Tight Better-Than-Worst-Case Bounds for Element Distinctness and Set Intersection

Ivor van der Hoog
IT University of Copenhagen, Denmark

Eva Rotenberg
IT University of Copenhagen, Denmark

Daniel Rutschmann
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Abstract

The *element distinctness* problem takes as input a list I of n values from a totally ordered universe, where pairwise comparisons between values are allowed, and the goal is to decide whether I contains any duplicates. It is a well-studied problem with a classical worst-case $\Omega(n \log n)$ comparison-based lower bound by Fredman [TCS'76]. At first glance, this lower bound appears to rule out any algorithm more efficient than the naive approach of sorting I and comparing adjacent elements. However, upon closer inspection, the $\Omega(n \log n)$ bound is overly pessimistic. For instance, if I contains $n/2$ identical elements, a median-finding algorithm will, regardless of the input order, find a duplicate in linear time. This raises a natural question: *Are there comparison-based lower bounds for element distinctness that are sensitive to the amount of duplicates in the input instance?*

To address this question, we derive instance-specific lower bounds. For any input instance I, we represent the combinatorial structure of the duplicates in I by an undirected graph $G(I)$ that connects identical elements. Each such graph G is a union of cliques, and we study algorithms by their worst-case running time over all inputs I' with $G(I') \cong G$. We establish an adversarial lower bound showing that, for any deterministic algorithm $\mathcal{A}$, there exists a graph G and an algorithm $\mathcal{A}'$ that, for all inputs I with $G(I) \cong G$, is a factor $O(\log \log n)$ faster than $\mathcal{A}$. Consequently, no deterministic algorithm can be $o(\log \log n)$-competitive for all graphs G. We complement this with an $O(\log \log n)$-competitive deterministic algorithm, thereby obtaining tight bounds for element distinctness that go beyond classical worst-case analysis. Subsequently, we study the related problem of *set intersection*. We show that no deterministic set intersection algorithm can be $o(\log n)$-competitive, and provide an $O(\log n)$-competitive deterministic algorithm. We find it interesting and surprising to discover tight $O(\log \log n)$-competitive bounds for element distinctness. Moreover, we find the separation between element distinctness and the set intersection problem unexpected.

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

Keywords and phrases Comparison-based analysis, set intersection, universal optimality

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.101

Related Version *Full Version*: https://arxiv.org/abs/2511.02954

Funding *Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann* thank the VILLUM Foundation grant (VIL37507) "Efficient Recomputations for Changeful Problems" for supporting this work. *Daniel Rutschmann* is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (No. 101019564).

1 Introduction

The *element distinctness* problem takes as input a list I of n values. The goal is to decide whether I contains any duplicates using pairwise comparisons. In the closely related problem of *set intersection*, the input consists of two lists A and B, each containing n values, and the task is to determine whether there exist $(a, b) \in A \times B$ such that $a = b$.

The algorithmic complexity of element distinctness depends on the operations permitted. The most general model for this problem is the *comparison-based* model, in which one may perform pairwise comparisons between elements of I, but cannot otherwise manipulate its values. In this model, the worst-case complexity depends on the type of comparisons. For instance, if a comparison only reveals whether $a = b$ or $a \neq b$, then element distinctness has a trivial $\Theta(n^2)$ worst-case lower bound. The canonical comparison-based model assumes that the input I comes from a totally ordered set. That is, for any $a, b \in I$, either they are duplicates ($a = b$), or they satisfy $a < b$ or $b < a$. In this model, Fredman [8, TCS'76] proved an $\Omega(n \log n)$ lower bound by showing that any correct algorithm can only output YES, DISTINCT if it can certify that no duplicates exist. Such a certificate defines a total order on I. Since there are $n!$ distinct total orders, any comparison-based algorithm requires $\Omega(n \log n)$ comparisons. This bound is tight: sorting I and checking adjacent elements suffices to decide distinctness. However, later work aimed to obtain more refined bounds. Ajtai [2, Combinatorica'88] extended the analysis by incorporating space usage. Let S denote the number of available RAM cells. Ajtai proved a comparison-based lower bound of $\Omega(f(n)/S)$, where $f(n)$ is superlinear. Beame, Borodin, Saks, and Skyum [3, STOC'89] refined this to $\Omega(n^{3/2}/\sqrt{S})$. Independently, Yao [28, FOCS'88] obtained a comparison-based lower bound, conjectured to be near-optimal, of $\Omega\big(n^{2-1/\sqrt{\log n}}/S\big)$. Patt-Shamir and Peleg [21, TCS'93] show an $\Omega(n^{3/2}/\sqrt{S})$ lower bound for set intersection. Further bounds are known for the *k-set intersection* problem, where the number of sets k is part of the input. Pătraşcu [22, STOC'10] reduced 3SUM to k-set intersection, yielding a conditional near-quadratic lower bound. Kopelowitz, Pettie, and Porat [18, SODA'16] strengthened this result by excluding strongly subquadratic algorithms. Chakrabarti, Cormode, and McGregor [6, STOC'08] established lower bounds for k-set intersection in the communication and streaming models, showing that $\Omega(n)$ bits of space are required for single-pass streaming algorithms.

Given the $\Omega(n \log n)$ comparison-based lower bound, faster algorithms exist only in less general models of computation. Ostlin and Pagh [20, STOC'03] showed that element distinctness can be solved in expected linear time and space when hashing is allowed. Beame, Clifford, and Machmouchi [4, FOCS'13] revisited the space–time trade-off introduced by Ajtai [2] and showed that, with access to hashing and integer arithmetic, one can achieve a randomized trade-off of $O(n^{3/2}/\sqrt{S})$. Other improvements rely on models that permit faster sorting. Assuming that I consists of integers and that the underlying word-RAM supports constant-time integer arithmetic, Han [12, STOC'02] gave a deterministic $O(n \log\log n)$ sorting algorithm, which directly yields an $O(n \log\log n)$ algorithm for element distinctness. Han and Thorup [13] further obtained a randomized $O(n\sqrt{\log\log n})$ algorithm for the same problem. Finally, Pătraşcu and Demaine [22, SICOMP'06] proved an $\Omega(\log n)$ lower bound for dynamic element distinctness and dynamic set intersection in the cell-probe model.

Problem statement. Fredman's $\Omega(n \log n)$ lower bound appears to rule out faster comparison-based algorithms. Subsequent works therefore prove lower bounds in other settings or in more specialized computational models. However, upon closer examination, the $\Omega(n \log n)$ bound is overly coarse. Consider an input instance I in which all elements are identical. In this case, any algorithm can correctly terminate after a single comparison. Consider the next example, where half of the elements in I are identical and the remaining half are distinct. Here, an adversarial argument yields an $\Omega(n)$ lower bound: any correct algorithm can only output NOT DISTINCT after it has identified a duplicate. This duplicate serves as a *witness*, and an adversary can force it to process all $n/2$ distinct elements before encountering any duplicate. This lower bound can be matched by the (deterministic) algorithm that finds the median of I, regardless of the order the input is presented in. These observations naturally raise the question: *Are there lower bounds for element distinctness that are sensitive to the*

amount of duplicates in the input instance? To answer this question, we examine frameworks that refine worst-case analysis by considering algorithmic performance under input-specific parameters. We explore a variety of such frameworks and provide tight bounds.

Better-than-worst-case optimality. There are many notions for better-than-worst-case complexity, and a long publication history. All these approaches share a common theme: if the worst-case running time of an algorithm is defined as the maximum over all possible input instances, can we instead derive lower bounds from the *input instance*?

The strictest interpretation of this idea is *(true) instance optimality*. An algorithm $\mathcal{A}$ is *instance-optimal* if there exists a constant c such that, for all sufficiently large input instances I and for all algorithms $\mathcal{A}'$, we have Running-Time$(\mathcal{A}, I) \leq c \cdot$Running-Time$(\mathcal{A}', I)$. Perhaps the most famous instance-optimality conjecture is due to Sleator and Tarjan [23], who proposed that there exists a constant c and a BST algorithm $\mathcal{A}$ such that, for every sufficiently long sequence I of predecessor queries, the total running time Running-Time$(\mathcal{A}, I)$ of answering all queries in I satisfies Running-Time$(\mathcal{A}, I) \leq c \cdot$ Running-Time$(\mathcal{A}', I)$ for every other BST algorithm $\mathcal{A}'$. This conjecture remains open, but Demaine et al. [7, FOCS'04] showed a BST algorithm $\mathcal{A}$ that is $O(\log \log n)$-instance-competitive: Formally, they showed that, for any sufficiently long sequence I of BST queries and every other BST $\mathcal{A}'$, Running-Time$(\mathcal{A}, I) \leq O(\log \log n) \cdot$ Running-Time$(\mathcal{A}', I)$.

True instance optimality is provably unattainable. Consider the family of quadratic-time algorithms $\{\mathcal{A}_{i,j}\}$ that perform all pairwise comparisons of elements in I in arbitrary order, where $\mathcal{A}_{i,j}$ starts by comparing $I[i]$ and $I[j]$. For any algorithm $\mathcal{A}$ and constant c, there exists an input I and pair (i, j) such that $I[i] = I[j]$ is the only duplicate and, after c comparisons, $\mathcal{A}$ has not yet compared $I[i]$ and $I[j]$. However, $\mathcal{A}_{i,j}$ finds the duplicate after one comparison. In fact, this construction shows that no $o(n \log n)$-instance-competitive algorithm can exist: for any algorithm, there exists an input I and pair (i, j) such that $I[i] = I[j]$ is the only duplicate, and after $\log(n!) - 1$ comparisons, $\mathcal{A}$ cannot certify whether $I[i] < I[j]$, $I[i] > I[j]$, or $I[i] = I[j]$. Yet, $\mathcal{A}_{i,j}$ terminates after one comparison. This calls for a notion of optimality that is more fine-grained than worst-case, but coarser than instance optimality.

Output-optimality. Fix some algorithmic problem and a value k, and denote by $\mathbb{I}_{n,k}$ all inputs of size n that have an output of size k. For a fixed algorithm $\mathcal{A}$ and integer pair (n, k), we define the output-sensitive running time as Output$(\mathcal{A}, n, k) := \max_{I \in \mathbb{I}_{n,k}}$ Running-Time$(\mathcal{A}, I)$. We say that an algorithm $\mathcal{A}$ is *output-optimal* if there exists a constant c such that, for all k and sufficiently large n, we have, for all $\mathcal{A}'$, Output$(\mathcal{A}, n, k) \leq c \cdot$Output$(\mathcal{A}', n, k)$. A classical result is the $O(n \log k)$-time convex hull algorithm by Kirkpatrick and Seidel [17]. A related but stronger notion is *order-oblivious optimality*, a term proposed by Afshani, Barbay, and Chan [1, FOCS'09]. Let $\Pi(I)$ denote all permutations of I. For an algorithm $\mathcal{A}$ and input I, we define the order-oblivious running time as Order$(\mathcal{A}, I) := \max_{I' \in \Pi(I)}$ Running-Time$(\mathcal{A}, I')$.

An algorithm $\mathcal{A}$ is *order-oblivious optimal* if for a constant c, for all sufficiently large I, Order$(\mathcal{A}, I) \leq c \cdot$ Order$(\mathcal{A}', I)$, for all algorithms $\mathcal{A}'$. [1] proved that the algorithm in [17] is optimal in this model. Note that this notion of optimality implies output-optimality.

Universal optimality. In recent years, *universal optimality* has been proposed as an umbrella concept that captures many notions of better-than-worst-case analysis. Haeupler, Wajc, and Zuzic [11, STOC'21] introduced this concept in a distributed setting, observing that the running time of their algorithm $\mathcal{A}$ depends on two aspects of the input: the graph topology G of the distributed system and the edge weights w. The worst-case running time then is:

$$\text{Worst-Case}(\mathcal{A}, n) := \max_{n\text{-vertex graphs } G} \max_{\text{weightings } w \text{ of } G} \text{Running-Time}(\mathcal{A}, G, w).$$

They argued that this double maximum provides an overly pessimistic view of performance and proposed a more refined analysis. Rather than maximizing over both parameters, they fix the first variable and define the *universal running time* as the maximum over the second:

$$\text{For a fixed } n\text{-vertex graph } G, \quad \text{Universal}(\mathcal{A}, G) := \max_{\text{weightings } w \text{ of } G} \text{Running-Time}(\mathcal{A}, G, w).$$

For fixed G, the *universal lower bound* is the minimum universal running time. We call $\mathcal{A}$ *universally optimal* if, for all sufficiently large G, it asymptotically matches this bound.

A universal framework. Haeupler, Hladík, Rozhoň, Tarjan, and Tětek [9, FOCS'24] extended this notion into a broader framework. If running time depends on X and Y, such that:

$$\text{Worst-case}(\mathcal{A}, n) = \max_{\text{variable } X \text{ of size } n} \max_{\text{inputs } Y \text{ derived from } X} \text{Running-Time}(\mathcal{A}, X, Y),$$

then we can define the universal running time by fixing X, and considering the worst-case Y:

$$\text{For a fixed input } X, \quad \text{Universal}(\mathcal{A}, X) := \max_{\text{inputs } Y \text{ derived from } X} \text{Running-Time}(\mathcal{A}, X, Y).$$

An algorithm $\mathcal{A}$ is *universally optimal* if there exists a constant c such that, for all sufficiently large X and all algorithms $\mathcal{A}'$, $\text{Universal}(\mathcal{A}, X) \leq c \cdot \text{Universal}(\mathcal{A}', X)$. Universal optimality is challenging to obtain, since the algorithm $\mathcal{A}$ must be efficient for all X, but competes against algorithms $\mathcal{A}'$ that can be uniquely designed for X. An algorithm $\mathcal{A}$ is $f(n)$-*competitive* if, for sufficiently large n, all X of size n, and all $\mathcal{A}'$, we have $\text{Universal}(\mathcal{A}, X) \leq f(n) \cdot \text{Universal}(\mathcal{A}', X)$. We say that an algorithm is $O(f(n))$-*competitive* if it is $g(n)$-competitive for some $g \in O(f)$. A universally optimal algorithm is $O(1)$-competitive.

Haeupler, Hladík, Rozhoň, Tarjan, and Tětek [9, FOCS'24] studied the single-source shortest path (SSSP) problem. They observed that the running time of an SSSP algorithm $\mathcal{A}$ depends on two aspects of the input: the underlying directed graph G and its edge weights w. Consequently, the worst-case running time can be expressed as a double maximum over both the graph structure and the weighting. The universal running time is then:

$$\text{For a fixed } G, \quad \text{Universal}(\mathcal{A}, G) := \max_{\text{weightings } w \text{ of } G} \text{Running-Time}(\mathcal{A}, G, w).$$

There is a long line of research on universally competitive algorithms for sorting under partial information. Here, the input is a partial order P (a directed, acyclic graph), and the goal is to determine a hidden underlying linear extension L of P using the fewest possible comparisons. The running time of a partial-sorting algorithm $\mathcal{A}$ depends on both the input poset P and the linear extension L. Thus, the universal running time of an algorithm can be defined as

$$\text{For a fixed DAG } P, \quad \text{Universal}(\mathcal{A}, P) := \max_{\text{linear extensions } L \text{ of } P} \text{Running-Time}(\mathcal{A}, P, L).$$

Better-than-worst-case algorithms predate the term *universal optimality*, and were proposed by Kahn and Saks [16, Order'84]. There exist increasingly competitive algorithms by Kahn and Kim [15, STOC'92] and Cardinal et al [5, STOC'10] until Haeupler et al. [10, SODA'25] (and [27, FOCS'24]) gave $O(1)$-competitive algorithms.

Order-oblivious optimality [1, 27, 29, 11, 9, 10, 14, 26, 25, 24] also fits into this framework, since running times depend on P and on the input order I_P. The resulting universal running time, defined by fixing P and taking the worst-case running time over all orders I_P of P, is thereby equivalent to the order-oblivious running time as defined in [1].

1.1 Better-than-worst-case element distinctness

We study better-than-worst-case algorithms for element distinctness in the comparison-based model of Fredman [8] and Yao [28]. We apply the universal optimality framework to derive lower bounds when the input contains duplicates. Universal optimality fixes the combinatorial structure of the input. For element distinctness, we identify two natural ways to capture this structure. Our first proposal is to describe it through the *duplicates* in I. For any instance I, let $G(I)$ be the labeled graph with vertex set $\{1, \ldots, n\}$ where the edge $\{i, j\}$ exists if and only if $I[i] = I[j]$. Thus, $G(I)$ is the union of disjoint cliques. We can fix a graph topology G and consider the worst-case running time on inputs with the same topology. Formally, let $\mathbb{G}_n$ denote the family of all n-vertex graphs that are unions of disjoint cliques, and let $\cong$ denote graph isomorphism. Then $\text{Worst-Case}(\mathcal{A}, n) = \max_{G \in \mathbb{G}_n} \max_{I: G(I) \cong G} \text{Running-Time}(\mathcal{A}, I)$. And,

$$\text{for fixed } G \in \mathbb{G}_n, \quad \text{Universal}(\mathcal{A}, G) = \max_{I: G(I) \cong G} \text{Running-Time}(\mathcal{A}, I).$$

This captures the earlier intuition. If I contains only one value, n times, then $G(I)$ is a single clique and the universal lower bound for this G is constant. If I contains $n/2$ identical elements and $n/2$ distinct ones, then $G(I)$ consists of a clique of size $n/2$ and $n/2$ isolated vertices, and the universal lower bound for such G is $\Omega(n)$. The universal lower bound for $G \in \mathbb{G}_n$ can take any value in $[1, n \log n]$. We say that an algorithm is $O(f(n))$-universally competitive if there exists a function $g \in O(f)$ such that, for all sufficiently large n, for all graphs $G \in \mathbb{G}_n$, and for all algorithms $\mathcal{A}'$, we have $\text{Universal}(\mathcal{A}, G) \leq g(n) \cdot \text{Universal}(\mathcal{A}', G)$.

Our second proposal is to apply universal optimality equivalent to *order-oblivious optimality*. For any input I, we consider the worst-case running time of an algorithm over all permutations I' of I. Equivalently, this can be expressed in terms of graph topology: every input I induces a directed graph $\overrightarrow{G}(I)$ whose vertices correspond to the elements of I, and where there is an arc from $I[i]$ to $I[j]$ if and only if $I[i] < I[j]$. For all inputs I of size n, the undirected complement of $\overrightarrow{G}(I)$ lies in $\mathbb{G}_n$. Let $\overrightarrow{\mathbb{G}}_n$ denote all such directed graphs. Then $\text{Worst-Case}(\mathcal{A}, n) = \max_{\overrightarrow{G} \in \overrightarrow{\mathbb{G}}_n} \max_{I: \overrightarrow{G}(I) \cong \overrightarrow{G}} \text{Running-Time}(\mathcal{A}, I)$, and

$$\text{for fixed } \overrightarrow{G} \in \overrightarrow{\mathbb{G}}_n, \quad \text{Order}(\mathcal{A}, \overrightarrow{G}) = \max_{I: \overrightarrow{G}(I) \cong \overrightarrow{G}} \text{Running-Time}(\mathcal{A}, I).$$

To distinguish between our two models, we say that an algorithm is $O(f(n))$-*order-competitive* if there exists a function $g \in O(f)$ such that, for all sufficiently large n, for all $\overrightarrow{G} \in \overrightarrow{\mathbb{G}}_n$, and all algorithms $\mathcal{A}'$, $\text{Order}(\mathcal{A}, \overrightarrow{G}) \leq g(n) \cdot \text{Order}(\mathcal{A}', \overrightarrow{G})$.

Contributions. Having established these natural and well-studied notions of better-than-worst-case optimality, we now present our results. Our main body is dedicated to an adaptive adversarial lower bound showing that no deterministic $o(\log \log n)$-competitive algorithm exists. We then match this bound with an $\Theta(\log \log n)$-competitive algorithm. Similarly, we show an adversarial lower bound proving that no deterministic $o(\log n)$-order-competitive algorithm exists, and we match this bound by giving an $\Theta(\log n)$-order-competitive algorithm. We find these results compelling and surprising: it is rare to find $\Omega(\log \log n)$ barriers, and surprising that there is a separation between our competitiveness and order-competitiveness.

While our $\Theta(\log \log n)$-competitive algorithm is fairly simple, the main difficulty lies in showing that no other algorithm can do much better. Our algorithm's running time is a complicated function, and the lower bound has to match that, up to a $\Theta(\log \log n)$-factor. The core of our lower bound consists of Claims 9 and 12, which require considerable finesse.

The appendix further extends our approach in two directions. First, we consider a variant of universal optimality that was proposed by Cardinal et al. [5, STOC'10] who consider the following question: "If one is allowed to preprocess the universally fixed graph G, can one then design competitive algorithms?" We show that, for any graph $G \in \mathbb{G}_n$, one can preprocess G in $O(m)$ time (m is the number of cliques in G) to obtain an $O(1)$-competitive algorithm. Second, we study the related *set intersection* problem. Here we obtain a strict separation: for set intersection, there is no $o(\log n)$-competitive algorithm, regardless of whether the fixed quantity is the undirected duplicate graph or the induced DAG. We conclude with a simple $O(\log n)$-competitive algorithm for these settings, yielding a detailed classification of the difficulty of element distinctness and set intersection under various inputs.

Our paper presents a comprehensive and extensive study of the complexity of the element distinctness and set intersection problems for when the input contains duplicates. Our bounds cover a variety of popular models for better-than-worst-case analysis and are tight.

2 Preliminaries

We study the *element distinctness* and *set intersection* problems in a comparison model of computation. For element distinctness, the input is a list I of n elements from some totally ordered set U. Comparing two elements x and y returns one of "$x < y$", "$x = y$", or "$x > y$", and this is the only operation permitted on the elements of I. We output any pair of equal elements, or report that no such pair exists. In the set intersection problem the input consists of two lists A and B, each with n elements from the ordered universe U. We output a pair of elements $a \in A$ and $b \in B$ such that $a = b$, or report that no such pair exists.

This output model is equivalent to the decision version of element distinctness. Indeed, suppose that an algorithm $\mathcal{A}$ outputs NOT DISTINCT before it has identified a pair of equal elements. Then, the adversary can change the input and make all elements distinct while keeping the relative order consistent with the comparisons performed. On the modified input, $\mathcal{A}$ outputs NOT DISTINCT, so $\mathcal{A}$ is not a correct algorithm.

Algorithms and runtime. Our lower bounds count the number of comparisons required before the algorithm can find a witness. Naturally, the number of comparisons performed provides a lower bound on the total running time. Our upper bound algorithms require additional operations such as reading the input. However, if the input I is provided as a stream, the running time of our algorithms is dominated by the number of comparisons they perform. Accordingly, all our bounds can be expressed in terms of algorithmic running time. We assume that algorithms are *deterministic*. Because the output size is constant, randomized algorithms may behave very differently. Consider the example where I consists of $\frac{n}{2}$ copies of the same value and $\frac{n}{2}$ distinct values. An adversarial argument gives a deterministic lower bound of $\Omega(n)$ comparisons, as the algorithm can be forced to inspect all distinct elements first. However, an algorithm that randomly selects (i, j) and compares $I[i]$ and $I[j]$ terminates in expected constant time.

Defining cluster graphs. An instance I of element distinctness induces an undirected graph $G(I)$ on n vertices. The vertex set is $[n] = \{1, \ldots, n\}$, and for $x, y \in [n]$, there is an edge xy if $I[x] = I[y]$. Similarly, an instance (A, B) of set intersection induces an undirected graph $G(A, B)$ on $2n$ vertices. All connected components of $G(I)$ and $G(A, B)$ are cliques which we dub the *clusters* of I, or of (A, B). The *size* of a cluster is its number of vertices. We

Algorithm 1 Our algorithms `Block Sorting` and `Median Recursion` for element distinctness.

(a) `Block Sorting`

Require: Input I, integer $k \geq 1$

while $|I| \geq 2k$ **do**
  $S \leftarrow k$ arbitrary elements from I
  Sort S
  if S contains a duplicate **then**
    return it
  end if
  $I \leftarrow I - S$
end while
Sort I
if I contains a duplicate **then**
  return it
end if
Give up and **terminate**.

(b) `Median Recursion`

Require: Input I, integer $L \geq 1$

if $|I| \geq L$ **then**
  $m \leftarrow \text{median}(I)$
  Check if m occurs multiple times
  if m occurs multiple times **then**
    return m
  end if
  m-recursion$(\{x \in I : x < m\}, L)$
  m-recursion$(\{x \in I : x > m\}, L)$
end if
Give up and **terminate**.

denote by $\mathbb{G}_n$ the set of all such n-vertex graphs. For a fixed $G \in \mathbb{G}_n$ and integer L, we define the *cumulative small size* $C(L)$ as the *total size* of all clusters of size smaller than L, and the *distinct dense clusters* $D(L)$ as the *number* of clusters of size at least L.

3 Algorithms for element distinctness

We present two comparison-based procedures for element distinctness, which we run in parallel, terminating as soon as either procedure completes. The first, `Block Sorting`, operates on fixed-size batches of elements. Given a parameter k, it repeatedly sorts a block of k elements and halts if a duplicate appears within the block; otherwise it discards the block and continues (Algorithm 1 (a)). This procedure is vaguely reminiscent of the Misra-Gries algorithm [19]. The second, `Median Recursion`, uses a parameter L. Similar to deterministic quick sort, it recursively partitions the input at the median, but it prunes any recursive branch whose subproblem has fewer than L elements (Algorithm 1(b)).

Intuitively, `Median Recursion` is preferable when the universal running time lies between $O(n)$ and $O(n \log n)$, whereas `Block Sorting` can terminate sooner on instances where the universal running time is sublinear. To analyze the universal running time of these algorithms, we fix a graph G and an arbitrary value $L \geq 2$. Recall that graph G induces two quantities: $C(L)$ is the *total size* of clusters of size less than L, and $D(L)$ is the *number* of clusters of size at least L. Let I be an input with $G(I) \cong G$. We now fix an arbitrary L and run `Median Recursion` on I with this parameter. In addition, suppose $D(L)$ is known. We run `Block Sorting` on the same input with $k = 2D(L)$. With this choice, the running times of both algorithms can be expressed in terms of n, $C(L)$, and $D(L)$. If G were known in advance, we could choose L to minimize the overall running time. We show that, even without knowledge of G, it suffices to try $O(\log \log n)$ values of k and L in parallel to be $O(\log \log n)$-competitive.

Block Sorting. We analyze the universal running time of Algorithm 1 (a).

▶ **Lemma 1.** *Fix $G \in \mathbb{G}_n$ and fix some $L \geq 2$ with $C(L) < n/2$. `Block Sorting` with $k \geq 2D(L)$ and $k \in O(D(L))$ finds a duplicate after $O\big((C(L) + D(L)) \max\{1, \log D(L)\}\big)$ comparisons.*

Proof. The algorithm terminates only by returning a duplicate or by giving up. We show that, for $k \geq 2D(L)$ and $C(L) < n/2$, it always returns a duplicate.

Assume for contradiction that the algorithm reaches iteration $1 + \lceil C(L)/D(L) \rceil$ without finding a duplicate. In iteration i, let S_i be the set of elements inspected by the algorithm. Then $|S_i| = k \geq 2D(L)$. Since no duplicate is found within S_i, all elements of S_i belong to distinct clusters. Since there are only $D(L)$ clusters of size at least L, at least $D(L)$ elements of S_i lie in clusters of size less than L. Over $1 + \lceil C(L)/D(L) \rceil$ iterations, this accounts for more than $C(L)$ distinct elements lying in clusters of size less than L, a contradiction. Hence, a duplicate is found within at most $1 + \lceil C(L)/D(L) \rceil$ iterations.

Each iteration sorts k elements, requiring $O(k \log k)$ comparisons. With at most $1 + \lceil C(L)/D(L) \rceil$ iterations and the fact that $k \in O(D(L))$, the lemma follows. ◀

Median Recursion. We next analyze the universal running time of Algorithm 1(b).

▶ **Lemma 2.** *Fix $G \in \mathbb{G}$ and some $L \geq 2$. Suppose that $C(L) < n$. `Median Recursion` finds a duplicate after $O\big(n + C(L) \max\{1, \log \frac{C(L)}{L}\}\big)$ comparisons.*

Proof. Algorithm 1 (b) partitions an input I' by comparing against the median. Either the algorithm finds an element in I' that equals the median (and the algorithm terminates), or every cluster in I' remains intact after the partition. We refer to a recursive call on I' with $|I'| < L$ as a *small call*. In a small call, all elements of I' belong to clusters of size $< L$. Distinct small calls are disjoint, and each small call cannot contain any cluster of size $\geq L$, so the total input size across all small calls is at most $C(L)$. Moreover, since we partition at the median, there can be at most $O(C(L)/L)$ small calls.

Suppose that k small calls have terminated before our algorithm terminates. Our recursion is depth-first and does not further recurse after each small call. This creates a recursion tree, where the leaves are small calls. By the time our algorithm terminates, this tree has created k leaves and consists of two parts: the bottom part is a balanced binary subtree T of the recursion tree of height $O(\log k)$. The top part is a path P that connects the root of T to the root of the whole recursion tree. Each call to the `Median Recursion` algorithm performs $O(|I'|)$ comparisons on its subproblem I'. Thus, the total number of comparisons done by calls in P is $O(n)$. The work caused by calls in T is bounded by the total input size of each level of T, multiplied by the number of levels. Each leaf in T corresponds to a call to our algorithm with clusters of size less than L. Thus, the total input size across all leaves of T (and therefore all levels of T) is at most $C(L)$. Since $k \in O(C(L)/L)$, it follows that there are at most $O\big(C(L) \max\{1, \log \frac{C(L)}{L}\}\big)$ comparisons done across all calls in T, which implies the claimed bound. Since $C(L) < n$, at least one cluster has size $\geq L$, so the algorithm eventually exposes a duplicate and terminates. ◀

3.1 Clairvoyant and Oblivious Algorithms

If the universal graph G were known, then for any L, we can compute $C(L)$ and $D(L)$ and select L to minimize the running time. This yields a *clairvoyant* algorithm that achieves the better of both bounds, which we will later show to be universally optimal (Section 4.2).

▶ **Theorem 3.** *For every G, there is an element distinctness algorithm $\mathcal{A}_G$ with* $\text{Universal}(\mathcal{A}_G, G) \in$

$$O\Big(\min\Big(\min_{\substack{L \geq 2 \\ C(L) < n}} (n + C(L) \log \frac{C(L)}{L}),\ \min_{\substack{L \geq 1 \\ C(L) < \frac{n}{2}}} (C(L) + D(L)) \max\{1, \log D(L)\}\Big)\Big).$$

Next, we design an algorithm that is oblivious to G. It runs multiple instances of `Median Recursion` and `Block Sorting` on the same input I, each with a different parameter choice, and stops when any instance terminates with a witness for `yes`. We prove that, for any input I with induced graph $G(I)$, there exists an instance whose running time asymptotically matches the running time of the clairvoyant algorithm for the graph $G \cong G(I)$. We run $O(\log\log n)$ instances in parallel. Thus a sequential simulation of this parallel algorithm incurs at most an $O(\log\log n)$ multiplicative overhead over the clairvoyant algorithm.

▶ **Theorem 4.** *There is an algorithm $\mathcal{A}$ for element distinctness with* $\text{Universal}(\mathcal{A}, G) \in$

$$O\Big(\log\log(n)\cdot\min\Big(\min_{\substack{L\geq 2\\ C(L)<n}}\big(n+C(L)\log\frac{C(L)}{L}\big),\ \min_{\substack{L\geq 1\\ C(L)<\frac{n}{2}}}(C(L)+D(L))\max\{1,\log D(L)\}\Big)\Big).$$

We prove the theorem via two claims, on `Block Sorting` and `Median Recursion`:

▷ Claim 5. There exists an algorithm $\mathcal{A}$ for element distinctness with

$$\text{Universal}(\mathcal{A}, G) \in O\Big(\log\log(n)\cdot\min_{\substack{L\geq 1\\ C(L)<\frac{n}{2}}}(C(L)+D(L))\max\{1,\log D(L)\}\Big).$$

Proof. Let I be the input. Then $\mathcal{A}$ runs the following $O(\log\log n)$ algorithms in parallel:

- for $i = 0, \ldots, \lceil\log\log n\rceil$, a `Block Sorting` instance with $k = 2\cdot 2^{2^i}$; and
- a doubling scheme that starts with $\ell = 2$, checks $\min\{n, \ell\}$ arbitrary elements $I' \subseteq I$ for a duplicate via sorting I' (sampling with replacement), and then doubles ℓ.

Let $L_0 = \arg\min_{L\geq 1}^{C(L)<n/2}\big(C(L)+D(L)\big)\max\{1,\log D(L)\}$. We show that some parallel branch halts within $O\big((C(L_0)+D(L_0))\max\{1,\log D(L_0)\}\big)$ comparisons.

If $C(L_0) \geq D(L_0)^2$, pick $i = \lceil\log\log D(L_0)\rceil$. Then $k = 2\cdot 2^{2^i}$ satisfies $2D(L_0) = k \leq 2D(L_0)^2 \leq 2C(L_0)$ and $k \in O(D(L))$. We apply Lemma 1 to obtain a duplicate in $O\big((C(L_0)+k)\log k\big) \subseteq O\big(C(L_0)\max\{1,\log D(L_0)\}\big)$ comparisons. Otherwise $C(L_0) \leq D(L_0)^2$. When the doubling scheme reaches $\ell = 2^{\lceil\log(C(L_0)+D(L_0)+1)\rceil}$, then $\ell > C(L_0)+D(L_0)$, so by the pigeonhole principle, any subset $I' \subset I$ of size ℓ must contain at least two items of some cluster of size $\geq L_0$. Thus, this detects a duplicate in $O(\ell\log\ell) \subseteq O\big((C(L_0)+D(L_0))\max\{1,\log D(L_0)\}\big)$ comparisons. ◁

▷ Claim 6. There exists an $\mathcal{A}$ with $\text{Universal}(\mathcal{A}, G) \in O\Big(\log\log n\cdot\min_{\substack{L\geq 2\\ C(L)<n}}\big(n+C(L)\log\frac{C(L)}{L}\big)\Big)$.

Proof. Let $L_0 = \arg\min_{L\geq 2}^{C(L)<n}(n+C(L)\log\frac{C(L)}{L})$. Suppose that we guess a $C > C(L_0)$ and an $L \leq L_0$. Then, we can find a duplicate by finding the C smallest elements I' of I with linear time selection, and running `Median Recursion` with this L on those elements. Indeed, either the element of rank exactly C is a duplicate, and gets found during the linear time selection, or every cluster in I' is also a cluster in I. In the latter case, since $C > C(L_0)$, there is a cluster of size $\geq L_0$ among I', so `Median Recursion` find a duplicate by Lemma 2. Computing I' takes $O(n)$ time, and the `Median Recursion` takes $O(C + C\log\frac{C}{L})$ time since $|I'| = C$. If we moreover have $C \leq 2C(L_0)$ and $L \geq \frac{L_0^2}{C(L_0)}$, then $O(C + C\log\frac{C}{L}) = O(C(L_0) + C(L_0)\log\frac{C(L_0)}{L_0})$.

To guess C and L, algorithm $\mathcal{A}$ tries $O(\log\log n)$ different values of $\log\frac{C(L)}{L}$ in parallel and uses a doubling strategy to guess C. Since C is always a power of two, the algorithm $\mathcal{A}$ can precomputed the result of the linear selection. See Algorithm 2.

Algorithm 2 Oblivious Median Recursion.

Require: Input I

1: Precompute the 2^k smallest elements of I for $k = 0, \ldots, \lfloor \log n \rfloor$. If we encounter a pair of duplicate elements, **return** it.
2: **for** $i = 1, 2, 4, 8, \ldots, \log(n)$ in parallel **do**
3: **for** $C = 1, 2, 4, 8, \ldots$ **do**
4: Run median recursion with $L = \max(2, C/2^i)$ on the smallest $\min(n, C)$ elements.
5: **end for**
6: **end for**

We now analyze Algorithm 2. Precomputing the smallest 2^k elements of I for every $k = 0, \ldots, \log n$ can be done in $O(n)$ time: First, use linear time selection to find the $2^{\lfloor \log n \rfloor}$ smallest elements of I. Then, repeatedly partition at the median and discard the larger half, similar to performing deterministic quick select for the minimum. We now analyze the nested for loops. Let $C = 2^{\lceil \log(C(L_0)) \rceil}$ and $i = 2^{\lceil \log \log \frac{2C(L_0)}{L_0} \rceil}$. Put $L = \max(2, C/2^i)$. Then, $C \in [C(L_0), 2C(L_0)]$ and $L \leq L_0$, but also $\log \frac{C}{L} \in O(\log \frac{C(L_0)}{L_0})$. Therefore, the (i, C)-iteration of Algorithm 2 will find a duplicate, in $O(C + C \log \frac{C}{L}) = O(C(L_0) + C(L_0) \log \frac{C(L_0)}{L_0})$ comparisons. The number of comparisons of previous $(i, *)$ iterations forms a geometric series, which sums to $O(C + C \log \frac{C}{L}) = O(C(L_0) + C(L_0) \log \frac{C(L_0)}{L_0})$. Since Algorithm 2 tries $O(\log \log n)$ different values of i in parallel, this shows the claim. ◁

Theorem 4 now follows by sequentially simulating the parallel execution of Claim 5 and 6.

4 Lower Bounds for Element distinctness

4.1 No element distinctness algorithm can be $o(\log \log n)$-competitive

Let $\mathcal{A}$ be an arbitrary algorithm for element distinctness. We show an adaptive adversary lower bound to prove that $\mathcal{A}$ cannot be universally $o(\log \log n)$-competitive for every graph G. Formally, let $n \in \mathbb{N}$ be arbitrary. We consider the following game between the algorithm and an (adaptive) adversary on a list X of n labeled elements. In each round, the algorithm compares two elements, and the adversary has to answer with "$<$", "$=$" or "$>$". The goal of the adversary is to survive for $\frac{1}{8} n(\log \log n - 1)$ rounds without ever answering "$=$". Afterwards, the adversary constructs the input I from X by picking an ordered set U and assigning each $x \in X$ a value in U, so that the pairwise comparisons on I are consistent with all given answers. It then sets $G = G(I)$. This ensures that Universal$(\mathcal{A}, G) \geq \frac{1}{8} n(\log \log n - 1)$. Finally, the adversary has to pick an algorithm $\mathcal{A}'$ with Universal$(\mathcal{A}', G) \in O(n)$.

The strategy. The adversary uses the following strategic game to answer the comparisons: Imagine a complete binary tree T of height $n \log n$, where each element $x \in X$ is stored at some node $u(x)$ in the tree. Initially, all elements are placed at the root of T. In the end, the adversary will pick U to be the set of leaves of T, ordered from left to right. When the algorithm compares two elements $x, y \in X$, the adversary answers as follows:

- If there exists a node $v \in T$ such that $u(x)$ lies in the left subtree of v and $u(y)$ lies in the right subtree of v, answer "$x < y$".
- If there exists a node $v \in T$ such that $u(x)$ lies in the right subtree of v and $u(y)$ lies in the left subtree of v, answer "$y < x$".
- If $u(x) = u(y) = v$, move $u(x)$ to the left of v and $u(y)$ to the right. Answer "$x < y$".

- If $u(x)$ and $u(y)$ lie on the same root-to-leaf path and $u(y)$ is in the right subtree of $u(x)$, put $u = u(x)$, move $u(x)$ to the left child of u, and answer "$x < y$". If $u(y)$ is in the left subtree of $u(x)$ instead, put $v = u(x)$, move $u(x)$ to the right of v, and answer "$y > x$".
- If $u(x), u(y)$ lie on the same root-to-leaf path and $u(y)$ is above $u(x)$ is symmetric.

The adversary always moves at most two elements down the tree and these elements move one step. After $\frac{1}{8}n(\log\log n - 1)$ rounds, not too many elements can be deep in the tree:

▷ Claim 7. After $\frac{1}{8}n(\log\log n - 1)$ rounds have passed, $\exists i \in \{\lfloor \frac{1}{2}\log\log n\rfloor, \ldots, \lfloor \log\log n\rfloor - 2\}$ such that there are strictly fewer than $n/2^i$ elements of depth at least 2^i.

Proof. We show the claim via an indirect proof: If there is no such i, then the total depth of all elements is at least

$$\sum_{i=\lfloor \frac{1}{2}\log\log n\rfloor}^{\lfloor \log\log n\rfloor - 2} 2^i \cdot \left(\frac{n}{2^i} - \frac{n}{2^{i+1}}\right) = \sum_{i=\lfloor \frac{1}{2}\log\log n\rfloor}^{\lfloor \log\log n\rfloor - 2} \frac{n}{2} = \frac{n}{2}\left(\lfloor \log\log n\rfloor - \lfloor \frac{1}{2}\log\log n\rfloor + 1 - 2\right) > \frac{n}{4}(\log\log n - 1),$$

so more than $\frac{1}{8}n(\log\log n - 1)$ rounds have passed. This shows the claim. ◁

The core idea is as follows: the adversary plays the game without ever revealing a duplicate. After $\frac{1}{8}n(\log\log n - 1)$ rounds, it collects all comparisons made so far. It then chooses an integer L and defines a graph G on X that contains many clusters of size L, formed by grouping elements that share a root-to-leaf path. It then moves all elements in a cluster down to the leaf of this path. Let U be the set of leaves of T, ordered from left to right. If $X = (x_1, \ldots, x_n)$, put $I = (u(x_1), \ldots, u(x_n))$, then $G(I) \cong G$. Since elements are only moved downward, the pairwise comparisons on I remain consistent with the adversary's answers. Hence, we obtain an input I of n elements from U, on which, after $\frac{1}{8}n(\log\log n - 1)$ comparisons, algorithm $\mathcal{A}$ has not yet found a duplicate, although one exists. Finally, we show that a clairvoyant algorithm, knowing G, can solve all inputs I' with $G(I') \cong G$ in linear time. Therefore, for this graph G, there exists an algorithm with linear universal running time, whereas $\mathcal{A}$ has universal running time $\Omega(n\log\log n)$.

▶ **Theorem 8.** *For every algorithm $\mathcal{A}$ and every n, there exists an n-vertex graph G and an algorithm $\mathcal{A}'$ such that* $\text{Universal}(\mathcal{A}, G) \geq \frac{1}{8}n(\log\log n - 1)$*, but* $\text{Universal}(\mathcal{A}', G) \in O(n)$.

Proof. Let X be a list of n labeled elements. Run $\mathcal{A}$ on X and play the adversarial game that we described. After $\frac{1}{8}n(\log\log n - 1)$ rounds, let i be as in Claim 7 and set $L = \frac{n}{2^{2^i+1}} > 1$.

To define the input I, let U be the set of leaves of T, ordered from left to right. To turn X into I, we move each element of X into a leaf of T. More precisely: Observe that, after $\frac{1}{8}n(\log\log n - 1)$ comparisons, there can be no element in X where $u(x)$ is in a leaf of T. While there is an unoccupied leaf in T with root-to-leaf path π for which at least L elements $x \in X$ have $u(x)$ in π, pick an arbitrary subset X' of L of these elements. Intuitively, we form X' into a cluster of size L. Formally, move all elements in X' down along π to the unoccupied leaf. Repeat this procedure until no such path exists anymore. Then, one-by-one, move all remaining elements down to an unoccupied leaf of T. Put $I = (u(x_1), \ldots, u(x_n))$ where $X = (x_1, \ldots, x_n)$ then I is a list of elements from U and all answers given by the strategic game are consistent with U. Since the game never answers equality, the algorithm $\mathcal{A}$ did not find a duplicate on I after $\frac{1}{8}n(\log\log n - 1)$ comparisons.

Let $G := G(I)$. Since all elements in G either lie in a cluster of size L, or in a singleton cluster, it follows that $C(L)$ equals the number of singleton elements. We use the original state of the tree T to bound this quantity. By Claim 7, our tree T had fewer than $\frac{n}{2^i}$ elements

of depth at least 2^i. On the other hand, the elements of depth $\leq 2^i$ lie on at most $\leq 2^{2^i}$ distinct paths from the root. This counting argument shows that there are at most $L \cdot 2^{2^i}$ such elements that were not packed into a cluster, hence

$$C(L) \leq \frac{n}{2^i} + L \cdot 2^{2^i} \leq \frac{n}{2^i} + \frac{n2^{2^i}}{2^{2^{i+1}}} = \frac{n}{2^i} + \frac{n}{2^{2^i}} \leq \frac{n}{2^{i-1}},$$

First, this implies $C(L) < n$ since $i \geq \lfloor \frac{1}{2} \log \log n \rfloor$. Second, this implies

$$C(L) \log \frac{C(L)}{L} \leq C(L) \log \frac{n}{L} \leq \frac{n}{2^{i-1}} \log 2^{2^{i+1}} = 4n.$$

We choose $\mathcal{A}'$ as follows: First, $\mathcal{A}'$ runs the `Median Recursion` algorithm with parameter L. If `Median Recursion` gives up without finding a duplicate, then $\mathcal{A}'$ sorts the input to check for a duplicate, so $\mathcal{A}'$ is correct. By Lemma 2, for any input I' such that $G(I') \cong G$, the running time of this algorithm is $O(n)$ as $\mathcal{A}'$ always finds a duplicate via `Median Recursion`. ◀

4.2 A $\log \log n$-competitive algorithm

We complement Theorem 8 by showing that the algorithm in Theorem 4 is $O(\log \log n)$-competitive. I.e., using an adaptive adversary, we show that for all graphs G no algorithm $\mathcal{A}$ can beat the corresponding clairvoyant algorithm from Theorem 3.

Fix some $G \in \mathbb{G}$ that contains at least one cluster of size at least two. Then, for every choice of L, this induces some $C(L)$ and $D(L)$. Consider any algorithm $\mathcal{A}$. The adversary employs the strategic game from Section 4.1 on a list X of n labeled elements. We compute some M depending on the values of $C(L)$ and $D(L)$ for every L. The adversary then plays the game for M rounds without revealing a duplicate. After M rounds, all elements of X are in some node of a tree T. The adversary then moves all elements from X to the leaves of T in a specific manner. The input $I[i]$ is the leaf that contains the i'th element from X and we prove that $G(I) \cong G$. The pairwise comparisons on I are automatically consistent with the adversary's answers. Thus, for this input I, the algorithm $\mathcal{A}$ fails to find a duplicate after M comparisons, even though I contains at least one pair of equal elements.

4.2.1 A lower bound complementing Lemma 2

We first show a lower bound that complements Lemma 2, but that is missing the $O(n)$-term. This already shows that our algorithm from Theorem 4 is $O(\log \log n)$-competitive for inputs that require at least a linear number of comparisons.

▷ Claim 9. Fix some $G \in \mathbb{G}$ and define $M := \min_{L \geq 2}^{C(L)<n} \left(\frac{1}{4} C(L) \cdot \max\{0, \log_2 \frac{C(L)}{2L}\} \right)$. Play the adversarial game on a list X of size n for exactly M rounds. Let be T the resulting state of the tree. Then, there exists a graph G' on X that is isomorphic to G such that for each cluster in G', all elements share a root-to-leaf path in T.

Proof. Recall that any graph $G \in \mathbb{G}$ consists of only edges that are in a cluster (clique). We consider the clusters in G in decreasing order of size. Let the current cluster that we consider have size L. We find some root-to-leaf path that contains at least L unused elements, use these elements to form a cluster of size L, and remove them from T. This procedure, if successful, creates a graph G' on X that is isomorphic to G.

What remains is to prove that, for all L that we consider, there exists such a root-to-leaf path. If there exists an integer $L \geq 2$ with $C(L) < n$, such that $\log_2 \frac{C(L)}{2L} \leq 0$, then $M \leq 0$. However, this means we have played zero rounds, so all elements of X are at the root of T and

we can create any graph on X, which shows the claim. Therefore, suppose that for all $L \geq 2$, we have $\log \frac{C(L)}{2L} > 0$. When considering a cluster of size L, there are per definition at least $C(L)$ unused elements in the tree. Also, since at most $\frac{1}{4}C(L) \log \frac{C(L)}{2L}$ rounds have passed, there are at most $\frac{1}{2}C(L)$ elements of depth $\geq \log \frac{C(L)}{2L}$. Hence, there are at least $\frac{1}{2}C(L)$ unused elements of depth $\leq \lfloor \log \frac{C(L)}{2L} \rfloor$. These elements lie on $2^{\lfloor \log \frac{C(L)}{2L} \rfloor} \leq \frac{C(L)}{2L}$ different root-to-leaf paths. Thus, one of these paths contains at least $\frac{C(L)}{2} / \frac{C(L)}{2L} = L$ elements. ◁

Claim 9 immediately implies the following lower bound.

▶ **Proposition 10.** *Let $G \in \mathbb{G}$ and define $M := \min_{L \geq 2}^{C(L)<n} \left(\frac{1}{4}C(L) \cdot \max\{0, \log_2 \frac{C(L)}{2L}\} \right)$. Then every algorithm $\mathcal{A}$ satisfies:* $\text{Universal}(\mathcal{A}, G) \geq M$.

Proof. For any algorithm $\mathcal{A}$, the adversary plays M rounds and defines the universe U to be the set of leaves of T. Then, it applies Claim 9 and pushes all elements in a cluster down to the same distinct leaf. This defines input I with $G(I) \cong G$, and all answers given by the adversary are consistent with I. It follows that, on this input, $\mathcal{A}$ did not find a duplicate after M comparisons, even though there is a duplicate. ◀

4.2.2 A lower bound complementing Lemma 1

We use our adversarial game to construct a lower bound matching Lemma 1.

▶ **Definition 11.** *Let $G \in \mathbb{G}$ be an n-vertex graph with at least two clusters. We define by G' the graph that is obtained by iteratively deleting the largest cluster from G until G' has n' vertices with $n' \leq \frac{3}{4}n$. Define for all L, the value $C'(L)$ as the total size of clusters in G' of size strictly less than L, and by $D'(L)$ the number of clusters of size at least L.*

Consider the adversarial game on a list $X = (x_1, \ldots, x_n)$ of n elements. If this game lasts $\Omega(n)$ rounds, then Proposition 10 is tight. Therefore, suppose that this game lasts at most $\frac{n}{8}$ rounds. We preset a different construction for transforming X into an input I with $G(I) \cong G$ that yields a stronger bound in this regime. The idea behind this construction is as follows: denote by T the state of our binary tree after at most $\frac{n}{8}$ rounds. Recall that, in each round, the adversary selects at most two elements $x \in X$ and moves $u(x)$ down the tree by one step. It follows that there are still $\frac{3}{4}n$ elements in X for which $u(x)$ is at the root. Since $n' \leq \frac{3}{4}n$, we can always realize G' on these elements. However, we can do even better and use G' to get rid of all elements that are not stored at the root.

Our reconstruction algorithm. First, we consider the graph G' from Definition 11 and we iterate over its clusters in decreasing order by size. For each cluster C, we pick a non-root node $v \in T$ of minimal depth that contains at least one element. If there are at least $|C|$ elements $x \in X$ for which $u(x) = v$, then we use $|C|$ of these elements to form a cluster of size $|C|$, and remove them from T. If there are fewer than $|C|$ such elements, then we add elements $x \in X$ where $u(x)$ is the root of T. Since $n' \leq \frac{3}{4}n$, this is always possible. Finally, if we run out of non-root nodes of T, then all remaining clusters of G (including those not in G') are formed by iteratively selecting an arbitrary set of elements $x \in X$ with $u(x) = \text{root}(T)$.

▷ Claim 12. If we can reconstruct all clusters in G' without running out of non-root elements, then the total depth of all elements is $> \frac{1}{16} \min_{L \geq 1}^{C'(L)<n'} ((C'(L) + D'(L)) \max\{1, \log D'(L)\})$

Proof. Let T be the original state of our binary tree. We call an element $x \in X$ a *root* element if $u(x)$ is in the root of the original tree T and a non-root element otherwise. We call a cluster in G' *whole* if its corresponding cluster in X consists exclusively of non-root elements. We call the cluster *split* otherwise. We make the following observations:

1. If there are k split clusters so far, then all non-root elements have depth $\geq \max\{1, \log k - 1\}$.
2. Every whole cluster consists of only non-root elements.
3. Every split cluster contains at least one non-root element.

Let $U_1, \ldots, U_m$ be the clusters in G' in increasing order of size. Pick the smallest integer i such that at least half the clusters in $U_i, U_{i+1}, \ldots, U_m$ are split. Put $L = |U_i|$, then there are at least $\frac{1}{2}D'(L)$ such split clusters. Thus, by (1) and (3), the total depth of all elements is at least $\frac{1}{4}D'(L)\max\{1, \log(\frac{1}{4}D'(L)) - 1\}$. On the other hand, since i is minimal, for any $j \leq i$, at most half the clusters $U_j, U_{j+1}, \ldots, U_{i-1}$ are split. Therefore, among $U_1, \ldots, U_{i-1}$, we can match each split cluster to a distinct non-smaller whole cluster. Hence, the total size of whole clusters among $U_1, \ldots, U_{i-1}$ is at least $\frac{1}{2}C'(L)$. Thus, by (1) and (2), the total depth of all elements is at least $\frac{1}{2}C'(L)\max\{1, \log(\frac{1}{2}D'(L)) - 1\}$. Putting things together, the total depth of all elements is

$$\geq (\frac{1}{4}D'(L) + \frac{1}{2}C'(L))\max\{1, \log(\frac{1}{4}D'(L)) - 1\} \geq \frac{1}{16}(D'(L) + C'(L))\max\{1, \log D'(L)\},$$

Since $L = |U_i|$, we have $L \geq 1$ and $C'(L) < n'$. ◁

▶ **Lemma 13.** *Let $G \in \mathbb{G}$ be an n-vertex graph with at least two clusters and G' be the graph from Definition 11. Then:* $\min(n'/8, \frac{1}{32}\min_{\substack{L \geq 1 \\ C'(L) < n'}}(C'(L) + D'(L))\max\{1, \log D'(L)\})$ $\geq \frac{1}{256}\min(n, \min_{\substack{L \geq 1 \\ C(L) < n/2}}(C(L) + D(L))\max\{1, \log D(L)\})$.

Proof. We note that G' is non-empty since G consists of at least two clusters. Let L_1 be the size of the smallest cluster that was deleted from G (i.e., the last cluster that we deleted before we obtained G') and let

$$L_0 = \arg \min_{\substack{L \geq 1 \\ C'(L) < n'}} (C'(L) + D'(L))\max\{1, \log D'(L)\}).$$

Then $L_0 \leq L_1$ since $C'(L_0) < n'$, so $C'(L_0) = C(L_0)$. Suppose first that $L_1 \geq \frac{n}{4}$. Then, we deleted a single cluster of size L_1, so $D'(L_0) = D(L_0) - 1$. This implies

$$(C'(L_0) + D'(L_0))\max\{1, \log D'(L_0)\}) \geq \frac{1}{4}(C(L_0) + D(L_0))\max\{1, \log D(L_0)\}.$$

Otherwise, we have $L_1 < \frac{n}{4}$. We now make a case distinction. If $C(L_0) \geq \frac{n}{2}$, then

$$(C'(L_0) + D'(L_0))\max\{1, \log D'(L_0)\}) \geq \frac{n}{2}.$$

Otherwise, we consider all clusters of size $\geq L_0$. Initially, there were $D(L_0)$ such clusters, of total size $n - C(L_0) \geq \frac{n}{2}$. We deleted one cluster of size L_1, plus some clusters of total size at most $\frac{n}{4}$. This is at most half of $\frac{n}{2}$, and we always deleted a largest cluster, so we deleted at most $1 + \frac{1}{2}D(L_0)$ clusters in total. This shows $D'(L_0) \geq \frac{1}{2}D(L_0) - 1$. We also have $D'(L_0) \geq 1$ since $C'(L_0) < n'$. Thus,

$$(C'(L_0) + D'(L_0))\max\{1, \log D'(L_0)\}) \geq \frac{1}{8}(C(L_0) + D(L_0))\max\{1, \log D(L_0)\}.$$ ◀

▶ **Proposition 14.** *Let* $G \in \mathbb{G}$ *and define* $M := \frac{1}{256} \min\Big\{n, \min_{L\geq 1}^{C(L)<n/2}(C(L)+D(L)) \max\{1, \log D(L)\}\Big\}$. *Then every algorithm* $\mathcal{A}$ *satisfies* $\text{Universal}(\mathcal{A}, G) \geq M$.

Proof. If G contains only one cluster, then the proposition holds since $M < 1$, while any algorithm needs to do at least one comparison. Otherwise, let $M' = \min(n'/8, \frac{1}{32} \min_{L\geq 1}^{C'(L)<n'}(C'(L)+D'(L)) \max\{1, \log D'(L)\})$. For any algorithm $\mathcal{A}$, the adversary plays the game for M rounds. By Lemma 13, $M' \geq M$, so the total depth of all elements is at most $2M'$. Thus, by Claim 12, there exists a graph G'' on X that is isomorphic to G such that, for each cluster in G'', all elements share a root-to-leaf path in T. Let U be the set of leaves of T. By pushing the elements in the same cluster down to a distinct leaf, the adversary constructs an input I such that $G(I) = G'' \cong G$ and all answers given by it are consistent with I. It follows that, on this input, $\mathcal{A}$ did not find a duplicate after M comparisons, even though there is a duplicate. ◀

4.3 Concluding our argument

▶ **Theorem 15.** *Let* $G \in \mathbb{G}$. *Then every algorithm* $\mathcal{A}$ *satisfies* $\text{Universal}(\mathcal{A}, G) \geq \frac{1}{256} \min\Big(n + \min_{\substack{L\geq 2 \\ C(L)<n}} C(L) \log \frac{C(L)}{L}, \min_{\substack{L\geq 1 \\ C(L)<n/2}} (C(L)+D(L)) \max\{1, \log D(L)\}\Big)$.

Proof. This is the direct combination of Propositions 10 and 14. ◀

By combining Theorem 4 with Theorem 15, we get the following (which is tight by Theorem 8):

▶ **Theorem 16.** *There is an algorithm* $\mathcal{A}$ *for element distinctness that is universally* $O(\log\log n)$*-competitive, that is, there is a constant* $c > 0$ *such that for every* $G \in \mathbb{G}$ *and all algorithm* $\mathcal{A}'$*, we have* $\text{Universal}(\mathcal{A}, G) \leq c \max\{1, \log\log(n)\} \cdot \text{Universal}(\mathcal{A}', G)$.

References

1. Peyman Afshani, Jérémy Barbay, and Timothy Chan. Instance-optimal geometric algorithms. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 129–138. Association for Computing Machinery, 2009. doi:10.1145/3046673.
2. Miklós Ajtai. A lower bound for finding predecessors in Yao's cell probe model. *Combinatorica*, 8(3):235–247, 1988. doi:10.1007/BF02126797.
3. Paul Beame, Allan Borodin, Michael Saks, and Sven Skyum. Time–space tradeoffs for branching programs. *Journal of Computer and System Sciences*, 44(2):272–298, 1991. doi:10.1016/0022-0000(91)90035-2.
4. Paul Beame, Raphael Clifford, and Widad Machmouchi. Element distinctness, frequency moments, and sliding windows. In *Symposium on Foundations of Computer Science (FOCS)*, FOCS '13, pages 290–299, USA, 2013. IEEE Computer Society. doi:10.1109/FOCS.2013.39.
5. Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M. Jungers, and J. Ian Munro. Sorting under partial information (without the ellipsoid algorithm). In *ACM Symposium on Theory of Computing (STOC)*, STOC '10, New York, NY, USA, June 2010. Association for Computing Machinery. doi:10.1145/1806689.1806740.
6. Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *Symposium on Theory of Computing (STOC)*, STOC '08, pages 641–650, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1374376.1374470.
7. Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Patrascu. Dynamic optimality – Almost. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 484–490, USA, 2004. IEEE Computer Society. doi:10.1109/FOCS.2004.23.

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

9 Bernhard Haeupler, Richard Hladík, Václav Rozhon, Robert E. Tarjan, and Jakub Tetek. Universal optimality of dijkstra via beyond-worst-case heaps. In *Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2099–2130. IEEE, 2024. doi:10.1109/FOCS61266.2024.00125.

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

11 Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In *ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1166–1179, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451081.

12 Yijie Han. Deterministic sorting in o(n log log n) time and linear space. In *ACM Symposium on Theory of Computing (STOC)*, STOC '02, pages 602–608, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/509907.509993.

13 Yijie Han and Mikkel Thorup. Integer sorting in 0(n sqrt (log log n)) expected time and linear space. In *Foundations of Computer Science (FOCS)*, pages 135–144. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181890.

14 Richard Hladík and Jakub Tětek. Near-Universally-Optimal Differentially Private Minimum Spanning Trees. In Mark Bun, editor, *Foundations of Responsible Computing (FORC)*, volume 329 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:19, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FORC.2025.6.

15 Jeff Kahn and Jeong Han Kim. Entropy and sorting. In *ACM symposium on Theory of Computing (STOC)*, STOC '92, pages 178–187, New York, NY, USA, July 1992. Association for Computing Machinery. doi:10.1145/129712.129731.

16 Jeff Kahn and Michael Saks. Balancing poset extensions. *Order*, 1(2):113–126, June 1984. doi:10.1007/BF00565647.

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

18 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, SODA '16, pages 1272–1287, USA, 2016. Society for Industrial and Applied Mathematics. doi:10.5555/2884435.2884524.

19 J. Misra and David Gries. Finding repeated elements. *Science of Computer Programming*, 2(2):143–152, 1982. doi:10.1016/0167-6423(82)90012-0.

20 Anna Ostlin and Rasmus Pagh. Uniform hashing in constant time and linear space. In *ACM Symposium on Theory of Computing (STOC)*, STOC '03, pages 622–628, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/780542.780633.

21 Boaz Patt-Shamir and David Peleg. Time-space tradeoffs for set operations. *Theoreticl Compututer Science*, 110(1):99–129, March 1993. doi:10.1016/0304-3975(93)90352-T.

22 Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006. doi:10.1137/S0097539705447256.

23 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, July 1985. doi:10.1145/3828.3835.

24 Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. A combinatorial proof of universal optimality for computing a planar convex hull. In Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman, editors, *European Symposium on Algorithms (ESA)*, volume 351 of *LIPIcs*, pages 102:1–102:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.102.

25 Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. Simpler optimal sorting from a directed acyclic graph. In *Symposium on Simplicity in Algorithms (SOSA)*. SIAM, 2025. doi:10.1137/1.9781611978315.26.

26 Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. Simpler universally optimal dijkstra. In *European Symposium on Algorithms (ESA)*, volume 351 of *LIPIcs*, pages 71:1–71:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.71.

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

28 Andrew Chi-Chih Yao. Near-optimal time-space tradeoff for element distinctness. *SIAM Journal on Computing*, 23(5):966–975, 1994. doi:10.1137/S0097539788148959.

29 Goran Zuzic, Gramoz Goranci, Mingquan Ye, Bernhard Haeupler, and Xiaorui Sun. Universally-optimal distributed shortest paths and transshipment via graph-based ℓ1-oblivious routing. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2549–2579, 2022. doi:10.1137/1.9781611977073.100.

A Preprocessing Algorithms for Element distinctness

First, we consider a variant of universal optimality that was proposed by Cardinal et al. [5, STOC'10] who consider the following: "If one is allowed to preprocess the universally fixed graph G, can one then design competitive algorithms?"

Formally, we consider a two-stage algorithmic problem where the algorithm is first given a graph G to preprocess. Then, the algorithm gets the list I of n elements from some ordered set U, with the promise that $G(I) \cong G$. From this point on, the algorithm can do comparisons between elements in I. In this model, we want to minimize both the processing time and the number of comparisons performed. To avoid trivial lower bounds from reading a dense graph G, we encode the graph with m clusters as a sequence $s_1, \ldots, s_m$ of cluster sizes where $s_1 + \cdots + s_m = n$.

By Theorem 15, even with unbounded preprocessing time, any algorithm needs to perform at least

$$\frac{1}{256} \min\left(n + \min_{\substack{L\geq 2\\ C(L)<n}} C(L)\log\frac{C(L)}{L}, \min_{\substack{L\geq 1\\ C(L)<n/2}} (C(L)+D(L))\max\{1,\log D(L)\}\right)$$

comparisons.

We show that, after $O(m)$ preprocessing, we can give an $O(1)$-competitive algorithm.

▶ **Theorem 17.** *There is an algorithm that can preprocess any $G \in \mathbb{G}$ encoded as as $s_1, \ldots, s_m$ in $O(m)$ time, and then, given an input I with $G(I) \cong G$, can find a duplicate in*

$$O\Big(\min\left(n + \min_{\substack{L\geq 2\\ C(L)<n}} C(L)\log\frac{C(L)}{L}, \min_{\substack{L\geq 1\\ C(L)<n/2}} (C(L)+D(L))\max\{1,\log D(L)\}\right)\Big)$$

comparisons.

Proof. Let

$$L_1 = \arg\min_{\substack{L\geq 2\\ C(L)<n}} C(L)\log\frac{C(L)}{L}, \quad L_2 = \arg\min_{\substack{L\geq 1\\ C(L)<n/2}} (C(L)+D(L))\max\{1,\log D(L)\}.$$

We first describe such an algorithm with $O(nm)$ preprocessing time: For each $L \in [n]$, compute $C(L)$ and $D(L)$ in $O(m)$ time. Determine L_1 and L_2. Then, given I, run either median recursion with $L = L_1$, or block sorting with $k = 2D(L_2)$. By Lemmas 1 and 2, this yields the number-of-comparisons bound in Theorem 17.

To reduce the preprocessing time to $O(m \log m)$, observe that $C(L)$ and $D(L)$ are step functions that only change when $L = s_i$ for some $i \in m$. Hence, there are $O(m)$ possible values for L_1 and L_2. We can sort $s_1, \ldots, s_m$ in $O(m \log m)$ time, and compute $C(L), D(L)$ for the relevant values of L via sweep line in $O(m)$ time. Alternatively, we can achieve a preprocessing time of $O(n)$ by bucket sorting $s_1, \ldots, s_m$. (This works even on a pointer machine, since $s_1 + \cdots + s_m = n$.)

We can further reduce the preprocessing time to $O(m)$. If $(C(L_2) + D(L_2)) \max\{1, \log D(L_2)\} \in \Omega(n)$, then the algorithm has to do $\Omega(n)$ comparisons, so we skip doing any preprocessing and compute $C(.), D(.), L_1, L_2$ in $O(n)$ time once we get access to I. Otherwise, L_1 is irrelevant (due to the $O(n + \ldots)$ term) and we only have to consider L_2. We observe that we do not need to compute L_2 exactly. In fact, it suffices to approximate $\max\{1, \log D(L_2)\}$ up to a constant factor. Consider the following recursive procedure that operates on some list S', where initially $S' = s_1, \ldots, s_m$: Recursively find the median of S', partition at it, and recurse on the half of S' that contains values at least as big as the median. For $j = 1, \ldots, \lceil \log m \rceil$, let t_j be the minimum value of S' in the j-th recursive call. During the recursion, for each j, compute $C(t_j)$ and $D(T_j)$. Afterwards, put

$$L_2' = \arg \min_{j=1}^{\lceil \log m \rceil} \big((C(t_j) + D(t_j)) \max\{1, \log D(T_j)\} \big).$$

The recursive procedure takes $O(m)$ time in total (even on a pointer machine). Suppose that the ℓ-th recursive calls is the final recursive call where $L_2 \in S'$. Then $t_\ell \leq L_2$ and $D(t_\ell) \leq 2D(L_2) + 1$. This implies

$$\begin{aligned}(C(L_2') + D(L_2')) \max\{1, \log D(L_2')\} &\leq (C(t_\ell) + D(t_\ell)) \max\{1, \log D(t_\ell)\} \\ &\leq 3(C(L_2) + D(L_2)) \max\{1, \log D(L_2)\}.\end{aligned}$$

Thus, in this case, running block recursion with $k = 2D(L_2')$ yields the number-of-comparisons bound in Theorem 17. ◀

B Order Optimality for Element distinctness

The main body of this paper considers universal optimality where the fixed object is an undirected graph G. We observed that each input I can induce an undirected graph $G(I)$ where the vertex set is $\{1, 2, \ldots, n\}$ and there is an edge between i and j if and only if $I[i] = I[j]$. For every fixed undirected graph $G \in \mathbb{G}_n$, we were interested in an algorithm's worst-case over all inputs I where $G(I) \cong I$.

However, we noted in the introduction that previous papers [1, 27, 10] instead consider fixing, for (partially) ordered input I, the directed acyclic graph $\overrightarrow{G}(I)$. This is a directed graph where the vertex set is $\{1, 2, \ldots, n\}$ and there is an edge from i to j if and only if $I[i] < I[j]$. Note that the undirected complement of $\overrightarrow{G}(I)$ is $G(I)$.

In the introduction, we observed that universally optimality can also be defined by fixing a such DAG. Recall that $\overrightarrow{\mathbb{G}}_n$ is the set of all n-vertex DAGs $\overrightarrow{G}$ where the undirected complement of $\overrightarrow{G}$ is in $\mathbb{G}$, and for $\overrightarrow{G} \in \overrightarrow{\mathbb{G}}_n$, we defined

$$\text{Order}(\mathcal{A}, \overrightarrow{G}) := \max_{I : \overrightarrow{G}(I) \cong \overrightarrow{G}} \text{Running-Time}(\mathcal{A}, I).$$

Observe that this definition is equivalent to fixing I and considering an algorithm's worst-case running time over all input lists that are a permutation of I (thus, it is equivalent to the notion of *instance optimality in the order-oblivious setting* in [1]). Recall that, to distinguish between our two models of optimality, we say that an algorithm is $O(f(n))$-*order-competitive* if there exists a function $g \in O(f)$ such that, for all sufficiently large n, for all graphs $\overrightarrow{G} \in \overrightarrow{\mathbb{G}}_n$, and for all algorithms $\mathcal{A}'$, we have

$$\text{Order}(\mathcal{A}, \overrightarrow{G}) \leq g(n) \cdot \text{Order}(\mathcal{A}', \overrightarrow{G}).$$

We show that a simple doubling algorithm achieves tight order-competitive bounds. Consider the following algorithm $\mathcal{A}$: Starting from $k = 2$, pick $\min(n, k)$ arbitrary elements and sort them. If there is a pair of equal elements, output them, otherwise double k and repeat.

▶ **Lemma 18.** *The algorithm $\mathcal{A}$ is $O(\log n)$-order-competitive, that is, we prove that there exists a constant $c > 0$ such that, for every n-vertex DAG $\overrightarrow{G} \in \overrightarrow{\mathbb{G}}$ and every algorithm $\mathcal{A}'$, we have* $\text{Order}(\mathcal{A}, \overrightarrow{G}) \leq c \max\{1, \log(n)\} \cdot \text{Order}(\mathcal{A}', \overrightarrow{G})$.

Proof. Let $\mathcal{A}'$ and $\overrightarrow{G}$ be arbitrary. Let k be the largest power of two for which there is an input I with $\overrightarrow{G}(I) \cong \overrightarrow{G}$ so that $\mathcal{A}$, when run on I, does not find a duplicate for this value of k. Then $\mathcal{A}$ always finds a duplicate when sorting at most $2k$ elements, so $\text{Order}(\mathcal{A}, \overrightarrow{G}) \in O(k \log k) \subseteq O(k \log n)$. On the other hand, since $\mathcal{A}$ did not find a duplicate, there are k distinct elements in I. There is a permutation I' of I such that the first $k/2$ comparisons by $\mathcal{A}'$ happen among these k elements. Then, $\overrightarrow{G}(I') \cong \overrightarrow{G}(I)$ and $\mathcal{A}'$ does not find a duplicate in I' after $k/2$ comparisons, so $k/2 \leq \text{Order}(\mathcal{A}', \overrightarrow{G})$. ◀

We show that Lemma 18 is tight: no algorithm is universally $o(\log n)$-competitive.

▶ **Lemma 19.** *For every algorithm $\mathcal{A}$ and every positive integer n, there is a DAG $\overrightarrow{G}$ and an algorithm $\mathcal{A}'$ such that* $\text{Order}(\mathcal{A}, \overrightarrow{G}) \in \Omega(n \log n)$ *but* $\text{Order}(\mathcal{A}', \overrightarrow{G}) \in O(n)$.

Proof. Let $\mathcal{A}$ and n be arbitrary. Let G be an undirected graph with 1 cluster of size 2 and $n - 2$ clusters of size 1. By Proposition 10, there is an input I with $G(I) \cong G$ and $\text{Running-Time}(\mathcal{A}, I) \in \Omega(n \log n)$. Let $\overrightarrow{G} = \overrightarrow{G}(I)$ and let k be the rank in $\overrightarrow{G}$ of the size-2 cluster. On the input I, the algorithm $\mathcal{A}'$ selects the elements of rank k and $k+1$, respectively, in $O(n)$ time (e.g. via quick select). Then, $\mathcal{A}'$ outputs these two elements. ◀

C Set Intersection

We finally consider universal optimality and order optimality for the set intersection problem. Recall that the input is two lists A and B, each of size n, and that $G(A, B)$ is a graph with $2n$ vertices. We use *A-vertices* and *B-vertices* to refer to the n vertices that correspond to elements in A or B, respectively. The connected components of $G(A, B)$ are cliques, which we call *clusters*. The *A-size* and *B-size* of a cluster is the number of A-vertices or B-vertices, respectively. We denote by $\mathbb{G}_{n,n}$ the set of all such graphs with such a partition into A-vertices and B-vertices. Two graphs in $\mathbb{G}_{n,n}$ are isomorphic if there is an isomorphism of undirected graphs that sends A-vertices to A-vertices and B-vertices to B-vertices.

For fixed $G \in \mathbb{G}_{n,n}$, we define: $\text{Universal}(\mathcal{A}, G) = \max_{(A,B):\, G(A,B) \cong G} \text{Running-Time}(\mathcal{A}, A, B)$.

We denote by $\overrightarrow{G}_{n,n}$ the set of all $2n$-vertex DAGs, with a partition into A-vertices and B-vertices, whose undirected complement lies $\mathbb{G}_{n,n}$.

For fixed $\overrightarrow{G} \in \overrightarrow{\mathbb{G}}_{n,n}$, we define: $\text{Order}(\mathcal{A}, \overrightarrow{G}) = \max_{(A,B):\, \overrightarrow{G}(A,B) \cong \overrightarrow{G}} \text{Running-Time}(\mathcal{A}, A, B)$.

We show that a simple doubling algorithm achieves tight bounds for the set intersection problem. The algorithm $\mathcal{A}$ works as follows: Starting from $k = 2$, pick $\min(|A|, k)$ elements from A and $\min(|B|, k)$ from B and sort them. If there is a pair of equal elements from A and B, output them, otherwise double k and repeat.

▶ **Lemma 20.** *The algorithm $\mathcal{A}$ is $O(\log n)$-competitive, that is, there is a function $f \in O(\log n)$ such for all $G \in \mathbb{G}_n$ and for every algorithm $\mathcal{A}'$, we have*

$$\text{Universal}(\mathcal{A}, G) \leq f(n) \cdot \text{Universal}(\mathcal{A}', G).$$

Proof. The proof closely resembles that of Lemma 18. Let $\mathcal{A}'$ and G be arbitrary. Let k be the largest power of two for which there is an input (A, B) with $G(A, B) \cong G$ so that $\mathcal{A}$, when run on (A, B), does not find a set intersection for this value of k. Then $k \leq |A| + |B|$, and $\mathcal{A}$ always finds a set intersection when sorting at most $4k$ elements, so $\text{Universal}(\mathcal{A}, G) \in O(k \log k) \subseteq O(k \log n)$. On the other hand, since $\mathcal{A}$ did not find a set intersection, there are $\min(|A|, k)$ from A and $\min(|B|, k)$ elements from B with no set intersection. Since $|A| + |B| \geq k$, these are $\geq k$ elements in total. There is a permutation A' of A and a permutation B' of B such that the first $k/2$ comparisons by $\mathcal{A}'$ happen among these $\geq k$ elements. Then, $G(A', B') \cong G(A, B)$ and $\mathcal{A}'$ does not find a set intersection in (A', B') after $k/2$ comparisons, so $k/2 \leq \text{Universal}(\mathcal{A}', G)$. ◀

▶ **Lemma 21.** *The algorithm $\mathcal{A}$ is also $O(\log n)$-order competitive. That is, there is a function $f \in O(\log n)$ such for all $\overrightarrow{G} \in \overrightarrow{\mathbb{G}}_n$ and for every algorithm $\mathcal{A}'$, we have*

$$\text{Order}(\mathcal{A}, \overrightarrow{G}) \leq f(n) \cdot \text{Order}(\mathcal{A}', \overrightarrow{G}).$$

Proof. Let $\mathcal{A}'$ and $\overrightarrow{G}$ be arbitrary. Let k be the largest power of two for which there is an input (A, B) with $\overrightarrow{G}(A, B) \cong \overrightarrow{G}$ so that $\mathcal{A}$, when run on (A, B), does not find a set intersection for this value of k. Then, proceed as in the proof of Lemma 20. ◀

We show that Lemmas 20 and 21 are tight: no algorithm is universally $o(\log n)$-competitive, or universally $o(\log n)$-order competitive.

▶ **Lemma 22.** *For every algorithm $\mathcal{A}$ and every positive integer n, there is a graph $G \in \mathbb{G}_{n,n}$ and algorithm $\mathcal{A}'$ such that* $\text{Universal}(\mathcal{A}, G) \in \Omega(n \log n)$ *but* $\text{Universal}(\mathcal{A}', G) \in O(n)$. *Moreover, there is is a DAG $\overrightarrow{G}$ whose undirected complement is G such that such that* $\text{Order}(\mathcal{A}, \overrightarrow{G}) \in \Omega(n \log n)$ *but* $\text{Order}(\mathcal{A}', \overrightarrow{G}) \in O(n)$.

Proof. We first show the bounds on the universal running time. Let $\mathcal{A}$ and n be arbitrary. For $i \in [n^{1/3}]$, consider the graph $G_i \in \mathbb{G}_{n,n}$ with:
(1) For $j \in [n^{1/3}]$, one cluster of A-size j and B-size $\delta_{i,j}$, that is, 1 if $i = j$ and 0 otherwise.
(2) $n - 1$ clusters of A-size 0 and B-size 1.
(3) One cluster of A-size $n - \Theta(n^{2/3})$ and B-size 0.
Note that the cluster of type (1) with A-size i is the only cluster that contains both elements from A and from B. The algorithm $\mathcal{A}_i$ operates as follows: First compute the median of A to find the cluster (3) and delete the corresponding elements from A. Then,

sort the remaining $O(n^{2/3})$ elements of A. This partitions A into the $n^{1/3}$ clusters of type (1). Among these, find the cluster of A-size i. Pick an arbitrary element from this cluster, and compare it to all elements in B to find the set intersection. We have $\text{Universal}(\mathcal{A}_i, G_i) \in O(n + n^{2/3} \log n) = O(n)$.

Conversely, let $\mathcal{A}$ be any algorithm. Without loss of generality, suppose $n^{1/3} = 2^\ell$ for some integer ℓ. We consider an adaptive adversary that uses a modified version of the binary tree strategy: Consider a complete binary tree T of height $n \log n$. The elements of B all start out at the root of T. The elements of A all start at a leaf of T: The elements in the cluster (3) start out at the leftmost leaf of T. (These elements are smaller than all other elements, and thus irrelevant for the rest of the proof.) For each cluster of type (1), pick a different node u of depth ℓ. The A-elements in this cluster start at the same arbitrary leaf in the subtree of u. The adversary answers comparisons by moving elements in B down the tree, as described in Section 4.

We play the adversarial game with $\mathcal{A}$ for $\frac{1}{2}n \cdot \ell = \frac{1}{6}n \log n$ rounds. Then, there is an element $x \in B$ that is at a tree node of depth $\leq \ell$. In particular, in the subtree of x, there is a leaf that contains a cluster of type (1), say the one with index j. We add x to this cluster, and turn all other elements of B into clusters of size 1. This defines a graph G on X with $G \cong G_j$, where each cluster consists of elements on the same root-to-leaf path of T. Thus, there is an input (A, B) with $G(A, B) \cong G_j$ such that the answers given by the adversary are consistent with (A, B). It follows that $\mathcal{A}$ did not find a set intersection after $\frac{1}{6}n \log n$ comparisons, even though there is one. Therefore, $\text{Universal}(\mathcal{A}, G_j) \geq \frac{1}{6}n \log n$. We put $\mathcal{A}' = \mathcal{A}_j$, then $\text{Universal}(\mathcal{A}', G_j) \in O(n)$ as noted earlier. This shows the universal optimality part of Lemma.

Finally, we show the bounds on the order running time. Put $\overrightarrow{G} = \overrightarrow{G}(A, B)$. Then, $\text{Order}(\mathcal{A}, \overrightarrow{G}) \in \Omega(n \log n)$ since $\mathcal{A}$ does at least $\frac{1}{6}n \log n$ comparisons on (A, B). On the other hand, $\text{Order}(\mathcal{A}', \overrightarrow{G}) \leq \text{Universal}(\mathcal{A}', \overrightarrow{G}) \in O(n)$, since we take the maximum over a smaller subset of inputs. This shows the Lemma. ◀