11) gives

$$\frac{k}{4}(P-D) - 4D \leq G \leq a_\eta q, \qquad D \leq q/k.$$

Rearranging proves the claim, for example with $C = 8$ since $k \geq 64$. □

The proof of Lemma 3.6 tells us that long paths help pay for their own removal: their distinct anchors force a large volume gain. Since $Z \leq 1$, these gains must be offset by losses from comparisons and degree repairs. Both losses are bounded in terms of the number of comparisons, so frequent simplification requires many comparisons.

4.2 Counting with the queried labels

Recall that T_q is the number of total orders consistent with the first q comparison answers. To lower-bound T_q, we consider all total orders obtained by inserting the unqueried labels into orders of the queried labels represented by the adversary's tree. This can include orders that the tree itself excludes. For example, if $a < b$ lie in the same child subtree and an unqueried label z is stored at the root, the tree keeps z outside that child block, while the transcript also permits $a < z < b$.

Let s be the number of labels that have appeared in queries. For the counting argument, we can omit the other labels without changing the product volume Z. To see this, consider an unqueried item. Its vertex has no incident arcs, since every arc records a comparison. `Select` requests moves only for items in the current query. `Repair` selects only vertices incident to arcs. Thus the item stays at the root. Its interval remains $(0, 1)$: no comparison involves it, and no bound can propagate to it along an arc. Its coordinate therefore contributes an independent factor of volume one. Omitting all such coordinates leaves the volume Z for the s queried labels.

To count orders of these s labels, we sample positions in the adversary's tree. Assign each child at the reduced dyadic anchor $c = (2i+1)/2^{m+1}$ the weight

$$w(c) = \frac{1}{k^2}\left(\frac{1-1/k}{2}\right)^m. \tag{12}$$

The sum over all possible anchors is $1/k$ because there are 2^m anchors for each m.

For each of the s labels independently, start at the root. At each node, enter a child with probability equal to its weight, or stop with the remaining probability and draw a uniform coordinate in $(0, 1)$ there. The stopping probability is at least $1 - 1/k$. Order the resulting positions by the tree and its anchors.

For each label x, let d_x be the depth of the node storing it, and write $c_{x,1}, \ldots, c_{x,d_x}$ for the child anchors on the path from the root to that node. The product of these weights over all

labels is

$$\mathcal{W} = \prod_{x} \prod_{j=1}^{d_x} w(c_{x,j}). \tag{13}$$

A label stored at the root contributes one, so $\mathcal{W}$ is unchanged if the product is taken over only the queried labels.

Lemma 4.1 (Orders consistent with the transcript)**.** *After any of the first $Q(n)$ calls to* `Compare`*,*

$$T_q \geq n!(1-1/k)^s Z\mathcal{W}.$$

The same inequality holds before the first call.

Proof. The rule above samples each queried label's position by following children with probabilities $w(c)$ and choosing a uniform coordinate at the node where it stops. These positions are independent and identically distributed. Ties have probability zero, so each of their $s!$ relative orders has probability $1/s!$.

The product $\mathcal{W}$ is the probability that all s labels reach the nodes where the adversary stores them. Once there, they all stop with probability at least $(1-1/k)^s$. Conditional on these choices, the independently uniform coordinates satisfy all node constraints with probability equal to the product of the node volumes. Omitting unqueried labels leaves that product equal to Z.

Multiplying these three factors gives probability at least $(1-1/k)^s Z\mathcal{W}$ of obtaining an order consistent with the transcript, by Proposition 3.5. Since each relative order has probability $1/s!$, there are at least $s!(1-1/k)^s Z\mathcal{W}$ such orders of the queried labels.

Each extends to $\binom{n}{s}(n-s)! = n!/s!$ full orders: choose the positions of the queried labels and arrange the remaining labels. Every extension respects the transcript, since no answer involves an unqueried label. Therefore

$$T_q \geq \left[s!(1-1/k)^s Z\mathcal{W}\right]\frac{n!}{s!} = n!(1-1/k)^s Z\mathcal{W}.$$

□

4.3 Bounding the number of remaining orders

Each refinement multiplies the product volume Z by $f_x(c)$ and the sampling weight $\mathcal{W}$ by $w(c)$. The anchor guarantee relates these two changes: the volume gain almost pays for the loss in sampling weight. This is the cancellation behind Theorem 3.7.

Proof of Theorem 3.7. For the first $Q(n)$ calls to `Compare`, legality and termination follow from Proposition 3.5. Later calls use fixed ranks consistent with the recorded answers. All choices are deterministic. We first prove the bound for $q < Q(n)$.

Let s be the number of queried labels and put $d_k = -\log(1-1/k) = O(1/k)$. Taking logarithms in Lemma 4.1 gives

$$\log T_q \geq \log(n!) - sd_k + \log Z + \log \mathcal{W}.$$

Since $s \leq 2q$, the term sd_k costs only $O(q/k)$. It remains to control the loss $-\log \mathcal{W}$.

A refinement through anchor c contributes $\ell(c) = -\log w(c)$ to this loss. For $c = (2i+1)/2^{m+1}$ in reduced form, (12) gives

$$\ell(c) = 2\log k + (1+d_k)m.$$

If $g = \log f_x(c)$ is the refinement's volume gain, the anchor guarantee (8) implies $m \le (g + 4)/(1 - 1/k)$. Hence

$$\ell(c) \le 2\log k + \frac{1 + d_k}{1 - 1/k}(g + 4) = ag + b,$$

where $a = (1 + d_k)/(1 - 1/k)$ and $b = 4a + 2\log k$. Thus $a \ge 1$, $a = 1 + O(1/k)$, and $b = O(\log k)$.

Let P be the number of refinements and G their total volume gain. Summing over their anchors c_r gives

$$-\log \mathcal{W} = \sum_r \ell(c_r) \le aG + bP.$$

The volume accounting bound (11) gives $G \le \log Z + a_\eta q$. Substituting both inequalities into the counting bound yields

$$\begin{aligned}\log T_q &\ge \log(n!) - sd_k + \log Z - aG - bP \\ &\ge \log(n!) - sd_k + (1 - a)\log Z - aa_\eta q - bP.\end{aligned}$$

The volume terms have combined into $(1 - a)\log Z$, which is nonnegative because $a \ge 1$ and $Z \le 1$. Discarding this term, using $s \le 2q$, and applying $P \le Cq/k$ from Lemma 3.6, we obtain

$$\log T_q \ge \log(n!) - \left(aa_\eta + 2d_k + \frac{Cb}{k}\right) q.$$

Since $a_\eta = 1 + O(1/k)$ by (10), we may take

$$\delta_k = aa_\eta - 1 + 2d_k + \frac{Cb}{k} = O\left(\frac{\log k}{k}\right).$$

This proves (9) for $q < Q(n)$. For $q \ge Q(n) \ge \log(n!)$, its right-hand side is nonpositive and $T_q \ge 1$, so the bound holds for all q. □

Intuitively, the proof of Theorem 3.7 shows that the volume gains almost pay for the losses in sampling weight. The remaining $O(\log k)$ cost per refinement totals only $O(q \log k/k)$, because Lemma 3.6 limits the number of refinements to $O(q/k)$. This gives the error term $\delta_k = O(\log k/k)$ in the theorem's bound on the number of remaining orders.

The bound on T_q also applies to tasks where the algorithm may stop with several total orders still possible (Section 6).

5 Running time

`Select` and `ChooseAnchor` inspect neighborhoods in an order-restriction DAG whose sizes depend only on k. Their cost also depends on the precision of the interval endpoints. The following lemma bounds the number of digit assignments they enumerate and the precision of the anchors they return.

Lemma 5.1 (Cost of `Select` and `ChooseAnchor`)**.** *For* `Select` *and* `ChooseAnchor` *under the assumptions of Lemmas 3.1 and 3.4, there is a computable budget E_k, depending only on k, such that each call enumerates at most E_k digit assignments. If the stored interval bounds have at most Λ fractional binary digits, every returned anchor has at most $\Lambda + \log E_k$ such digits.*

Proof. See Theorem B.1 and Appendix F. □

A fixed program generates the parameters and E_k for $k = 64, 128, \ldots$, writing each completed tuple in binary. On input size n, run this program for n steps and choose the largest completed k with $E_k \le n$, denoted by $k(n)$. If no such value exists, answer from the fixed label order. The

simulation writes at most n bits, so parsing its output and selecting a tuple also take polynomial time.

For every k in this sequence, both its generation time and E_k are finite. That tuple therefore qualifies for all sufficiently large n, proving $k(n) \to \infty$.[2] Appendix F shows that the interval endpoints require polynomially many bits, and that tree traversal and updates to the in-node DAGs take polynomial time. Each comparison calls `Select` at most three times and `ChooseAnchor` at most n times, giving a fixed polynomial bound on bit operations per query, independent of the length of the interaction.

Proof of Theorem 1.1. The adversary is deterministic and polynomial-time. The fixed label order gives legal answers, and legality for every parameter $k \geq 64$ follows from Theorem 3.7. For sufficiently large n, the procedure above selects a value $k(n)$ with $E_{k(n)} \leq n$. A correct sorter can then stop only when $T_q = 1$, which requires

$$q \geq \frac{\log(n!)}{1 + \delta_{k(n)}} = (1 - o(1))n \log n,$$

since $\delta_k = O(\log k/k)$ and $k(n) \to \infty$. □

6 Consequences and further questions

The bound on the number of total orders consistent with the answers (Theorem 3.7) also yields comparison lower bounds when the algorithm starts or finishes with a partial order. The two settings have different information bounds: initial relations reduce what remains to be learned, whereas a target partial order specifies what the algorithm must establish.

6.1 Producing a prescribed partial order

Let R be a partial order on n slots. In *partial-order production*, the input consists of n keys with no known relations. The algorithm must assign one key to each slot so that the key in slot u is smaller than the key in slot v whenever $u <_R v$. Sorting corresponds to a chain. Finding the minimum corresponds to one slot preceding all the others, with no other required relations.

A fixed assignment is valid for exactly $e(R)$ total orders of the keys. The information-theoretic lower bound is therefore

$$I(R) = \log(n!) - \log e(R).$$

Cardinal et al. [14] give a deterministic polynomial-time algorithm using $I(R) + o(I(R)) + O(n)$ comparisons.

Corollary 6.1 (Partial-order production)**.** *The adversary of Theorem 1.1 forces every correct comparison algorithm producing an n-element partial order R to make at least*

$$(1 - o(1))I(R)$$

comparisons. The $o(1)$ term depends only on n.

Proof. See Appendix H. □

Its leading term matches Cardinal et al.'s upper bound when $I(R) = \omega(n)$. The information bound can be smaller than the task's comparison complexity. For example, producing a minimum has $I(R) = \log n$ and requires $n - 1$ comparisons.

[2] These parameters give an asymptotic guarantee, with no claim of practical running time.

6.2 Sorting from supplied partial information

Now let P describe relations already known between the input keys, supplied as a DAG. The sorter must identify one of its $e(P)$ linear extensions, so the information bound is $\log e(P)$.

Corollary 6.2 (Supplied partial information)**.** *There is a deterministic adversary that can be initialized from any n-element partial order P in polynomial time, gives legal answers in polynomial time per comparison, and forces every correct sorter to make at least*

$$(1 - o(1)) \log e(P) - O(n \log \log n)$$

additional comparisons. Both error terms are uniform over P.

Proof. See Appendix H. □

The construction uses Cardinal et al.'s polynomial-time procedure [14] to extend P to a *weak order*: a sequence of antichain layers, with every key in an earlier layer preceding every key in a later one. The additional relations reduce $\log e(P)$ by at most $O(n \log \log n)$. Run an independent copy of our adversary within each layer and answer between layers from their fixed order. All copies use the same parameters, chosen from the total size n, so the guarantee also covers small layers.

The comparison bound approaches $\log e(P)$ when $\log e(P) = \omega(n \log \log n)$. Kahn and Kim's adversary already forces $\Omega(\log e(P))$ for every P. The improvement here is the coefficient approaching one in this regime.

6.3 Sampling linear extensions

Huber [23] gives a near-linear expected-time exact sampler for height-two posets of bounded degree. Our contraction estimate also gives a sampler for any fixed degree and directed-height bounds.

Corollary 6.3 (Exact sampling)**.** *Fix integers $d, h \geq 1$. Given adjacency lists of an n-vertex DAG of degree at most d and with directed paths of at most h vertices, one can sample a linear extension exactly uniformly using $O(n \log n)$ arithmetic operations and fair random bits in expectation.*

Proof. See Appendix H.5. □

6.4 Open questions

Can a deterministic polynomial-time adversary force $(1 - o(1)) \log e(P)$ further comparisons for every supplied partial order P? The additive loss in Corollary 6.2 can overwhelm the information that remains. For two disjoint chains of length $n/2$, $\log e(P) = \Theta(n)$. For a chain on $n - 1$ items and one unrelated item, $\log e(P) = \log n$.

Reducing the additive loss to $O(n)$ would already extend the useful range. Cardinal et al. [14, Section 4] exhibit partial orders P for which every weak-order extension W satisfies $\log e(W) \leq \log e(P) - \Omega(n \log \log n)$. A sharper construction must therefore preserve more of the possible interleavings between layers.

Initial partial information also arises when sorting points whose locations are known only to lie in specified intervals [15, 16]. Extending our adversary to those models would require answers compatible with realizable coordinates and with the permitted geometric operations.

Volume approximation for order-restriction DAGs. The closures of the regions $K(D)$ are *marked order polytopes* [17]: fixed marked elements encode the interval endpoints. This connects the local calculations of this paper to a global approximation problem. For fixed integers d, h, can a deterministic algorithm, given an order-restriction DAG of degree at most d, directed paths of at most h vertices, rational interval endpoints, and $0 < \varepsilon < 1$, compute

$$(1-\varepsilon)V(D) \le \widehat{V} \le (1+\varepsilon)V(D)$$

in time polynomial in the input bit length and $1/\varepsilon$, with the polynomial allowed to depend on d, h? Assume $V(D) > 0$. This asks for a deterministic fully polynomial-time approximation scheme (FPTAS). Theorem B.1 supplies local enclosing intervals, but its enumeration bound is not polynomial in the inverse accuracy.

A concrete first case is $h = 2$, $d = 3$, with every interval equal to $(0, 1)$. Orient the edges of a bipartite graph from one side to the other. Reflecting the coordinates on the second side changes $X_a < X_b$ into $Y_a + Y_b < 1$, preserving volume. Thus this case is also a volume problem for bipartite independent-set polytopes. Recent deterministic schemes give approximation guarantees for suitable truncations of these polytopes [18]. Whether corresponding guarantees hold for the full interval $(0, 1)$ is the question posed here.

The remaining gap to deterministic counting. When every interval is $(0, 1)$, $V(D) = e(P)/n!$ for the partial order P represented by D [9]. For general P, a deterministic approximation scheme for $e(P)$ remains open [19, Sections 3.6 and 12.2]. Even a deterministic constant-factor approximation would imply an FPTAS. Apply it to the disjoint union of copies of the partial order P, divide by the number of ways to interleave the copies, and take a root. The corresponding DAG is a disjoint union of copies of D, so the degree and path-length bounds are preserved. Thus the same argument applies for each fixed d, h (Appendix H.3). An FPTAS for arbitrary P would also resolve the supplied-partial-order question above: estimate both comparison branches and retain the one with the larger estimate (Appendix H.4). Our adversary does not supply such a counting algorithm: it uses local probability information and makes additional restrictions whose total cost is controlled by the analysis. Can these local calculations be made sufficiently efficient at high accuracy to estimate global volume?

You are now entering AI-slop-land.

The appendices of this paper have not been carefully reviewed by a human. The main sorting result has been formalized in Lean.

A Canonical intervals and densities

We prove Lemma 3.2 for order-restriction DAGs. We then establish the density bounds used to choose anchors and analyse probability certificates.

A.1 Normalization and restriction

Lemma A.1 (Normalization)**.** `Normalize` *preserves* $K(D)$*. The resulting region has positive volume if and only if every interval has positive width; in that case the intervals are canonical.*

Proof of Lemmas A.1 and 3.2. Every propagated bound is already implied by an arc and an interval bound. A forward topological pass propagates every lower bound to all its successors; the reverse pass propagates every upper bound to all its predecessors. Thus normalization preserves the region and gives $\ell_x \le \ell_y$ and $r_x \le r_y$ along every arc $x \to y$.

Positive volume requires positive widths. Conversely, suppose all widths are positive. Choose numbers $0 < \theta_x < 1$ increasing in a topological order, and set

$$X_x = (1 - \theta_x)\ell_x + \theta_x r_x.$$

Along an arc $x \to y$, increasing θ_x to θ_y strictly increases this value, because $r_x > \ell_x$. Replacing the endpoints by ℓ_y, r_y cannot decrease it. Thus $X_x < X_y$, all interval bounds hold strictly, and a small open neighborhood is feasible.

Now fix $X_x = c$ in the normalized interval and delete its coordinate. Every incident inequality becomes exactly the bound imposed by `Restrict`, while all other inequalities persist. This proves that the resulting region is the section at c. Lower bounds propagate forward and upper bounds backward. Any earlier lower bound that reached a successor through x is superseded by $c > \ell_x$; any earlier upper bound that reached a predecessor through x is superseded by $c < r_x$. The normalized intervals of the remaining coordinates are therefore

$$\begin{cases} (\ell_y, \min(r_y, c)), & y \text{ precedes } x, \\ (\max(\ell_y, c), r_y), & y \text{ succeeds } x, \\ (\ell_y, r_y), & y \text{ is incomparable with } x. \end{cases} \tag{14}$$

A predecessor retains positive width because $r_y > \ell_y$ and $c > \ell_x \ge \ell_y$. A successor does too because $r_y > \ell_y$ and $c < r_x \le r_y$. Other widths are unchanged. The criterion just proved shows that the section has positive volume. Since this holds for every $c \in (\ell_x, r_x)$, that interval is exactly the coordinate projection. The volume identity in Lemma 3.2 follows from the definition of $f_x(c)$ as section volume divided by $V(D)$. □

A.2 Bounds from convex sections

Let V be the set of coordinates, let $N = |V|$, and let $K \subseteq \mathbb{R}^V$ be their feasible region. For a coordinate x and a real number t, write

$$K_x(t) = \{(z_i)_{i \ne x} : z \in K,\ z_x = t\}, \qquad S_x(t) = \mathrm{vol}_{N-1} K_x(t).$$

Here $K_x(t)$ is the $(N-1)$-dimensional section obtained by fixing coordinate x at t. The uniform distribution on K therefore gives x the density

$$f_x(t) = \frac{S_x(t)}{\operatorname{vol}_N K}.$$

These formulas define f_x pointwise, and we use this definition throughout. If some coordinates have already been fixed, we use the same definitions in the remaining free coordinates. In dimension zero, the volume of the singleton consisting of the empty assignment is one.

Lemma A.2 (Density of a linear projection)**.** *Let $C \subseteq \mathbb{R}^d$ be a bounded open convex region of positive volume, and let φ be a nonzero linear functional with $\varphi(C) = (\ell, r)$ of width W. For X uniform on C, the density g of $\varphi(X)$ satisfies*

$$g(t) \le \frac{d}{W}, \qquad g(t) \ge \frac{4^{-(d-1)}}{W} \quad (\ell + W/4 \le t \le r - W/4).$$

Proof. An invertible linear change of coordinates makes φ the first coordinate. Its constant Jacobian preserves uniformity on the transformed region, so it suffices to prove the bounds for a coordinate projection. Write $S(t)$ for the section volume at that coordinate's value t; then $g(t) = S(t)/\operatorname{vol} C$.

Choose points in $\overline{C}$ whose first coordinates are ℓ and r. Such points exist because C is bounded. For $\ell < t < s < r$, join every point of the section at t to the point at r. Since C is open and convex, the image with first coordinate s lies in the section of C there. Each of its $d-1$ directions is scaled by $(r-s)/(r-t)$, so

$$S(s) \ge \left(\frac{r-s}{r-t}\right)^{d-1} S(t) \qquad (t < s). \tag{15}$$

Joining instead to the point at ℓ gives

$$S(s) \ge \left(\frac{s-\ell}{t-\ell}\right)^{d-1} S(t) \qquad (s < t). \tag{16}$$

Normalize these inequalities and integrate over s on both sides of t:

$$1 \ge g(t)\left(\int_\ell^t \left(\frac{s-\ell}{t-\ell}\right)^{d-1} ds + \int_t^r \left(\frac{r-s}{r-t}\right)^{d-1} ds\right) = \frac{W g(t)}{d}.$$

Thus $g(t) \le d/W$.

For the lower bound, choose t_0 with $g(t_0) \ge 1/W$, which exists because g integrates to one over an interval of length W. For any $s \in [\ell + W/4, r - W/4]$, the appropriate ratio in (15) or (16) is at least $1/4$. Consequently

$$g(s) \ge 4^{-(d-1)} g(t_0) \ge \frac{4^{-(d-1)}}{W}.$$

The case $s = t_0$ satisfies the same bound directly. □

Lemma A.3 (Coordinate density bounds)**.** *Let D be a canonical order-restriction DAG with $V(D) > 0$, and suppose x has at most B other comparable coordinates and canonical interval (ℓ, r) of width W. Its density satisfies*

$$f_x(t) \le \frac{B+1}{W}, \tag{17}$$

$$f_x(t) \ge \frac{4^{-B}}{W} \qquad (\ell + W/4 \le t \le r - W/4). \tag{18}$$

Proof. Fix all coordinates incomparable with x, at values leaving a nonempty region. By the section formula in Lemma 3.2, each such fixing leaves x's canonical interval unchanged. The remaining region C is open and convex, has dimension $d \le B+1$, and still projects onto (ℓ, r) in coordinate x. Lemma A.2 therefore gives both required bounds for the density of x in C.

Average over the fixed coordinates. If ω denotes their values, C_ω the remaining region, and g_ω its coordinate density, integration of each section gives the pointwise identity

$$f_x(t) = \frac{\int \mathrm{vol}(C_\omega) g_\omega(t)\, d\omega}{\int \mathrm{vol}(C_\omega)\, d\omega}.$$

Empty slices contribute zero. The bounds on g_ω are uniform in ω, so they also hold for f_x. □

Lemma A.4 (A bound for small differences)**.** *Let X be uniform on $K(D)$ for a canonical order-restriction DAG D with $V(D) > 0$, and suppose x has at most B other comparable coordinates and canonical interval of width W. Let Y be a fixed number or X_y for any distinct free coordinate y. For every interval I of length a,*

$$\mathbb{P}(X_x - Y \in I) \le \frac{(B+1)a}{W}. \tag{19}$$

Proof. For a fixed number Y, integrate the coordinate-density upper bound over $I + Y$. Now let $Y = X_y$ for any free coordinate $y \neq x$. As above, fix all coordinates incomparable with x. The remaining convex region C has dimension $d \le B+1$, and x retains its width W. We show that the projection $X_x - X_y$ has width at least W in each such region.

If y is incomparable with x, it is now fixed, so the projection is a translate of X_x and has width W. Otherwise y remains free. Write (ℓ_z, r_z) for canonical intervals in C; x's endpoints are unchanged. If a directed path goes from x to y, then $r_y \ge r_x$. Fixing $X_y = t \in (\ell_y, r_y)$ leaves x the interval $(\ell_x, \min(r_x, t))$, by Lemma 3.2. As t approaches r_y, its width tends to W. On that section the difference $X_x - t$ varies over an interval of the same width. Thus the projection of the whole region C has width at least W. If a directed path goes from y to x, let t approach ℓ_y instead. Here $\ell_y \le \ell_x$, and the remaining interval $(\max(\ell_x, t), r_x)$ again has width tending to W.

Lemma A.2 bounds the conditional density of the difference by $d/W \le (B+1)/W$. Integrating over I and averaging over the fixed coordinates proves (19). If both coordinates have at most B comparable coordinates, exchanging x and y gives the same bound with $\max(W_x, W_y)$ in the denominator. □

A.3 Density in a central interval

A density is *log-concave* if its positive support is convex and its logarithm is concave there. The coordinate densities above are log-concave pointwise: the uniform density on K is log-concave, and integrating out the other coordinates preserves this property by Prékopa's marginal theorem [13, Theorem 8]. We finish with the density bound used for positions having substantial probability on each side.

Lemma A.5 (Density at central positions)**.** *Let X have a log-concave density f. If $\mathbb{P}(X \le c) \ge p$ and $\mathbb{P}(X \ge c) \ge p$ for some $0 < p \le 1/2$, then*

$$f(c) \ge p \sup_t f(t).$$

Proof. For $s \le c \le t$, log-concavity gives

$$f(s) f(t) \le f(c) f(s + t - c).$$

Indeed, c and $s+t-c$ are complementary convex combinations of s,t; multiply the two log-concavity inequalities. Fixing $t \geq c$ and integrating over $s \leq c$ gives

$$pf(t) \leq f(c) \int_{-\infty}^{c} f(s+t-c)\, ds \leq f(c).$$

Here the first inequality uses the mass at least p to the left of c, and the last uses that a density integrates to one. Fixing $s \leq c$ and integrating over $t \geq c$ instead uses the right-hand mass and gives $pf(s) \leq f(c)$. Taking the supremum over both sides of c proves the claim. □

Lemma A.6 (Density in a central interval)**.** *Let X be uniform on $K(D)$ for an order-restriction DAG D with $V(D) > 0$. Suppose $J = [u, v] \subset (\ell_x, r_x)$ satisfies*

$$\mathbb{P}(X_x \leq u) \geq \tfrac{1}{4}, \qquad \mathbb{P}(X_x \in J) \geq \tfrac{1}{4}, \qquad \mathbb{P}(X_x \geq v) \geq \tfrac{1}{4}.$$

Then, for every $c \in J$,

$$f_x(c) \geq \frac{1}{16|J|}.$$

Proof of Lemma A.6. Put $M = \sup_t f_x(t)$. The inside-mass condition gives $M|J| \geq 1/4$. Every $c \in J$ has probability at least $1/4$ on each side, so Lemma A.5 gives $f_x(c) \geq M/4 \geq 1/(16|J|)$. □

B Local calculations on order-restriction DAGs

We construct the routines in Lemmas 3.1 and 3.4 for a single order-restriction DAG. Both use lists of intervals enclosing the queried coordinates. A pair of disjoint intervals certifies an order, and a list of intervals can certify how much probability lies on each side of a candidate anchor interval.

B.1 A coordinate update with finitely many choices

Let D be an order-restriction DAG with canonical interval endpoints ℓ_z, r_z. Write G for its underlying graph. For a current assignment Y, the interval permitted at coordinate z has endpoints

$$L_z = \max(\{\ell_z\} \cup \{Y_u : u \to z\}), \qquad U_z = \min(\{r_z\} \cup \{Y_u : z \to u\}).$$

Fix a power of two S. For a digit $j \in \{0, \dots, S-1\}$, update

$$Y_z \leftarrow \frac{Y_z + (S-1-j)L_z + jU_z}{S}. \tag{20}$$

The coefficients are nonnegative and sum to one. Thus an assignment satisfying the interval bounds and arc inequalities weakly continues to do so after an update, including when $L_z = U_z$.

Choosing the digit uniformly preserves the uniform distribution on the feasible region. Start from a uniform feasible assignment and condition on all coordinates other than z. Then Y_z is uniform on (L_z, U_z). Its normalized position $r = (Y_z - L_z)/(U_z - L_z)$ is sent to

$$r' = \frac{r+j}{S}.$$

For a fixed digit, this is uniform on the corresponding one of the S equal subintervals. Averaging over the digits makes r' uniform on $(0,1)$ again.

B.2 The enclosing-interval guarantee

In Algorithm 3, call the vertices at distance ρ from T the *boundary*, and those at smaller distance the *interior*. A *sweep* is one repetition of the updates over the interior vertices. If there are b' interior vertices, the list has

$$N = S^{b't} \tag{21}$$

entries, counting repetitions. Each interior vertex has all its neighbors in the retained DAG.

Theorem B.1 (Local enclosing intervals)**.** *Let $d, h \geq 1$ be integers and let $\delta, \alpha \in (0,1)$ be rational. There are computable parameters S, t, ρ and a bound E, depending only on d, h, δ, α, with the following property.*

Suppose D is an order-restriction DAG with canonical dyadic intervals, degree at most d, directed paths of at most h vertices, and $V(D) > 0$. Write $K = K(D)$. For any nonempty set T of one or two coordinates, Algorithm 3 returns a list satisfying two guarantees:

1. *A uniformly selected entry and a uniform feasible assignment X can be drawn jointly so that $A_x \leq X_x \leq B_x$ for every $x \in T$.*
2. *At most a fraction α of its entries have $\max_{x \in T}(B_x - A_x) > \delta$.*

The calculation enumerates at most E digit assignments. If the stored bounds have at most Λ fractional binary digits, every computed endpoint has at most $\Lambda + \log E$ fractional digits.

Proof. The enclosure and dependence arguments follow below. See Appendix C for the width and enumeration bounds, and Appendix F for the precision bound. □

The enclosure guarantee holds for every finite number of sweeps and every choice of radius $\rho \geq 1$. Start a full assignment uniformly on K and update the entire graph with independent uniform digits. Share its digits with the two endpoint calculations at interior vertices. The update is increasing in every coordinate, since maxima, minima, and the nonnegative linear combination in (20) are increasing. Thus it preserves the inequalities $A_z \leq X_z \leq B_z$ at an interior update. At a boundary vertex the full position always remains between its fixed interval endpoints, even as it changes during the full calculation. The full assignment remains uniform, and every interior digit sequence has probability $1/N$. This proves the first guarantee.

B.3 How far a sweep can reach

For the width estimate, consider lower and upper sweeps that update the full graph. In either calculation, the update of a coordinate uses the new values of its predecessors, the old values of its successors, and its own old value. The resulting values are independent of the choice of topological order because these dependencies specify every update.

To trace a dependency backwards through one sweep, follow new values along incoming arcs. This follows a directed path with at most $h - 1$ arcs. Moving from that path to an old value crosses at most one more arc. A value after t sweeps thus depends only on digits, bounds and initial values within distance ht. With $\rho = ht + 1$, every update in this dependency calculation is in the retained interior. Its initial values, stored bounds, and digits are the same in the local and full calculations. Consequently the queried endpoints agree exactly for every assignment of the digits, with arbitrary digits outside the retained region.

Appendix C bounds the widths on the full graph. Exact agreement transfers that estimate to the local lists.

B.4 From intervals to answers and anchors

The enclosure guarantee makes the fractions counted by `Select` lower bounds on the corresponding answer probabilities.

The routine `ChooseAnchor` uses a one-coordinate list. It tests whether a proposed interval contains at least a quarter of the mass, with at least a quarter on either side. The test counts entries lying wholly in each of these three regions. On success it rounds within the tested interval. On failure it rounds in the middle half of the coordinate's range. Appendix D gives the endpoint selection and shows that a failed test identifies a short coordinate interval. The gain and precision estimates in Appendix E complete the proof of Lemmas 3.1 and 3.4.

Algorithm 8: `ChooseAnchor`(D, x): choose an interior position.

Input: A canonical order-restriction DAG D and a coordinate x with interval (ℓ_x, r_x).
Output: A dyadic position $c \in (\ell_x, r_x)$.
$J \leftarrow$ `CentralInterval`(D, x)
if J *is* `FAIL` **then**
  $W_x \leftarrow r_x - \ell_x$
  $J \leftarrow [\ell_x + W_x/4,\ r_x - W_x/4]$
return `RoundUp`(J)

C Why the enclosing intervals become narrow

We analyse lower and upper calculations on the full graph, with a fresh independent uniform digit at every vertex in every sweep. By Appendix B.3, the queried endpoints have the same distribution as those in the finite local list.

C.1 The decrease after one sweep

Write $\Delta_z = B_z - A_z$. Superscripts old and new refer to the start of a sweep and the value after a coordinate's update. For a vertex z, the shared digit j_z gives

$$\Delta_z^{\mathrm{new}} \le \frac{1}{S}\Delta_z^{\mathrm{old}} + \frac{S-1-j_z}{S}\max_{u\to z}\Delta_u^{\mathrm{new}} + \frac{j_z}{S}\max_{z\to u}\Delta_u^{\mathrm{old}}, \tag{22}$$

with an empty maximum equal to zero. The stored interval bounds in the two calculations are identical.

Let $B_h(z)$ be the vertices at undirected distance at most h from z. Put $D = \max_{u\in B_h(z)}\Delta_u^{\mathrm{old}}$ and $a_u = (S-1-j_u)/S$. At z, (22) gives

$$\Delta_z^{\mathrm{new}} \le (1-a_z)D + a_z \max_{u\to z}\Delta_u^{\mathrm{new}}.$$

Follow a predecessor attaining the maximum, and continue backwards to a source. This path has at most h vertices; all its vertices and their successors lie in $B_h(z)$. At the source the predecessor term is zero. Substitution along the path bounds the new width by $(1-\prod_u a_u)D$.

Write $\Delta_z(s)$ for the width after s sweeps. Call the pair (z,s) *bad* if, during sweep s, the digit at z or any vertex preceding z on a directed path equals $S-1$. Otherwise all factors a_u in the preceding expansion are at least $1/S$, and there are at most h of them. Put $\beta = 1 - S^{-h}$. Consequently,

$$\Delta_z(s) \le \kappa_z(s)\max_{u\in B_h(z)}\Delta_u(s-1), \qquad \kappa_z(s) = \begin{cases} 1, & (z,s) \text{ is bad,} \\ \beta, & \text{otherwise.} \end{cases} \tag{23}$$

C.2 Choosing the parameters

Write $r = 2$ for the bound on the number of queried coordinates. Both $B_h(z)$ and the set consisting of z and its directed-path predecessors have at most M vertices. Choose

$$M = (d+1)^{h+1}, \qquad S = \text{the least power of two at least } 4M^2. \tag{24}$$

The bad event in (23) involves only fresh digits from the current sweep. Its conditional probability is at most $M/S \le 1/(4M)$. With $\beta = 1 - S^{-h}$, choose the smallest positive integer p satisfying the rational test

$$\beta^p \le \frac{1}{4M}. \tag{25}$$

Consequently,

$$\mathbb{E}[\kappa_z(s)^p \mid \text{earlier sweeps}] \le \frac{1}{4M} + \beta^p \le \frac{1}{2M}.$$

This estimate uses all directed-path predecessors in defining the bad event, so it also covers the predecessor choices made in the one-sweep expansion. No independence between vertices within a sweep is required.

Choose the smallest positive integer t satisfying

$$2^{-t} \le \frac{\alpha\delta^p}{r}, \qquad \rho = ht + 1. \tag{26}$$

These choices are computable: $0 < \beta < 1$, and both tests compare rational numbers. Retain radius ρ around the queried set T, freezing the outer boundary as in Algorithm 3.

The widths before sweep s are fixed when we condition on earlier sweeps. Raising (23) to the power p, bounding the maximum by a sum, and taking expectations gives

$$\mathbb{E}\Delta_z(s)^p \le \frac{1}{2M} \sum_{u \in B_h(z)} \mathbb{E}\Delta_u(s-1)^p.$$

All initial widths are at most one and $|B_h(z)| \le M$. Induction gives $\mathbb{E}\Delta_z(s)^p \le 2^{-s}$ for every vertex. At sweep t, Markov's inequality gives $\mathbb{P}(\Delta_z(t) > \delta) \le 2^{-t}\delta^{-p} \le \alpha/r$. A union bound over the at most r queried positions proves the width guarantee.

Each radius-ρ neighborhood has at most $1 + d + \cdots + d^\rho \le (d+1)^{\rho+1}$ vertices. We may therefore use

$$b = r(d+1)^{\rho+1}, \qquad E = S^{bt} \tag{27}$$

as bounds on the retained vertices and enumerated digit assignments. All these choices depend only on d, h, δ, α.

D Calibration of the finite certificates

We verify the central-interval routine and choose the finite calculation's parameters. The purpose of the calibration is to ensure that a failed certificate identifies a coordinate whose short interval gives a high density throughout its middle half.

D.1 The central-interval test

For the one-coordinate list $[A_i, B_i]$, $1 \le i \le N$, take u to be the $N/4$-th upper endpoint in sorted order and v to be the $3N/4$-th lower endpoint. The alphabet size S is divisible by four, and the queried coordinate is interior, so $N/4$ is an integer. Return `FAIL` if $u \ge v$ or fewer than $N/4$ list intervals lie wholly in $[u, v]$. Otherwise return $[u, v]$.

At least $N/4$ intervals end at or below u, and at least $N/4$ begin at or above v. On success, at least $N/4$ intervals also lie wholly inside $[u, v]$. Enclosure certifies mass at least $1/4$ in each of the three regions. The true distribution is continuous and supported on (ℓ_x, r_x), so these masses imply $\ell_x < u < v < r_x$. The endpoints are dyadic, and sorting and counting finite lists are deterministic.

D.2 The width at which certificates succeed

The graph bounds limit how many coordinates can be comparable with a queried coordinate. Define

$$B = (k+1)^{2H+1}. \tag{28}$$

In an order-restriction DAG of degree at most k and directed paths of at most $2H$ vertices, every comparable coordinate lies within undirected distance $2H$. Thus B bounds the number of other comparable coordinates. Set

$$w_* = 2^{-2kB}. \tag{29}$$

This width calibrates the finite calculations. The algorithms test the computed certificates directly.

Lemma D.1 (When certification succeeds)**.** *Let D be a canonical order-restriction DAG with dyadic interval bounds, $V(D) > 0$, degree at most k, and directed paths of at most $2H$ vertices. If any coordinate involved in a comparison has interval width at least w_*, comparison certification succeeds. If $W_x \geq w_*$, then* `CentralInterval`(D, x) *succeeds.*

Proof of Lemma D.1. Calibrate both routines using Theorem B.1 with

$$d = k, \qquad h = 2H, \qquad \alpha = \frac{\eta}{2}, \qquad \delta = \frac{\eta w_*}{8(B+1)}. \tag{30}$$

These parameters are fixed for the interaction. They do not depend on the queried widths.

Suppose one queried coordinate x has $W_x \geq w_*$. If the computed intervals overlap and both have width at most δ, the enclosed positions differ by at most 2δ. Lemma A.4 therefore bounds the unresolved fraction by

$$1 - p - q \leq \alpha + \mathbb{P}(|X_x - X_y| \leq 2\delta) \leq \alpha + \frac{4(B+1)\delta}{w_*} = \eta.$$

Thus $\max(p, q) \geq (1-\eta)/2 > 1/2 - \eta$, so comparison certification succeeds. For a threshold c, an unresolved interval contains c. If its width is at most δ, its enclosed position is within δ of c, giving the stronger bound

$$\alpha + \frac{2(B+1)\delta}{w_*} = \frac{3\eta}{4}. \tag{31}$$

It remains to show that `CentralInterval` succeeds when $W_x \geq w_*$. By (31), at most a fraction $\tau = 3\eta/4$ of its intervals contain any fixed real number. This bound applies in particular to the selected endpoints u and v. An interval extending below u either ends strictly below u or contains u. By the rank defining u, fewer than $N/4$ intervals end strictly below it. Thus at most $(1/4 + \tau)N$ intervals extend below u. Similarly, at most $N/4$ intervals begin strictly above v, so at most $(1/4 + \tau)N$ extend above v.

At least $(1/2 - 2\tau)N \geq N/4$ intervals therefore lie wholly inside $[u, v]$, since $\eta < 1/32$. Enclosure gives this interval mass at least $1/4$. Continuity forces $u < v$, so both success tests pass. □

E Selection guarantees and state updates

E.1 Rounding an anchor

For a nondegenerate closed interval $J = [u, v] \subset (0, 1)$, choose the largest power-of-two spacing h no greater than its length. The routine `RoundUp` rounds u up to a multiple of h:

$$h = 2^{-r}, \qquad h \le |J| < 2h, \qquad c = h\lceil u/h \rceil . \tag{32}$$

Lemma E.1 (Rounding to a binary fraction)**.** *The value c returned by* `RoundUp`(J) *belongs to J. Writing $c = (2i + 1)/2^{m+1}$ in reduced form, it satisfies*

$$|J| < 2^{-m}. \tag{33}$$

Proof. The rounded value obeys $u \le c < u + h \le v$, so $c \in J$. The choice of spacing gives $h \le |J| < 2h$. Reduction of the fraction cannot increase its denominator, so $m + 1 \le r$ and $|J| < 2h \le 2^{-m}$. □

Proof of Lemmas 3.1 and 3.4. The interval comparisons in Appendix B certify lower probability bounds. If the larger bound is at least $1/2 - \eta$, `Select` returns its answer, breaking a tie in favor of the first queried label. On failure it rounds in the middle half of a queried coordinate's interval. For `ChooseAnchor`, a successful central test supplies the interval to round. On failure it too rounds in the middle half.

For a successful central test, Lemma A.6 gives $f_x(c) \ge 1/(16|J|)$. By Lemma E.1, $\log f_x(c) > m-4$. For either type of failed test, Lemma D.1 gives $W_x < w_* = 2^{-2kB}$ for the selected coordinate. Write $L = \log(1/W_x) > 2kB$. The density estimate (18) gives $\log f_x(c) \ge L - 2B > k$ throughout the middle half. That interval has length $W_x/2$, so $m < L + 1$ by Lemma E.1. Thus

$$\log f_x(c) \ge L - 2B > (1 - 1/k)L > (1 - 1/k)m - 4.$$

Both intervals used for rounding lie strictly inside the coordinate's range. □

E.2 Refinement and child admission

Proof of Lemma 3.3. By (3), restriction multiplies the parent volume by $f_x(c)$. Inserting the isolated vertex into the child replaces K_u by $K_u \times (0, 1)$, which has the same volume. The ratio of the products is therefore $f_x(c)$. □

Lemma E.2 (Refinement and child admission)**.** *Suppose the state represents only orders consistent with the transcript and every node has positive feasible volume. An interior refinement preserves both properties.*

Proof. Take feasible positions representing any order after the move. Remove the fresh coordinate of x from the child and restore $X_x = c$ in the parent. The parent positions satisfy the old inequalities by construction. Keeping the other coordinates fixed, we can perturb X_x slightly to either side of c, since its inequalities are strict. These two old orders place x immediately before and immediately after the child block. Both respect the transcript, so it contains no comparison between x and an item in the block. In the new order, x can occur anywhere inside that block, with its order relative to all outside items preserved. For a new child the block was empty, and the same argument places x at its former parent position. Positivity follows from Lemma 3.2 and the refinement identity. □

A refinement can change which orders the state represents. For instance, a move into an existing child allows interleavings inside that child that were previously excluded. The invariant concerns consistency with the transcript.

E.3 Restoring the graph bounds

Lemma E.3 (One-arc repair)**.** *Suppose all node graphs have degree below k and directed paths with fewer than H vertices before an answer is recorded. Recording the answer and executing* `Repair` *restores these bounds. During this operation, degrees are at most k and directed paths have fewer than $2H$ vertices.*

Proof. An answer involving an anchor adds only an interval bound. An answer between free items adds at most one arc, say $x \to y$. Its certified positive probability ensures that the resulting region is nonempty, so the arc cannot create a directed cycle. Only x and y can reach degree k. Every new path consists of two old paths joined by this arc, so it has fewer than $2H$ vertices.

If an H-vertex path now exists, it must contain $x \to y$. Refining its entire vertex set removes both endpoints. Otherwise a degree repair removes an endpoint that has reached degree k. Either repair deletes the new arc; the remaining parent DAG is a subgraph of the previously valid DAG. Each child receives only isolated vertices, so its bounds remain valid too. If neither violation occurs, no repair is needed. □

E.4 Consistency and termination

Proof of Proposition 3.5. Initially every node has positive volume, every order is represented, and the strict node graph bounds hold. We induct over completed calls and, within each call, over its changes to the state.

Before answering, a refinement deletes one vertex and adds an isolated vertex to a child, so the strict DAG bounds continue to hold. The anchor routine returns an interior position, as proved in Lemma 3.1. Lemma E.2 preserves positive node volumes and consistency with the transcript. Normalization preserves the represented region by Lemma A.1.

If the queried labels lie in different child subtrees, their anchors determine the answer in every represented order. Otherwise, the answer guarantee in Lemma 3.1 ensures that the chosen answer retains at least $1/2 - \eta > 0$ of its node's volume. Recording its arc or interval bound therefore preserves positive volume and makes the answer true in every represented order. A new arc cannot create a cycle, since that would make the region empty. Normalization preserves this updated region. The transcript is then extended by the chosen answer.

During the subsequent repair, the temporary DAG bounds follow from Lemma E.3, and Lemma 3.4 supplies interior anchors. Each move again preserves positive volume and the transcript by Lemma E.2. This applies to every vertex of a selected path, including those whose incident arcs were removed earlier in the batch. The full batch is completed, and Lemma E.3 restores the strict bounds for the next call.

The finite certificate calculations and the explicit rounding rule terminate. Before answering, only queried labels can move. After a move, the label has an isolated coordinate with interval $(0, 1)$. Further refinements leave it isolated with the same interval, so its density remains one. A second move would therefore have gain zero, contradicting the failed-certification gain guarantee. Thus there are at most two such moves. A repair makes at most n further moves, so every call terminates with at most $n + 2$ refinements. All analytic guarantees used here apply to arbitrary positive-volume regions with the stated DAG bounds, independently of the interaction history. □

F Exact arithmetic and the stored state

The budget $E_k \le n$ bounds the number of digit assignments and their length. We verify that it also bounds their bit cost, and that the adversary's data structure remains of polynomial size

during the first $Q(n)$ comparisons. Computation is measured on a fixed deterministic multitape Turing machine, with integers represented in binary.

Write S_k for the number of digit choices, t_k for the number of sweeps, and b_k for the bound on retained vertices in (27), with the parameters chosen in Appendix D. The enumeration bound is

$$E_k = S_k^{b_k t_k}. \tag{34}$$

F.1 Precision in the interval calculation

Suppose the stored endpoints have at most Λ fractional binary digits. An update takes maxima and minima, forms an integer linear combination, and divides by the power of two S. It therefore adds at most $\log S$ fractional digits. If there are b' interior vertices, each endpoint calculation makes $b't$ updates. The two calculations share digits but use their own values, so the precision in either is at most

$$\Lambda + b't \log S \le \Lambda + \log E. \tag{35}$$

This proves the precision claim of Theorem B.1. All computed positions remain in $[0, 1]$, so their integer numerators need at most one additional bit. Digits and counts have at most $\log S$ and $\log E + 1$ bits, respectively.

F.2 Precision of new anchors

If a nondegenerate interval has dyadic endpoints with at most F fractional digits, its length is at least 2^{-F}. The rounding rule of Lemma E.1 therefore uses a grid exponent at most F, and its anchor has at most F fractional digits.

For a certified central interval, (35) gives $F \le \Lambda + \log E_k$. Forming the middle half of a stored interval gives endpoints with at most $\Lambda + 2$ digits. Since $E_k \ge 4$, both choices produce anchors with at most $\Lambda + \log E_k$ digits. This also covers the direct middle-half choice after a failed comparison certificate. The update of neighboring bounds and `Normalize` take maxima and minima of existing endpoints and the chosen anchor, adding no precision.

All initial endpoints are 0 or 1. After P refinements, induction therefore gives $\Lambda \le P \log E_k$. By Lemma 3.6, the first $Q(n)$ completed calls to `Compare` account for $O(Q(n)/k)$ refinements. An unfinished call adds at most $n + 2$ by Proposition 3.5. Thus, during these first $Q(n)$ calls,

$$P = O(Q(n)/k + n), \tag{36}$$

and consequently

$$\Lambda \le P \log E_k = O(n \log^2 n). \tag{37}$$

Temporary endpoints in the next calculation add at most another $\log E_k$ digits.

F.3 Inventory and work per call

Each refinement creates at most one node in the adversary's tree. There are thus at most $P + 1$ nodes, n item records, and at most P stored child anchors. An item belongs to exactly one node graph, so their total numbers of vertices and arcs are at most n and n^2, respectively. The recorded prefix contains at most $Q(n)$ comparisons. Labels, pointers, depths, and counters have polynomial length, as do all dyadic endpoints by (37).

For one certificate, the algorithm enumerates at most $E_k \le n$ digit assignments. Each assignment uses two endpoint calculations, each with at most $b_k t_k \le \log E_k \le \log n$ updates. The boundary vertices remain fixed. Maxima, minima, integer multiplication, and division by powers of two operate on polynomially many bits. Central certification sorts at most n lower and upper endpoints and counts the intervals inside its candidate. Rounding computes a grid

exponent and a ceiling using the same bounded-precision integers. Standard binary arithmetic therefore gives a fixed polynomial bound for each operation.

Before selecting an answer, a call makes at most two failure refinements and three calls to `Select`. Failure refinements use the middle half directly. After recording the answer, the single structural repair makes at most n refinements and hence at most n calls to `ChooseAnchor`.

The remaining work is polynomial in the stored graph size. Topological passes normalize intervals and compute directed-path lengths. A finite graph search finds each retained neighborhood; it stops when either the prescribed radius or all available vertices have been reached. Routing uses the adversary's tree, and deterministic choices can be made by scanning labels. At the horizon, topologically sorting the bounded transcript DAG produces the fixed ranks for all later answers. Thus the bit-operation bound has a fixed exponent, independent of k and of the number of queries already answered.

G Parameter reference

Table 1 collects the parameters of the adversary and its analysis. The last two columns distinguish where a parameter's role is introduced from where its value is chosen and explained. All choices are fixed before the first query. Appendix C gives explicit choices for the digit alphabet, number of sweeps, and neighborhood radius in the local enclosure theorem.

Table 1: Parameters and their roles. References lead to the explanations in the text.

Symbol	Value	Role	Introduced	Choice
k	Integer, $k \geq 64$	Repair a vertex that reaches this degree after an answer.	§3.3	§4
H	2^k	Number of vertices refined in a selected directed path.	§3.3.3	§3.3.3
B	$(k+1)^{2H+1}$	Bounds the number of other vertices comparable with one vertex in its node graph.	App. D	App. D
w_*	2^{-2kB}	Calibrates the finite calculations: a coordinate of at least this width guarantees certification.	App. D	App. D
η	$1/(100k)$	Error in the retained fraction, abbreviated in the comparison-count proof.	§4.1	§3.3.1
$Q(n)$	$n\lceil \log n \rceil$	Number of answers before switching to a fixed consistent order.	§3.4	§3.3

H Proofs of the consequences

H.1 Producing a target order

Proof of Corollary 6.1. A successful output assigns each key to one slot of R. Every total order satisfying the transcript must respect this assignment, so $T_q \leq e(R)$. For all sufficiently large n, apply Theorem 3.7 with the parameter $k = k(n)$ selected in Section 5. It gives

$$q \geq \frac{I(R)}{1+\delta_{k(n)}} = (1-o(1))I(R),$$

since $k(n) \to \infty$ and $\delta_k = O(\log k/k)$. The $o(1)$ term depends only on n, so the bound is uniform over all targets R. □

H.2 A compatible partition into layers

The reduction for an initial partial order needs an ordered partition whose layers retain many permutations. Cardinal et al.'s construction gives such a partition.

Lemma H.1 (Weak-order extension)**.** *Given a partial order P on n elements, one can deterministically construct in polynomial time a weak order W extending P such that*

$$\log e(W) \geq \log e(P) - C_0 n \log\log n$$

for an absolute constant C_0.

Proof. Cardinal et al. [14, Section 3, Lemmas 1 and 3] construct W in polynomial time with

$$I(W) \leq I(P) + 2n \log(I(P)/n + 1) + O(n).$$

Since $I(P) \leq \log(n!) \leq n \log n$, the extra term is $O(n \log\log n)$. Substitute $I(W) - I(P) = \log e(P) - \log e(W)$. For two elements, take $W = P$. □

An arbitrary partition into antichains need not admit a compatible layer order. For example, the relations $a < b$ and $c < d$ permit the antichains $\{a, d\}$ and $\{b, c\}$, but require opposite orders between them. Lemma H.1 supplies both the partition and a layer order respecting all initial relations.

Proof of Corollary 6.2. Construct W using Lemma H.1, and let its layers have sizes $m_1, \ldots, m_t$. Choose $k = k(n)$ once, using the full input size. Within every layer of size at least two, run a copy of the ordinary adversary with these parameters and its local horizon $Q(m_i) = m_i \lceil \log m_i \rceil$. A singleton needs no comparisons. Answer comparisons between layers from their fixed order.

Each local transcript has a consistent total order. Concatenating these orders gives a linear extension of W and hence of P, proving legality. At termination, each layer must have a unique possible order: two possible orders in one layer, combined with the same orders in the others, would give two global orders consistent with the full transcript.

Let q_i count comparisons within layer i. Theorem 3.7 applies to the local transcript with $T_{q_i} = 1$ and gives

$$q_i \geq \frac{\log(m_i!)}{1 + \delta_k}.$$

The inequality also holds for a singleton. Since $\sum_i m_i = n$ and $e(W) = \prod_i m_i!$, summing gives

$$\begin{aligned} q \geq \sum_i q_i &\geq \frac{\log e(W)}{1 + \delta_k} \\ &\geq \frac{\log e(P) - C_0 n \log\log n}{1 + \delta_k}. \end{aligned}$$

As $k(n) \to \infty$, this implies the stated bound, uniformly over P.

The common choice of $k(n)$ is essential to this estimate: each layer has the same δ_k, even when its size stays small. It also preserves polynomial time in n. Each finite calculation enumerates at most $E_k \leq n$ assignments. Across all layers, the horizons sum to at most $n \lceil \log n \rceil$, and the refinement and precision bounds of Section 5 apply with the ambient size n. The construction of W is polynomial-time as well. For the finitely many sizes at which no tuple qualifies, use a fixed topological order of P. □

H.3 Amplifying a constant-factor estimate

Suppose a deterministic polynomial-time algorithm estimates $V(D) > 0$ within a fixed factor $C > 1$. Let $D^{(k)}$ be the disjoint union of k copies of the order-restriction DAG D, each with the same interval bounds. Its feasible region is a Cartesian product, so

$$V(D^{(k)}) = V(D)^k.$$

A factor-C estimate for this volume, followed by a kth root, estimates $V(D)$ within a factor $C^{1/k}$. Taking $k = O(1/\varepsilon)$ makes this factor at most $1 + \varepsilon/4$. Computing the root to relative error at most $\varepsilon/4$ then gives relative error at most ε. The expanded input and the required rational arithmetic have size polynomial in the original input length and $1/\varepsilon$. Disjoint union preserves both the maximum degree and the directed-path bound, so the argument applies to each fixed-d, h class.

For linear-extension counts, if Q is the disjoint union of k copies of an n-element partial order P, then

$$e(Q) = \frac{(kn)!}{(n!)^k}\, e(P)^k.$$

Choose a linear extension within each copy and then interleave the copies, giving the displayed multinomial factor. Divide a factor-C estimate for $e(Q)$ by this explicitly computable factor before taking the root. The factorials have $O(kn \log(kn))$ bits, so the same polynomial-time conclusion holds. In particular, a deterministic PTAS, run at any fixed sufficiently small error, would already yield an FPTAS.

H.4 From approximate counting to an adversary

Suppose a deterministic FPTAS for $e(P)$ is available. For an unresolved comparison, let a, b be the positive numbers of extensions consistent with its two answers. Estimate both counts to relative error at most $\eta = 1/n$ and choose the larger estimate. If the chosen branch has count a, then $(1 + \eta)a \geq (1 - \eta)b$, and hence

$$a \geq \frac{1 - \eta}{2}(a + b).$$

Already determined comparisons lose no extensions. Starting from any supplied P, after q answers at least $e(P)((1 - \eta)/2)^q$ extensions remain. A correct sorter can terminate only when this number is at most one, so

$$q \geq \frac{\log e(P)}{1 - \log(1 - 1/n)} = (1 - O(1/n)) \log e(P).$$

The approximation calls use polynomial time, and every chosen answer is legal. This proves the claimed implication uniformly over P.

H.5 Sampling linear extensions

Proof of Corollary 6.3. All asymptotic constants in this proof may depend on the fixed parameters d, h. Give every vertex the interval $(0, 1)$, and choose S, p as in (24)–(25). We use coupling from the past [24], stopping when all coordinate orders are certified, as in Huber's sampler [23].

Fix a topological order. For each negative integer s, let F_s be a full sweep of (20), using independent uniform digits at every vertex. Retain each digit once generated. For a horizon t, apply $F_{-t}, \ldots, F_{-1}$ in that order to the all-zero and all-one vectors, obtaining $A(t), B(t)$. A larger horizon prepends older sweeps and replays the entire history, without changing any previously used digit.

Each sweep is increasing and preserves the closed feasible region. Consequently,

$$A(t) \le A(t+1) \le B(t+1) \le B(t).$$

Appendix C gives $\mathbb{E}(B_x(t) - A_x(t))^p \le 2^{-t}$ for every coordinate x. Together with nesting, this implies that both vectors converge almost surely to a common limit Z. To identify its law, take U uniformly distributed on $K(D)$, independently of the digit history. Applying the same t sweeps to U gives a uniform point $Y(t)$, since each sweep preserves the uniform law after averaging its digits (Appendix B). Monotonicity gives $A(t) \le Y(t) \le B(t)$, so $Y(t) \to Z$ almost surely. Thus Z is uniform on $K(D)$; it is used only in the proof.

Start with $t_0 = \lceil \log(n+1) \rceil$ and double the horizon until the intervals $[A_x(t), B_x(t)]$ are pairwise disjoint. Sort their lower endpoints and check that each upper endpoint is strictly below the next lower endpoint. At termination, return this order. It equals the order of Z, which is a uniform linear extension: every compatible permutation occupies a simplex of volume $1/n!$. Since Z has distinct coordinates almost surely and the widths vanish, the algorithm terminates almost surely. The stopping rule therefore introduces no bias.

For the running time, the sorted coordinates of Z have the order-statistic distribution of n independent uniform points on $(0, 1)$. Indeed, conditioning those points on their relative permutation being a linear extension leaves their sorted values unchanged. A union bound gives

$$\mathbb{P}\big(\min_{x \ne y} |Z_x - Z_y| \le 2\delta\big) \le 2n(n-1)\delta.$$

If every width is at most δ and every coordinate gap exceeds 2δ, the enclosing intervals are disjoint. Markov's inequality and a union bound therefore give

$$\mathbb{P}(\text{failure at horizon } t) \le n 2^{-t} \delta^{-p} + 2n(n-1)\delta.$$

Taking $\delta = 2^{-t/(p+1)}$ yields

$$\mathbb{P}(\text{failure at horizon } t) \le (2n^2 - n) 2^{-t/(p+1)}. \tag{38}$$

Summing this tail over doubled horizons shows that the final horizon T satisfies $\mathbb{E}T = O(\log n)$ and $\mathbb{E}T^2 = O(\log^2 n)$.

A topological order and incoming and outgoing adjacency lists take $O(n)$ operations to prepare. Each stage uses $O(nt)$ arithmetic operations, including the $O(n \log n)$ comparisons needed to sort the intervals, since $t \ge t_0 \ge \log n$. Summing over doubled horizons gives $O(nT)$ operations. Each digit uses exactly $\log S$ fair bits, and retaining the history uses only $nT \log S$ bits in total. This proves both expected bounds.

All computed endpoints are exact dyadic rationals. After t sweeps they have at most $ht \log S$ fractional bits: within one sweep a new value can depend on earlier updates only along a predecessor path of at most h vertices, and each update adds at most $\log S$ bits. Thus the expected bit cost of the arithmetic is $O(n \log^2 n)$ in a random-access model, by summing $O(nt^2)$ over the stages and using the bound on $\mathbb{E}T^2$. □

References

[1] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Second edition, Addison-Wesley, 1998. Publisher's page.

[2] 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.

[3] Mikhail J. Atallah and S. Rao Kosaraju. *An adversary-based lower bound for sorting.* Information Processing Letters 13(2), 55–57, 1981.

[4] Allan Borodin, Leonidas J. Guibas, Nancy A. Lynch, and Andrew C. Yao. *Efficient searching using partial ordering.* Information Processing Letters 12(2), 71–75, 1981. doi:10.1016/0020-0190(81)90005-3.

[5] Dana Richards and Pravin M. Vaidya. *On the distribution of comparisons in sorting algorithms.* BIT 28, 764–774, 1988. Authors' paper.

[6] Gerth Stølting Brodal, Shiva Chaudhuri, and Jaikumar Radhakrishnan. *The randomized complexity of maintaining the minimum.* Nordic Journal of Computing 3(4), 337–351, 1996. Authors' report, BRICS RS-96-40.

[7] Haim Kaplan, Or Zamir, and Uri Zwick. *The amortized cost of finding the minimum.* Proceedings of the Twenty-Sixth Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), 757–768, 2015. Published online 22 December 2014. doi:10.1137/1.9781611973730.51. Authors' paper.

[8] Haim Kaplan, Or Zamir, and Uri Zwick. *A sort of an adversary.* Proceedings of the Thirtieth Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), 1291–1310, 2019. doi:10.1137/1.9781611975482.79.

[9] Richard P. Stanley. *Two poset polytopes.* Discrete & Computational Geometry 1, 9–23, 1986. doi:10.1007/BF02187680. Author's copy.

[10] Graham Brightwell and Peter Winkler. *Counting linear extensions.* Order 8(3), 225–242, 1991. doi:10.1007/BF00383444.

[11] Martin Dyer, Alan Frieze, and Ravi Kannan. *A random polynomial-time algorithm for approximating the volume of convex bodies.* Journal of the ACM 38(1), 1–17, 1991. doi:10.1145/102782.102783. Authors' manuscript.

[12] Jeff Kahn and Jeong Han Kim. *Entropy and sorting.* Journal of Computer and System Sciences 51(3), 390–399, 1995. doi:10.1006/jcss.1995.1077.

[13] András Prékopa. *On logarithmic concave measures and functions.* Acta Scientiarum Mathematicarum 34, 335–343, 1973. Publisher's paper.

[14] Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M. Jungers, and J. Ian Munro. *An efficient algorithm for partial order production.* SIAM Journal on Computing 39(7), 2927–2940, 2010. doi:10.1137/090759860. Authors' manuscript.

[15] Ivor van der Hoog, Irina Kostitsyna, Maarten Löffler, and Bettina Speckmann. *Preprocessing ambiguous imprecise points.* 35th International Symposium on Computational Geometry (SoCG), LIPIcs 129, 42:1–42:16, 2019. doi:10.4230/LIPIcs.SoCG.2019.42.

[16] Ivor van der Hoog, Irina Kostitsyna, Maarten Löffler, and Bettina Speckmann. *Sorting under partial (interval order) information.* Journal of Computational Geometry 15(1), 143–171, 2024. doi:10.20382/jocg.v15i1a6.

[17] Federico Ardila, Thomas Bliem, and Dido Salazar. *Gelfand–Tsetlin polytopes and Feigin–Fourier–Littelmann–Vinberg polytopes as marked poset polytopes.* Journal of Combinatorial Theory, Series A 118(8), 2454–2462, 2011. doi:10.1016/j.jcta.2011.06.004. arXiv:1008.2365.

[18] Ferenc Bencs and Guus Regts. *Approximating the volume of a truncated relaxation of the independence polytope.* Discrete & Computational Geometry 76, 508–525, 2026. doi:10.1007/s00454-026-00824-y.

[19] Swee Hong Chan and Igor Pak. *Linear extensions of finite posets.* EMS Surveys in Mathematical Sciences, published online 17 March 2025. doi:10.4171/EMSS/97. Authors' manuscript.

[20] Topi Talvitie, Kustaa Kangas, Teppo Niinimäki, and Mikko Koivisto. *A scalable scheme for counting linear extensions.* Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI), 5119–5125, 2018. Publisher's page. Authors' paper.

[21] Bernhard Haeupler, Richard Hladík, John Iacono, Václav Rozhoň, Robert E. Tarjan, and Jakub Tětek. *Fast and simple sorting using partial information.* Proceedings of the Thirty-Sixth Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), 2025. doi:10.1137/1.9781611978322.134. Authors' manuscript.

[22] Daniel Rutschmann. *Sorting under partial information with optimal preprocessing time via unified bound heaps.* Preprint, 2026. To appear at FOCS 2026. arXiv:2604.12653.

[23] Mark Huber. *Near-linear time simulation of linear extensions of a height-2 poset with bounded interaction.* Chicago Journal of Theoretical Computer Science 2014, Article 3, 1–16. doi:10.4086/cjtcs.2014.003.

[24] James G. Propp and David B. Wilson. *Exact sampling with coupled Markov chains and applications to statistical mechanics.* Random Structures & Algorithms 9(1–2), 223–252, 1996. Authors' paper.