

A sort of an adversary

Haim Kaplan *

Or Zamir †

Uri Zwick †

Abstract

We describe an *efficient* deterministic adversary that forces any comparison-based sorting algorithm to perform at least $\frac{37}{64} n \log n$ comparisons. This improves on previous efficient adversaries of Atallah and Kosaraju (1981), Richards and Vaidya (1988), and of Brodal et al. (1996) that force any sorting algorithm to perform at least $\frac{1}{2} n \log n$ comparisons.

1 Introduction

It is well-known that n items from a totally ordered domain can be sorted using at most $n \log n$ comparisons. (Throughout the paper, $\log n = \log_2 n$.) Binary insertion sort, for example, performs $\sum_{k=1}^n \lceil \log k \rceil = n \lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1$ comparisons (see, e.g., Knuth [6]). A sorting algorithm of Ford and Johnson [5] performs at most $\sum_{k=1}^n \lceil \log \frac{3}{4} k \rceil$ comparisons.

It is also well-known that any comparison-based sorting algorithm must perform at least $\log n! = n \log n - \frac{n}{\ln 2} + o(n)$ comparisons on at least some instances. This follows from the fact that the behavior of any comparison-based sorting algorithm can be described using a *comparison tree* which, in the case of sorting, must have at least $n!$ leaves. As comparison trees are binary, there must be at least one leaf at depth at least $\log n!$. This leaf corresponds to an instance on which the algorithm performs at least $\log n!$ comparisons. Finding such an instance, however, may involve explicitly constructing the comparison tree which is of exponential size.

An alternative approach to proving lower bounds for sorting, and other comparison problems, is using an *adversary*. A comparison-based algorithm is an algorithm that obtains information about the input instance only by comparing pairs of input items. An

input instance can thus be replaced by an adversary that answers, in a consistent way, the comparisons made by the algorithm.

The $\log n!$ lower bound for sorting can also be obtained using an adversary argument as follows. The adversary maintains a *partial order* P on the n input items. Initially this partial order is empty, i.e., all the n items are incomparable. When the algorithm issues a comparison $a : b$, the adversary checks whether $a <_P b$, or whether $b <_P a$. If so, it returns the appropriate answer. If a and b are incomparable in P , the adversary computes the number of *linear extensions* of the two partial orders $P \cup \{a < b\}$ and $P \cup \{b < a\}$ and gives the answer that corresponds to the partial order with a larger number of linear extensions and updates P accordingly. Each comparison made by the algorithm decreases the number of linear extensions of P by a factor of at most 2. The algorithm learns the sorted order of the input items only when a single linear extension is left. As there are initially $n!$ linear extensions of P , this adversary forces any comparison-based sorting algorithm to perform at least $\log n!$ comparisons. The adversary does not need to know anything about the sorting algorithm it is facing.

The above adversary, however, is not *efficient*, i.e., it does not run in *polynomial time*. Before answering a comparison it must compute the number of linear extensions of a given partial order, a #P-complete problem (Brightwell and Winkler [2]). To obtain a lower bound of $(1 - \epsilon) \log n!$, for some $\epsilon > 0$, it is enough to *approximate* the number of linear extensions to within some small enough multiplicative factor. However, it is also not known how to do that *deterministically* in polynomial time. There is, however, a fully polynomial time randomized approximation scheme (FPRAS) for the number of linear extensions. This follows from the fact that the number of linear extensions of a partial order P is related to the *volume* of an appropriately defined polytope, and there is a FPRAS for the volume of convex bodies (see, e.g., Dyer et al. [4]). This gives, for every $\epsilon > 0$, a polynomial-time *randomized* adversary that forces any comparison-based sorting algorithm to perform $(1 - \epsilon)n \log n$ comparisons, with high probability.

If we are willing to settle for a randomized ad-

*Blavatnik School of Computer Science, Tel Aviv University, Israel. E-mail: haimk@post.tau.ac.il. Research supported by ISF grant no. 1841-14 and by the Len Blavatnik and the Blavatnik Family foundation.

†Blavatnik School of Computer Science, Tel Aviv University, Israel. E-mail: orzamir@mail.tau.ac.il, zwick@tau.ac.il. Work partly done during a research visit to Copenhagen supported by Thorup's Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation. This work was carried out in partial fulfillment of the requirements for the Ph.D. degree of Or Zamir.

versary, there is a much simpler solution. The adversary simply chooses a *random permutation* of the input items, orders the items according to this permutation and answers the queries of the algorithm accordingly. This will force any sorting algorithm to perform at least $n \log n - O(n)$ comparisons, with high probability.

Is there, for every $\epsilon > 0$, an efficient *deterministic* adversary that forces any sorting algorithm to perform $(1-\epsilon)n \log n$ comparisons? This fundamental problem is still open. Atallah and Kosaraju [1] obtained an efficient deterministic adversary that forces any algorithm to perform $\frac{1}{2}n \log n$ comparisons. Two additional efficient deterministic adversaries that achieve the same result were obtained by Richards and Vaidya [7] and Brodal et al. [3].

The adversary of Brodal et al. [3], which may be viewed as a refinement of the adversary of Atallah and Kosaraju [1], and which we review in Section 2, is especially appealing. It keeps for each item a an open interval $I(a)$ of the form $(\frac{j}{2^i}, \frac{j+1}{2^i})$, for some integers $i \geq 0$ and $0 \leq j < 2^i$. Initially, $I(a) = (0, 1)$ for each item a . It follows that for every two items a and b , either $I(a) \cap I(b) = \emptyset$ or $I(a) \subseteq I(b)$ or $I(b) \subseteq I(a)$. Furthermore, if $I(a) \subseteq I(b)$, $I(a) = (\alpha_0, \alpha_1)$ and $I(b) = (\beta_0, \beta_1)$, then either $I(a) = I(b)$ or $I(a) \subseteq (\beta_0, \frac{\beta_0+\beta_1}{2})$, or $I(a) \subseteq (\frac{\beta_0+\beta_1}{2}, \beta_1)$. A similar condition holds if $I(b) \subseteq I(a)$. The adversary answers a comparison between items a and b with intervals $I(a) = (\alpha_0, \alpha_1)$ and $I(b) = (\beta_0, \beta_1)$ as follows. If $I(a) < I(b)$, i.e., $\alpha_1 < \beta_0$, it answers $a < b$. If $I(b) < I(a)$, i.e., $\beta_1 \leq \alpha_0$, it answers $b < a$. If $I(a) = I(b)$, it answers $a < b$ and sets $I(a) = (\alpha_0, \frac{\alpha_0+\alpha_1}{2})$ and $I(b) = (\frac{\alpha_0+\alpha_1}{2}, \alpha_1)$. If $I(a) \subseteq (\beta_0, \frac{\beta_0+\beta_1}{2})$, it answers $a < b$, leaves $I(a)$ unchanged and sets $I(b) = (\frac{\beta_0+\beta_1}{2}, \beta_1)$. If $I(a) \subseteq (\frac{\beta_0+\beta_1}{2}, \beta_1)$, it answers $b < a$, leaves $I(a)$ unchanged and sets $I(b) = (\beta_0, \frac{\beta_0+\beta_1}{2})$. The cases $I(b) \subseteq (\alpha_0, \frac{\alpha_0+\alpha_1}{2})$ and $I(b) \subseteq (\frac{\alpha_0+\alpha_1}{2}, \alpha_1)$ are handled symmetrically.

The adversary of Brodal et al. [3] is certainly efficient. It is not difficult to show that it forces any comparison-based sorting algorithm to perform at least $\frac{1}{2}n \log n$ comparisons. The way the adversary of Brodal et al. [3] is described above is equivalent, but not identical to its description in [3]. In their formulation, the items are kept in nodes of a (potentially infinite) binary tree. Each node corresponds to a *dyadic* interval $(\frac{j}{2^i}, \frac{j+1}{2^i})$. The two children of such a node are the subintervals $(\frac{2j}{2^{i+1}}, \frac{2j+1}{2^{i+1}})$ and $(\frac{2j+1}{2^{i+1}}, \frac{2j+2}{2^{i+1}})$. The node corresponding to the interval $(\frac{j}{2^i}, \frac{j+1}{2^i})$ is thus at level i of the tree, and if $0.j_1j_2 \dots j_i$ is the binary representation of $\frac{j}{2^i}$, then $j_1j_2 \dots j_i$ describe the i steps made from the root to get to this node, with 0 denoting a left turn and 1 denoting a right turn.

We obtain the first improvement over the $\frac{1}{2}n \log n$ bound of Atallah and Kosaraju [1], Richards and Vaidya [7] and Brodal et al. [3]. Namely, we describe a simple and efficient deterministic adversary that forces any sorting algorithm to perform at least $\frac{37}{64}n \log n \geq 0.578n \log n$ comparisons.

We start by describing an adversary that forces $0.55 \cdot n \log n$ comparisons. This adversary, with hindsight, can be seen as a natural generalization of the adversary of Brodal et al. [3]. We keep for each item a a 3-adic sub-interval of $(0, 1)$ of the form $(\frac{j}{3^i}, \frac{j+1}{3^i})$ or $(\frac{k}{3^i}, \frac{k+2}{3^i})$ where $k \not\equiv 2 \pmod{3}$. This, as we shall see, corresponds to keeping the items in a tree-like dag (directed acyclic graph) of outdegree 2 which emulates an infinite ternary tree. The details and the analysis, however, are more complicated and require additional ideas.

We obtain our $\frac{37}{64}n \log n$ lower bound using 4-adic intervals, i.e., sub-intervals of $(0, 1)$ of the form $(\frac{j}{4^i}, \frac{j+1}{4^i})$, $(\frac{j}{4^i}, \frac{j+2}{4^i})$ or $(\frac{j}{4^i}, \frac{j+3}{4^i})$. The details get yet more complicated, with several additional issues arising.

It is natural to speculate that by using k -adic intervals, for increasing values of k , we may get better and better adversaries forcing more comparisons and getting closer to the desired $n \log n$ lower bound. Unfortunately, it seems that this is *not* the case. We show that natural extensions of our techniques for $k \geq 5$ do not improve over the result we get with $k = 4$, so some additional new ideas are needed.

The rest of the paper is organized as follows. In Section 2 we review the adversary of Brodal et al. [3]. In Section 3 we obtain our first improved adversary using $k = 3$. In Section 5 we describe our best adversary obtained using $k = 4$. In Section 6 we explain why using $k \geq 5$ does not yield improved results, without the use of additional new techniques, and we discuss further possible lines of research and open problems.

2 The classical binary adversary

We begin by describing a simple adversary due to Brodal et al. [3] giving a lower bound of $\frac{1}{2} \cdot n \log n$.

We maintain the items to be sorted at the nodes of an infinite complete binary tree T . Each node of T may contain several items. A node of T is *occupied* if it contains at least one item, and *empty*, otherwise. Since the number of items is finite, only a finite portion of T is occupied. In what follows we refer by T to the finite subtree of T consisting of all the occupied nodes in T and their ancestors.

We let $v(x)$ denote the node in T containing item x . For a node $u \in T$ we let $child_0(u)$ and $child_1(u)$ denote the left and right children of u , respectively, and we let $T(u)$ denote the set of descendants of u in T ,

including u itself.

The adversary uses T to maintain a partial order $<$ on the items which is consistent with all the answers already given by the adversary. The partial order associated with T is defined as follows. For items x and y residing in the nodes $v(x)$ and $v(y)$ of T , respectively, we say that $x < y$ if and only if $v(x)$ is not an ancestor of $v(y)$, $v(y)$ is not an ancestor of $v(x)$, and $v(x)$ appears before $v(y)$ in the in-order traversal of T .¹

Note that x and y are *comparable* in $<$ if and only if none of $v(x)$ and $v(y)$ is an ancestor of the other. If $v(x)$ is an ancestor of $v(y)$, or vice versa, then x and y are *incomparable* in $<$. In particular, if $v(x) = v(y)$, i.e., x and y reside at the same node, then x and y are incomparable. It is not difficult to check that $<$ is indeed a partial order, i.e., if $x < y$ and $y < z$ then $x < z$.

The adversary responds to a comparison between x and y as follows:

- If $x < y$ or $y < x$ in the partial order, the adversary gives the corresponding response.
- If $v(x) = v(y)$, the adversary moves x to $child_0(v(x))$, moves y to $child_1(v(x))$, and answers that $x < y$.
- If $v(x)$ is an ancestor of $v(y)$, the adversary moves x to the child u of $v(x)$ which is not an ancestor of $v(y)$ and answers $x < y$ if $u = child_0(v(x))$, and $y < x$ if $u = child_1(v(x))$.
- If $v(y)$ is an ancestor of $v(x)$, the adversary moves y to the child u of $v(y)$ which is not an ancestor of $v(x)$ and answers $x < y$ if $u = child_1(v(y))$, and $y < x$ if $u = child_0(v(y))$.

An example showing the behavior of the adversary is given in Figure 1.

We say that T is *sorted* if each item resides in a distinct leaf. A tree T is sorted if and only if its corresponding partial order $<$ is a total order. Since the sum of the depths of the leaves in a binary tree with n leaves is at least $n \log n$,² we get that when sorted, the sum of depths of the n items in the tree is at least $n \log n$. At the beginning, the sum of the depths of all items is 0.

¹The *in-order* traversal of a binary tree T with root r , is defined recursively as the in-order traversal of $T(child_0(r))$, followed by r , followed by the in-order traversal of $T(child_1(r))$.

²By induction, the sum of the depths of the leaves in the left subtree is at least $n_1 \log n_1$ and the sum of the depths of the leaves in the right subtree is at least $n_2 \log n_2$. It follows that the sum of the depths of all leaves is at least $n_1 + n_1 \log n_1 + n_2 + n_2 \log n_2 \geq n + (n_1 + n_2) \log((n_1 + n_2)/2) = n \log n$ by the convexity of the function $f(x) = x \log x$.

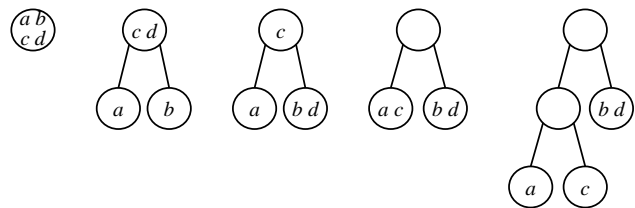


Figure 1: Various states of the tree kept by the adversary. The tree on the left is the tree after the insertion of items a, b, c and d . The following trees are the results of the comparisons $a : b$, $a : d$, $b : c$ and $a : c$.

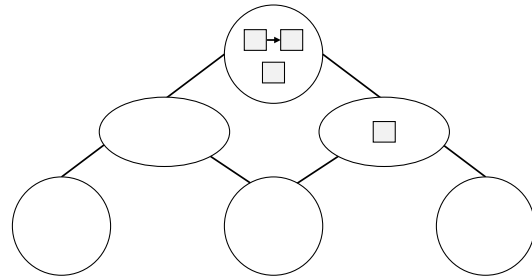


Figure 2: A schematic figure of the ternary adversary.

Each move of the adversary increases the sum of depths by at most 2 (as a response to a comparison). Thus, we need at least $\frac{1}{2} \cdot n \log n$ comparisons to sort. This proves the following theorem.

THEOREM 2.1. *The adversary described above forces any sorting algorithm to make $\frac{1}{2} \cdot n \log n$ comparisons in order to sort n items.*

3 Improving on the adversary

Consider the adversary of Section 2 when we sort four items a, b, c and d . Suppose the algorithm compares a and b and the adversary answers $a < b$, and then the algorithm compares c and d and the adversary answers $c < d$. The weakness of this adversary is that the algorithm now knows, without further comparisons, that $a < d$ and $c < b$. In fact, there is an algorithm that sorts with $\frac{1}{2} \cdot n \log n$ comparisons against this adversary: We partition the items into pairs, compare each pair, the results of these comparisons partition the set into the $n/2$ smallest and $n/2$ largest, and we continue recursively within each subset.

To improve upon this adversary we begin by making the tree T ternary. This alone is not enough, obviously, as still when two items are compared at a node they both have to be moved. For a node $u \in T$ we let $child_0(u)$, $child_1(u)$ and $child_2(u)$ denote the *left*, *middle* and *right* children of u , respectively. We then further consider an augmentation of T into a Directed Acyclic

Graph (DAG) T' that contains the nodes of T and in addition, for each node $u \in T$, two *intermediate nodes*, called the *left-middle* and *middle-right* intermediate nodes of u , and denoted by $child_{[0,1]}(u)$ and $child_{[1,2]}(u)$, respectively. In T' we make $child_{[0,1]}(u)$ a (left) child of u and parent of $child_0(u)$ and $child_1(u)$, similarly we make $child_{[1,2]}(u)$ a (right) child of u and a parent of $child_1(u)$ and $child_2(u)$. See Figure 2. We refer to nodes of $T \cap T'$ as *tree nodes*. (The tree nodes are the nodes at even levels at T , where the root is at level 0.)

The adversary maintains the items at the nodes of T' as before but now the tree nodes of T' may also store ordered pairs of related items. Specifically, for a tree node u of T' , we maintain a set $I(u)$ of (unpaired) items which are currently stored at u , and also a set $P(u)$ of ordered pairs which are currently stored at u . For an intermediate node u of T' , we maintain only the set $I(u)$ of items currently stored at u . We say that an item x is *contained* at a node u if $x \in I(u) \cup \{a \mid (a, b) \in P(u)\} \cup \{b \mid (a, b) \in P(u)\}$.

For an item x , we let $v(x)$ denote the node (tree or intermediate) of T' that currently contains x . We let $\bar{v}(x)$ be $v(x)$ if $v(x)$ is a tree node, or the tree node u for which $v(x)$ is $child_{[0,1]}(u)$ or $child_{[1,2]}(u)$ in case $v(x)$ is an intermediate node. We also denote by $T(u)$ the set of descendants of u in T' , including u itself.

The adversary starts with all items at the root of T and it uses T to maintain a partial order $<$ on the items that contains the partial order that it is already committed to. The partial order $<$ is defined as follows.

DEFINITION 1. (PARTIAL ORDER) *Let x, y be items residing in nodes $v(x), v(y)$ of T' . Let w be the lowest common ancestor in T of $\bar{v}(x), \bar{v}(y)$. We say that $x < y$ if and only if one of the following cases hold,*

- *There exists $i < j$ such that $x \in T'(child_i(w))$ and $y \in T'(child_j(w))$.*
- *x is in $T'(child_0(w))$ and y is in $child_{[1,2]}(w)$.*
- *x is in $child_{[0,1]}(w)$ and y is in $T'(child_2(w))$.*
- *$v(x) = v(y) = w$ and $(x, y) \in P(w)$.*

The adversary responds to a comparison between x and y as described in the following table (symmetric cases are omitted). We list first all cases of comparisons of two single items, then cases of comparisons between an item in a pair to an item not in a pair, and finally comparisons between two items that are in pairs.

The *depth* $d(x)$ of an item x is defined to be the depth of $\bar{v}(x)$ in T . The adversary tries to minimize the number of items whose depth increases following a comparison. Intuitively, the ability to pair items in

tree nodes and temporarily store items at intermediate nodes (when it knows that the item should be placed in one of two consecutive children of a tree node, but keeps the freedom to decide in which one), allows the adversary to delay the depth-increase of an item.

Since items may increase their depth gradually while temporarily residing in intermediate nodes and/or paired with other items while residing at tree nodes we define a potential $\Phi = \Phi(T')$ which can be viewed as a refined notion of the sum of the depths of the items. In particular, when all items are at tree nodes and there are no pairs then the potential is the sum of the depths of the items.

The definition of Φ is as follows. An item contained in a tree node contributes its depth to Φ . An item contained in an intermediate node contributes its depth plus some constant m to Φ , and each pair adds a constant p to Φ . (We fix the constants p and m later.) Specifically, the potential is

$$\begin{aligned} \Phi &= \sum_{\text{item } x} d(x) \\ &+ m \cdot \#\{x \mid x \text{ is in an intermediate node}\} \\ &+ \frac{p}{2} \cdot \#\{x \mid x \text{ is in a compared pair}\}. \end{aligned}$$

We define $\Delta(\phi)$ to be maximal increase in the potential of T following a response of the adversary to a single comparison.

THEOREM 3.1. *The adversary described in this section forces any sorting algorithm to perform at least $\frac{1}{\Delta(\phi) \cdot \log 3} \cdot n \log n$ comparisons in order to sort n items.*

Proof. When sorted, $\bar{v}(x)$ is unrelated to $\bar{v}(y)$ for every pair of items x and y . Therefore, the sum of depths of the n items in T' is at least $n \log_3 n$ and the potential is also at least $n \log_3 n$.³ Since, at the beginning, Φ is 0 and Φ increases at most by $\Delta(\phi)$ after each comparison, the theorem follows. \square

For $p = \frac{8}{7}$ and $m = \frac{3}{7}$,⁴ we get that $\Delta(\phi)$ is $8/7$ as, verified in the top rows of each cell in Tables 1-3. Thus, the following theorem follows from Theorem 3.1.

THEOREM 3.2. *The adversary described in this section can force any sorting algorithm to perform at least $\frac{7}{8 \log 3} \cdot n \log n \approx 0.5521 \cdot n \log n$ comparisons in order to sort n items.*

³A generalized claim is proved later in Lemma 4.1.

⁴These values of p and m are the unique solution to the linear system $\{\Delta = p, \Delta = 3 + m - 2p, \Delta = 2 - 2m\}$ which represents the changes in the potential for a set of comparisons we “guessed” should be the hardest.

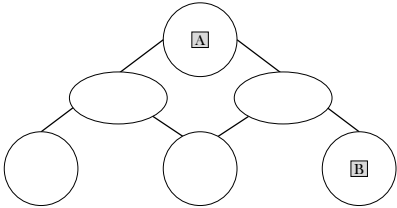
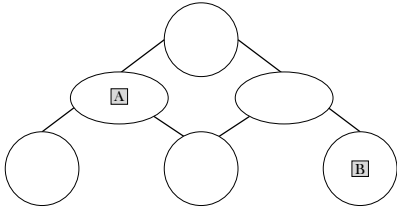
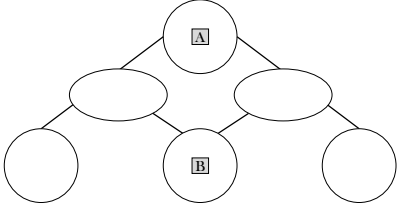
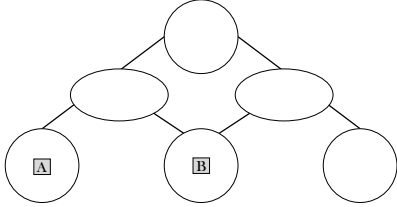
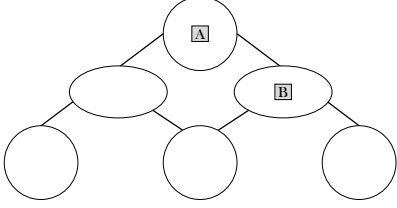
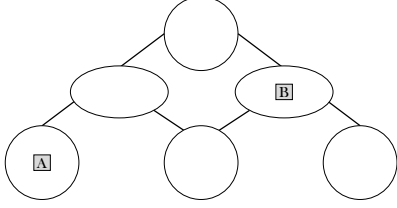
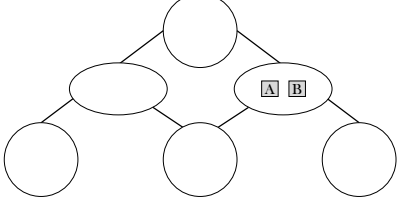
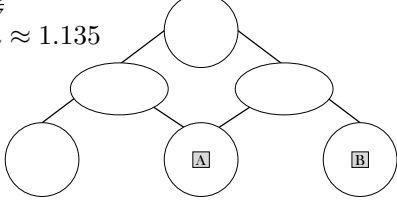
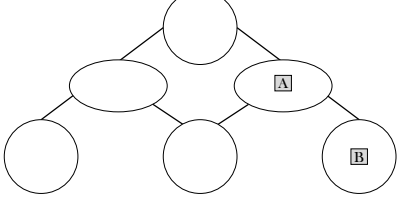
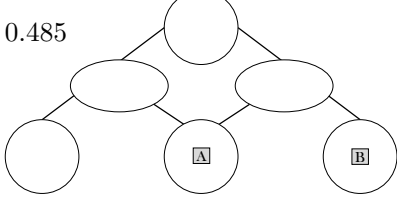
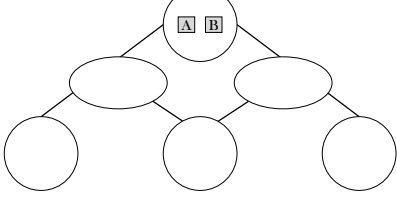
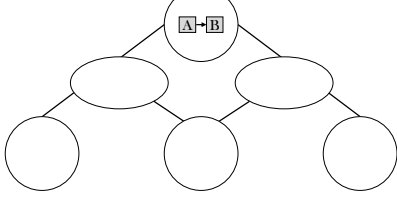
	$m = \frac{3}{7}$ $m \approx 0.404$	
	1 $w \approx 1.055$	
	1 $w \approx 1.055$	
	$2 - 2m = \frac{8}{7}$ $w + (3 - 2w) - 2m \approx 1.135$	
	$1 - m = \frac{4}{7}$ $(3 - 2w) - m \approx 0.485$	
	$p = \frac{8}{7}$ $p \approx 1.135$	

Table 1: Cases 1-6 of the adversary for $k = 3$. The expressions in the middle are the increases in potentials and weighted potentials.

	$2 - p = \frac{6}{7}$ $(3 - 2w) + w - p \approx 0.81$	
	$1 + m - p = \frac{2}{7}$ $w + m - p \approx 0.325$	
	$1 + m - p = \frac{2}{7}$ $w + m - p \approx 0.325$	
	$1 + m - p = \frac{2}{7}$ $w + m - p \approx 0.325$	
	$2 - p = \frac{6}{7}$ $(3 - 2w) + w - p \approx 0.81$	
	$1 + 2m - p = \frac{5}{7}$ $w + 2m - p \approx 0.73$	

Table 2: Cases 7-12 of the adversary for $k = 3$. Compared items are darker.

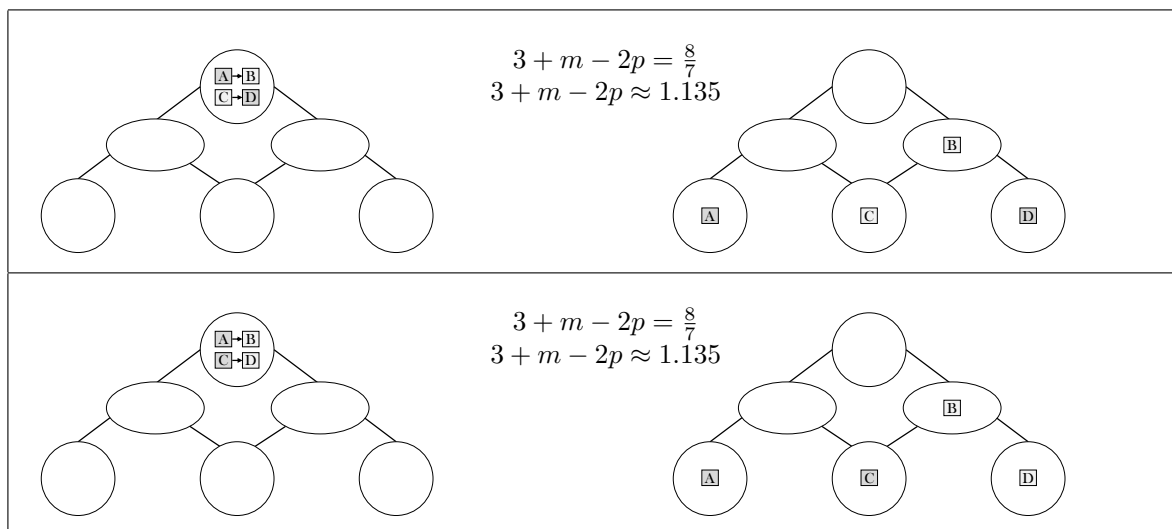


Table 3: Cases 13-14 of the adversary for $k = 3$. Compared items are darker.

The following sequence of comparisons, see Figure 3, indeed causes an average increase of $8/7$ in the sum of the depths of the items per comparison (8 depth increases per 7 comparisons),

- Begin with 8 items placed in the root.
- Make 4 comparisons between pairs of items in the root.
- Pair the pairs of items in the root, and make 2 comparisons, comparing the lower items in each of the pairs.
- Make one more comparison between the two items left in intermediate node(s).

We can repeat this sequence in order to sort n items initially at the root. We group the items into 8-tuples, perform the sequence on each tuple. Then we continue to apply the same sequence on the items in each of the three children of the root.

3.1 Upper bound Can we really sort against the adversary of Section 3 in $\frac{7}{8 \log 3} \cdot n \log n$ comparisons? In other words, is the lower bound of Theorem 3.2 tight for this adversary? The algorithm that sorts by comparing 8-tuples as described at the end of Section 3 indeed makes the adversary increase the depth of $8/7$ items per comparisons, but when it pushes down 8 items at a node u to $child_0(u)$, $child_1(u)$, and $child_2(u)$ by making 7 comparisons, it distributes these 8 items unevenly among $child_0(u)$, $child_1(u)$, and $child_2(u)$. Two of these items end up in $child_0(u)$, three of them in $child_1(u)$ and three of them in $child_2(u)$. As a result, when the items end up in unrelated leaves

the sum of their depths would be larger than $n \log_3 n$ and consequently we will have to perform more than $\frac{7}{8 \log 3} \cdot n \log n$ comparisons.

We can push 8 items down with 7 comparisons while distributing the items more evenly in $child_0(u)$, $child_1(u)$, and $child_2(u)$, by taking 16 items, push one half of them as described above and push the second one down by a symmetric sequence of comparisons such that three end up in $child_0(u)$, three in $child_1(u)$ and two in $child_2(u)$. The affect of both sequences together is that 16 items are pushed down by 14 comparisons such that five end up in $child_0(u)$, six in $child_1(u)$ and five in $child_2(u)$. This algorithm indeed sorts with less comparisons. To upper bound the number of comparisons performed by these algorithms we need the classical concept of entropy defined as follows.

DEFINITION 2. (ENTROPY) Given a distribution $p_1, \dots, p_k \in (0, 1]$ (where $\sum_{i=1}^k p_i = 1$), the entropy of the distribution is defined to be $H(p_1, \dots, p_k) = -\sum_{i=1}^k p_i \log p_i$.

We also need the following technical lemma.

LEMMA 3.1. (SPLITTING RECURSION) The solution of the recurrence relation $T(n) = \lambda n + \sum_{i=1}^k T(p_i n)$, where $\sum_{i=1}^k p_i = 1$, is $T(n) = \frac{\lambda}{H(p_1, \dots, p_k)} n \log n$.

Proof. By induction,

$$\begin{aligned} T(n) &= \lambda \cdot n + \sum_{i=1}^k T(p_i \cdot n) \\ &= \lambda \cdot n + \sum_{i=1}^k \frac{\lambda}{H(p_1, \dots, p_k)} \cdot (p_i \cdot n) \log(p_i \cdot n) \\ &= \lambda \cdot n + \frac{\lambda}{H(p_1, \dots, p_k)} \left(n \log n + n \sum_{i=1}^k p_i \log p_i \right) \\ &= \frac{\lambda}{H(p_1, \dots, p_k)} \cdot n \log n. \end{aligned}$$

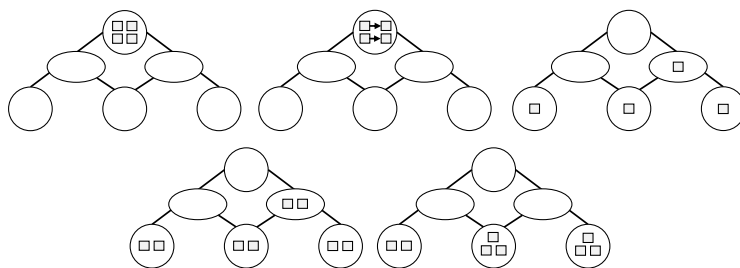


Figure 3: A sequence of comparisons increasing the depth of 8 items in 7 comparisons.

If we denote by $T(n)$ the number of comparisons performed by the algorithm above when sorting n items we get that $T(n)$ matches the recurrence of Lemma 3.1 with $\lambda = 7/8$, $p_1 = 5/16$, $p_2 = 6/16$ and $p_3 = 5/16$. Thus it follows from Lemma 3.1 that the algorithm performs $\frac{7}{H(\frac{5}{16}, \frac{3}{8}, \frac{5}{16})} \cdot n \log n \approx 0.5540 \cdot n \log n$ comparisons.

4 Using the unbalanced distribution

The number of comparisons made by the algorithm of Section 3.1 does not match the lower bound that we proved in Theorem 3.2 for the adversary of Section 3. The reason is that the algorithm cannot distribute the items down evenly, while making 7 comparisons per 8 items. In this section we exploit this to refine our analysis of Section 3 and strengthen Theorem 3.2. Forseeing our subsequent adversaries we give the following definition for general k -ary trees.

We associate weights $\mathbf{w} = (w_1, \dots, w_k)$ with the edges of a k -ary tree T as follows.⁵ For every node u we associate the weight w_i with the edge $(u, \text{child}_i(u))$, for $1 \leq i \leq k$. We define the *weighted depth*, $d_{\mathbf{w}}(c)$, of an item x to be the length of the path (according to the edge weights) from the root of T to $\bar{v}(x)$. The following lemma generalizes the fact that sum of the depths of the leaves in a (unweighted) k -ary tree is at least $n \log_k n$.

LEMMA 4.1. *Consider a weighted k -ary tree T with n leaves and weights $\mathbf{w} = (w_1, \dots, w_k)$. Then the sum of the weighted depths of the leaves of T is at least $\min_{\mathbf{a}} \frac{\langle \mathbf{a}, \mathbf{w} \rangle}{H(\mathbf{a})} n \log n$, where \mathbf{a} is a distribution on k items (i.e., $\mathbf{a} = (a_1, \dots, a_k)$, $a_i > 0$, $\sum a_i = 1$), $\langle \mathbf{a}, \mathbf{w} \rangle$ is the inner product $a_1 \cdot w_1 + \dots + a_k \cdot w_k$, and $H(\mathbf{a})$ is the previously defined entropy.*

⁵The weights are for the analysis only: The adversary does not use them.

□ *Proof.* Let $T(n)$ be the lowest possible such sum of weights. We note that

$$\begin{aligned} T(n) &= \min_{n_1 + \dots + n_k = n} \sum_{i=1}^k (w_i \cdot n_i + T(n_i)) \\ &= \min_{a_1 + \dots + a_k = 1} \sum_{i=1}^k (w_i \cdot a_i \cdot n + T(a_i \cdot n)) \\ &= \min_{a_1 + \dots + a_k = 1} \left(\langle \mathbf{a}, \mathbf{w} \rangle n + \sum_{i=1}^k T(a_i \cdot n) \right) \end{aligned}$$

Then, by induction similar to the one in Lemma 3.1 we get that $T(n) = \min_{\mathbf{a}} \frac{\langle \mathbf{a}, \mathbf{w} \rangle}{H(\mathbf{a})} n \log n$. □

We redefine our potential function to take into account the weighted depths of nodes, as follows

$$\begin{aligned} \Phi &= \sum_{\text{item } x} d_{\mathbf{w}}(x) \\ &+ m \cdot \#\{x \mid x \text{ is in an intermediate node}\} \\ &+ \frac{n}{2} \cdot \#\{x \mid x \text{ is in a compared pair}\}. \end{aligned}$$

We notice that this definition coincides with the previous one for $\mathbf{w} = (1, 1, 1)$. We denote by $\Delta(\phi)$ the maximum increase in the potential function following a comparison. The following theorem is a refinement of Theorem 3.1,

THEOREM 4.1. *Given a potential function ϕ as defined above and with respect to a weighted ternary tree with weights $\mathbf{w} = (w_1, w_2, w_3)$, the adversary of Section 3 forces any algorithm to perform at least $\frac{1}{\Delta(\phi)} \min_{\mathbf{a}} \frac{\langle \mathbf{a}, \mathbf{w} \rangle}{H(\mathbf{a})} n \log n$ comparisons, where the minimum is taken over all possible distributions \mathbf{a} on k items.*

Proof. When items are sorted, the nodes $\bar{v}(x)$ for all items x are unrelated, and therefore by Lemma 4.1, the potential is at least $\min_{\mathbf{a}} \frac{\langle \mathbf{a}, \mathbf{w} \rangle}{H(\mathbf{a})} n \log n$. Since at the beginning, the potential of T is 0, and it increases by at most $\Delta(\phi)$ following any comparison, the theorem follows. □

It is left to find an optimal weight vector $\mathbf{w} = (w_1, w_2, w_3)$. We note that it is natural to assume symmetry and without loss of generality also assume that $w_1 + w_2 + w_3 = 3$, thus we can write $\mathbf{w} = (w, 3 - 2w, w)$ for some $w \in [0, 1.5]$. It is also clear that if w is symmetric then the distribution \mathbf{a} that minimizes $\frac{\langle \mathbf{a}, \mathbf{w} \rangle}{H(\mathbf{a})} n \log n$ is symmetric as well, and therefore can be written as $\mathbf{a} = (a, 1 - 2a, a)$ for some $a \in [0, 0.5]$. Consequently we can write

$$\begin{aligned} \frac{\langle \mathbf{a}, \mathbf{w} \rangle}{H(\mathbf{a})} &= \frac{2 \cdot a \cdot w + (1 - 2a) \cdot (3 - 2w)}{-2 \cdot a \log a - (1 - 2a) \log(1 - 2a)} \\ &= \frac{(6a - 2) \cdot w + 3 - 6a}{-2 \cdot a \log a - (1 - 2a) \log(1 - 2a)}, \end{aligned}$$

and note that

$$\frac{\partial}{\partial a} (2 \cdot a \cdot w + (1 - 2a) \cdot (3 - 2w)) = 6w - 6$$

$$\frac{\partial}{\partial a} (-2a \log a - (1 - 2a) \log(1 - 2a)) = 2 \log(1 - 2a) - 2 \log a$$

Therefore, the a which minimizes $\frac{\langle \mathbf{a}, \mathbf{w} \rangle}{H(\mathbf{a})}$ satisfies that

$$\begin{aligned} &((6a - 2) \cdot w + 3 - 6a) \cdot (2 \log(1 - 2a) - 2 \log a) \\ &= (6w - 6) \cdot (-2 \cdot a \log a - (1 - 2a) \log(1 - 2a)). \end{aligned}$$

In the sequence we assume to be the hardest, the distribution of items corresponds to $a = \frac{5}{16}$. Plugging this value into the previous equation and solving for w gives

$$w = \frac{12 - 3 \log 5}{11 - \log 3 - 2 \log 5} \approx 1.055.$$

This gives us a good candidate for weights vector to try. Fixing that value of w we note that

$$\langle \mathbf{a}, \mathbf{w} \rangle = \frac{1}{8} \cdot (9 - w) \approx 0.993$$

By solving the following system of linear equations

$$\begin{aligned} d &= p \\ d &= 3 + m - 2p \\ d &= w + (3 - 2w) - 2m \end{aligned}$$

we find the unique solution given by $p = \frac{8}{7} \cdot \langle \mathbf{a}, \mathbf{w} \rangle \approx 1.135$, $m = \frac{24}{7} \cdot \langle \mathbf{a}, \mathbf{w} \rangle - 3 \approx 0.404$.

Thus, we now have a concrete potential function that should be optimal for the specific sequence of comparisons we assume to be hardest. It is left to verify that every other comparison does not increase the potential with respect to this function by more than the steps of the aforementioned sequence. This is done in the second row of each cell in Tables 1-3, where we verify that indeed $\Delta(\phi) = \frac{8}{7} \cdot \langle \mathbf{a}, \mathbf{w} \rangle \approx 1.135$. Therefore, this potential function and Theorem 4.1 together, give the following tight lower bound for the ternary adversary.

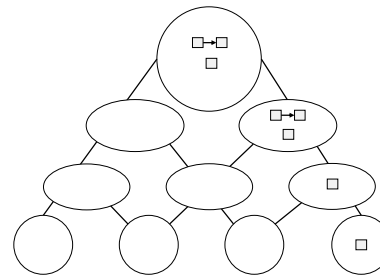


Figure 4: A schematic figure of the 4-ary adversary.

THEOREM 4.2. *The ternary adversary of Section 3 forces any algorithm to perform at least $\frac{7}{H(\frac{5}{16}, \frac{3}{8}, \frac{5}{16})} \cdot n \log n \approx 0.5540 \cdot n \log n$ comparisons in order to sort n items.*

5 Moving to degree four

In this section we describe an adversary that maintains the items in a DAG T' associated with a 4-ary tree T , see Figure 4. This is the direct generalization of the previous section to a larger degree. In T' we have a lattice for each $u \in T$ that contain a node for each “interval” of consecutive children of u . We denote by $child_{[i,j]}(u)$ the node in this lattice that corresponds to the children of indices between i and j for $0 \leq i < j \leq 3$.

In T' , $child_{[0,2]}(u)$ a (left) child of u and parent of $child_{[0,1]}(u)$ and $child_{[1,2]}(u)$, similarly, $child_{[1,3]}(u)$ a (right) child of u and a parent of $child_{[1,2]}(u)$ and $child_{[2,3]}(u)$. Similarly, $child_{[0,1]}(u)$ is a parent of $child_0(u)$ and $child_1(u)$, $child_{[1,2]}(u)$ is a parent of $child_1(u)$ and $child_2(u)$, and $child_{[2,3]}(u)$ is a parent of $child_2(u)$ and $child_3(u)$. As in Section 3, we refer to nodes of $T \cap T'$ as *tree nodes* and to the lattice nodes in $T' \setminus T$ as *intermediate nodes*.

As before, the adversary maintains the items at the nodes of T' where each node may store single items and ordered pairs of items. Intermediate nodes corresponding to intervals of length 2 may store single items only.

The partial order of the adversary is defined as follows. Let x and y be two items residing at the nodes $v(x)$ and $v(y)$ of T' . Let w be the lowest common ancestor in T of $\bar{v}(x)$ and $\bar{v}(y)$. We associate with x an interval as follows. If $v(x)$ is an intermediate node $child_{[i,j]}(w)$ then the interval of x is $[i, j]$, if $v(x)$ is in the subtree $T'(child_i(w))$ then the interval of x is (a single point) $[i, i]$, if $v(x) = w$ then the interval of x is $[0, 3]$. We associate an interval with y analogously. We define that $x < y$ if either the interval of x and the interval of y are disjoint and the interval of x is to the left of the interval of y , or $w = v(x) = v(y)$ and they are related

in the partial order induced by a pair in w .

Table 4 describes the way the adversary handles a few of the possible comparisons, which we intuitively consider to be “hardest”.

The potential function we use is similar to the one of Section 3. We define it to be the sum of the weighted depths of all the items, plus a constant for every item residing in an intermediate node and for each pair at a node. The constant depends on the structure and the type of the intermediate node.

While a priori this might sound sub-optimal given the discussion about the degree 3 case, we consider only potential functions where the weights vector is uniform. Later we would see that this is in fact optimal as the sequence we would prove to be the hardest with respect to this adversary distributes items uniformly between the lattice leaves. Note, for intuition, that this is possible as the degree now is a power of two. Thus, the constants that we add to the sum of the weighted depths are as follows. We add a constant p_0 for every pair in a tree node, a constant m_0 for every singleton item in an intermediate node whose interval is of length 3 (i.e., $child_{[0,2]}(u)$ or $child_{[1,3]}(u)$), a constant p_1 for every pair structure in an intermediate node corresponding to an interval of length 3, a constant m_1 for every singleton item in an intermediate node corresponding to an interval of length 2. We assume that the weights vector is the uniform $(1, 1, 1, 1)$.

We “guess” the parameters (p_0, m_0, p_1, m_1) by minimizing the average increase in potential for some tentative “hardest” sequence of comparisons and then show that this choice of parameters indeed has the same average potential increase per comparison in all cases.

The sequence we consider is the following:

- Begin with 32 items placed in the root.
- Compare 16 disjoint pairs of items creating 16 pairs at the root.
- Pair the 16 pairs in the root, and in each of the 8 pairs of pairs compare the two minimal items. This moves 8 single items into $[0]$, 8 single items into $[1, 3]$ and 8 pairs of items into $[1, 3]$.
- Compare 4 disjoint pairs of single items in $[1, 3]$. This leaves us with 8 items in $[0]$ and 12 pairs in $[1, 3]$.
- Pair the 12 pairs in $[1, 3]$ and do a comparison within each of the 6 pairs of pairs. This time, for 4 out of 6 pairs of pairs, compare the two minimal items. For the 2 others, compare the two maximal items. This leaves us with 8 items in $[0]$, 6 items in each of $[1], [2], [3]$, 2 items in $[1, 2]$ and 4 items in $[2, 3]$.
- Compare the two items in $[1, 2]$ and 2 disjoint pairs

of items in $[2, 3]$. This leaves us with items only in the leaves, spread as 8, 7, 9, 8 respectively.

- Repeat the entire sequence symmetrically, finishing with 16 items in each leaf.

Up to symmetries, all comparisons used in this sequence appear in Table 4.

This sequence uses $2 \cdot 37$ comparisons to increase the depth of $2 \cdot 32$ items, spreading them uniformly among the leaves.

This gives (as in the previous section) an upper bound of $\frac{37}{2} n \log n = 0.578125 n \log n$ for this adversary, we now prove its tightness.

We may deduce a choice for the potential function parameters by solving the following set of linear equations, corresponding to the different types of comparisons done during the aforementioned sequence

- $\Delta = p_0$
- $\Delta = 1 + m_0 + p_1 - 2p_0$
- $\Delta = p_1 - 2m_0$
- $\Delta = 3 + m_1 - 2p_1$
- $\Delta = 2 - 2m_1$

The unique solution for this system is

$$p_0 = \frac{32}{37}, m_0 = \frac{9}{37}, p_1 = \frac{50}{37}, m_1 = \frac{21}{37}$$

A sufficient description of the adversary’s strategy to answer queries would be to simply enumerate over all possible resolutions for the comparison and pick the one that minimizes the increase in this specific potential. In Table 5-16, we verify that $\Delta(\phi) = \frac{32}{37}$ and thus, by Theorem 4.1, we prove a lower bound of $\frac{32}{2} n \log n = 0.578125 n \log n$. Note that in some of the comparison resolutions, we use the fact that pairs of items maintained in the adversary cannot end up in the same leaf. For example, if two items a, b are kept in a pair in the node corresponding to $[0, 2]$ and an item c in $[2, 3]$, we can already tell that $a < c$ as a may end up only in $[0, 1]$ while b only in $[1, 2]$. In fact, pairs of items in the adversary may be viewed as pairs being kept in adjacent siblings instead of inside a single node.

6 Degree five and beyond

Continuing with the discussed approach, it seems reasonable to consider higher degrees. For general $k \geq 2$, we may consider k -ary trees, with intermediate nodes corresponding to every integral sub-interval of $[0, k - 1]$, maintaining single items and pairs of items in the nodes.

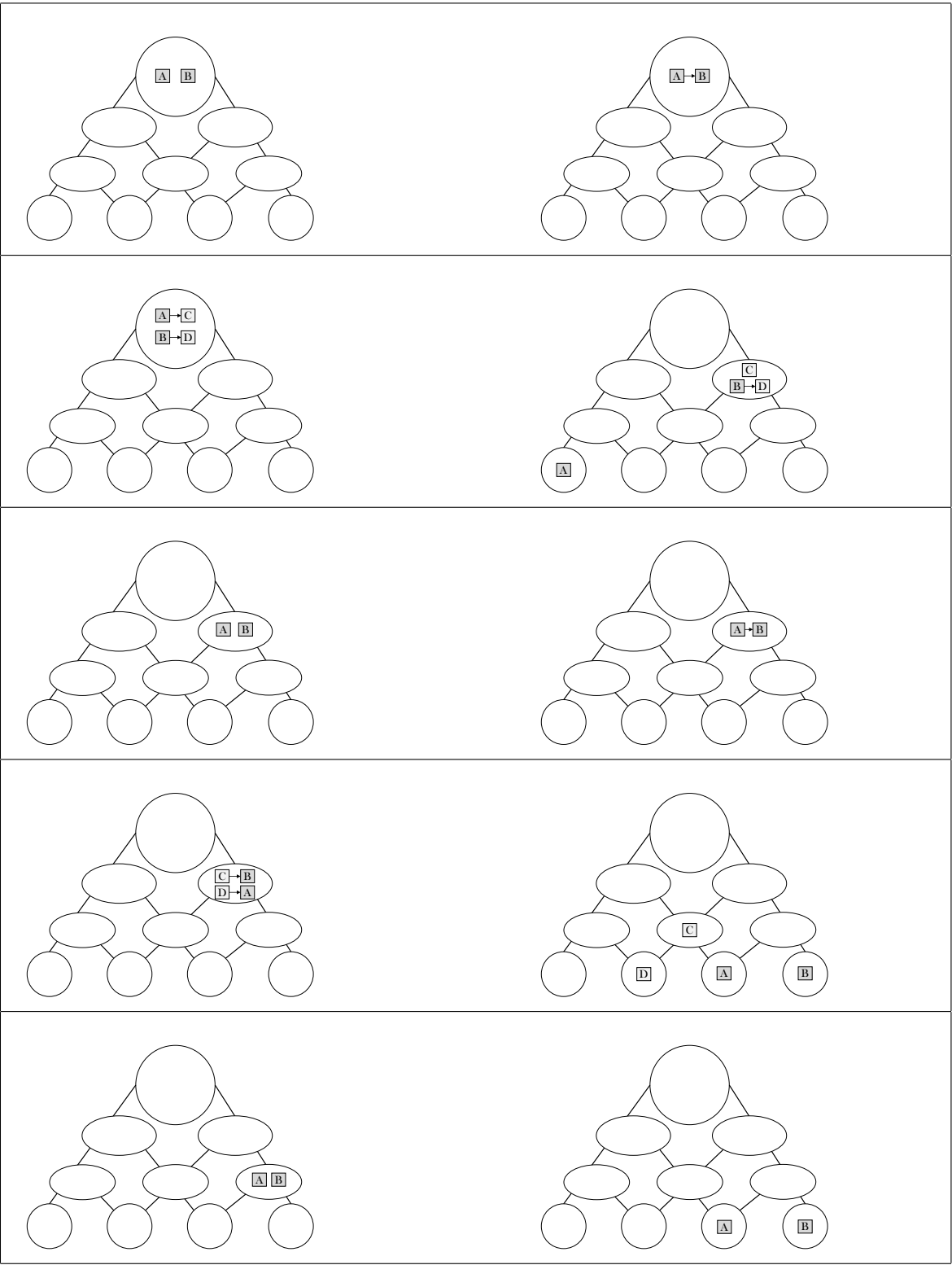


Table 4: The types of comparisons used in the presumed “hardest” sequence, and the way the adversary handles them.

Building on the approach presented in this paper, the general idea would be to “guess” the hardest sequence of comparisons for the adversary (assuming it is usually made of comparing identical structures, as this

way no extra information is revealed). Then, by solving a set of linear equations get a suggestion for a potential function, then verifying its correctness on all other comparisons.

Unfortunately, even for $k = 5$ this approach fails. The reason for that, is that while the solution on the assumed hardest sequence gets better, some other comparisons appear as too hard for the potential function we get.

At the end of the previous section, we discuss viewing pairs as something that does not happen within a lattice node, but between two of them. From that point of view, keeping only pairs between nodes that are adjacent at the same level, does not seem necessary or optimal. It may be that the right way to resolve the comparisons that appear as too hard in the $k = 5$ case is by introducing a more generalized notion of pairs.

The problem now, though, is that “guessing” the potential function becomes harder - as some of the possible structures do not even appear in the sequence that we assume is the hardest. While it may still be possible computationally to find the best potential, it is very technical and probably would not generalize well to higher degrees without a deeper understanding of the structure of these potential functions.

Thus, one of the main open problems arising from this paper is whether we can find a natural way to generalize our technique for larger degrees?

An important observation is that our adversaries can be described in terms of discrete intervals (and disjoint pairs of intervals), as we did for the simple adversary of Brodal in the introduction. As k gets larger, our intervals get more general and the adversary approaches an adversary that maintains continuous sub-intervals of $[0, 1]$ (and disjoint pairs of these intervals). It is plausible that the “right” descriptions of our adversaries do not go through the concretization that the lattices offer, but directly use some representation by continuous intervals.

The prime suspect for the lower bound that we would be able to get using such approach is $\frac{1}{2H(\frac{1}{4}, \frac{3}{4})} n \log n \approx 0.616311 \cdot n \log n$. The reasoning is as follows. Assume we have four items with identical intervals (equivalently, appearing in the same lattice node). Without loss of generality, this interval is $[0, 1]$. By comparing two disjoint pairs of them, and then comparing the minimal items in both pairs, it seems like the reasonable resolution of the three comparisons by the adversary would be moving one of the items to the interval $[0, \frac{1}{4}]$, and all three others to the interval $[\frac{1}{4}, 1]$ where two of them are paired. If we give the algorithm the option to regret the comparison of the two items that are left paired (in other words, we ignore that pair

but think of it as only two comparisons were made), we see that with $\frac{n}{2}$ comparisons the algorithm managed to split the n items into two ordered buckets of sizes $\frac{n}{4}$ and $\frac{3n}{4}$. Thus, it would be able to recursively sort in time $\frac{1}{2H(\frac{1}{4}, \frac{3}{4})} n \log n$. It is now clear that the same holds even without letting the algorithm “regret” the comparisons, as it does the same comparisons anyway.

To break this $0.616311 \cdot n \log n$ barrier, more complicated structures would have to be considered. For example, instead of only singletons and pairs, we may also consider binomial trees of degree 2 (and later, of higher degrees), either using the lattice representation or an interval representation.

Our results are a proof-of-concept for the idea of reducing the general problem of maintaining a partial order to that of maintaining a partial order where only small subsets of items are dependant in a non-trivial way. The main question left is whether this approach can be used to get an efficient optimal adversary that forces any sorting algorithm to perform $(1 - o(1)) n \log n$ comparisons.

References

- [1] Mikhail J. Atallah and S. Rao Kosaraju. An adversary-based lower bound for sorting. *Information Processing Letters*, 13(2):55–57, 1981.
- [2] Graham Brightwell and Peter Winkler. Counting linear extensions. *Order*, 8(3):225–242, 1991.
- [3] Gerth Stølting Brodal, Shiva Chaudhuri, and Jaikumar Radhakrishnan. The randomized complexity of maintaining the minimum. *Nordic Journal of Computing*, 3(4):337–351, 1996.
- [4] Martin E. Dyer, Alan M. Frieze, and Ravi Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991.
- [5] Lester R. Ford and Selmer M. Johnson. A tournament problem. *The American Mathematical Monthly*, 66(5):387–389, 1959.
- [6] D.E. Knuth. *Sorting and searching*, volume 3 of *The art of computer programming*. Addison-Wesley, second edition, 1998.
- [7] Dana S. Richards and Pravin M. Vaidya. On the distribution of comparisons in sorting algorithms. *BIT*, 28(4):764–774, 1988.

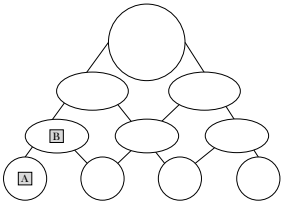
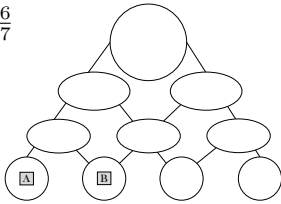
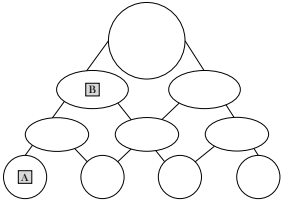
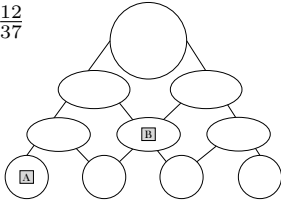
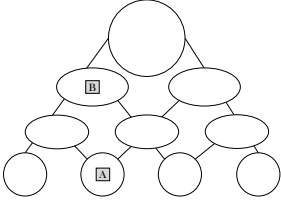
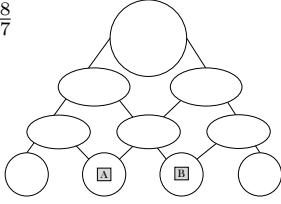
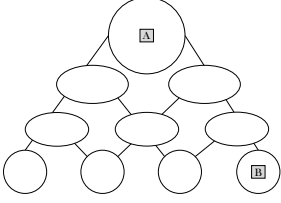
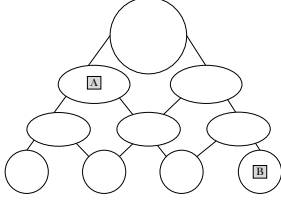
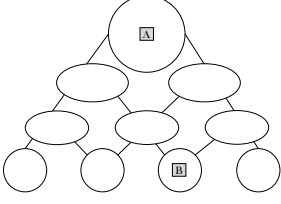
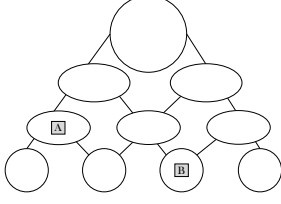
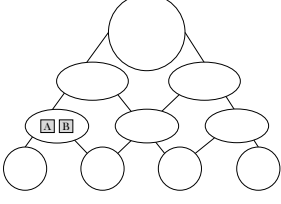
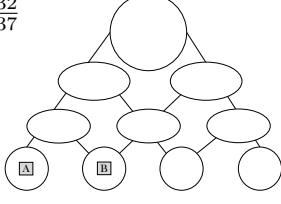
	$1 - m_1 = \frac{16}{37}$	
	$m_1 - m_0 = \frac{12}{37}$	
	$1 - m_0 = \frac{28}{37}$	
	$m_0 = \frac{9}{37}$	
	$m_1 = \frac{21}{37}$	
	$2 - 2m_1 = \frac{32}{37}$	

Table 5: Comparisons of two singletons.

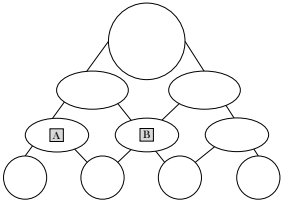
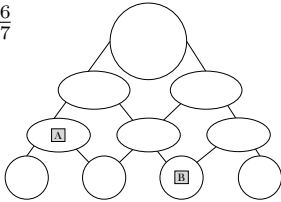
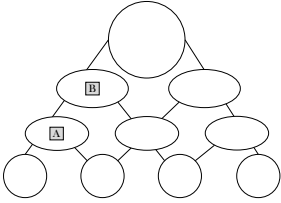
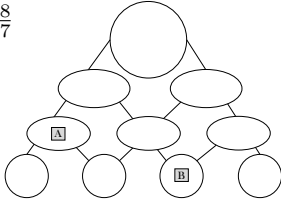
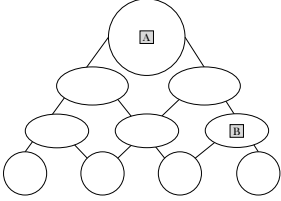
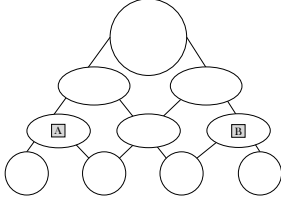
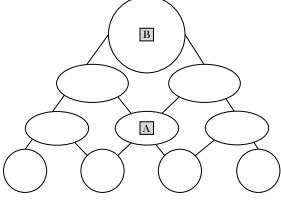
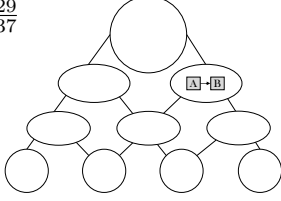
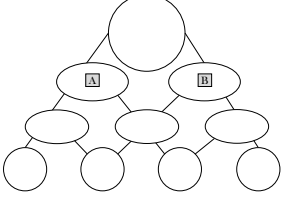
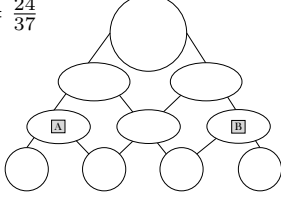
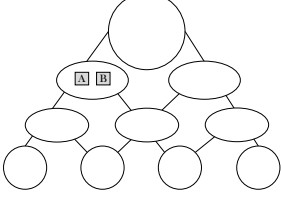
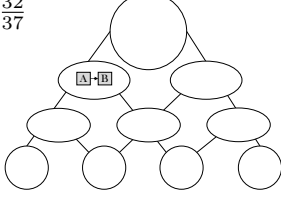
	$1 - m_1 = \frac{16}{37}$	
	$1 - m_0 = \frac{28}{37}$	
	$m_1 = \frac{21}{37}$	
	$p_1 - m_1 = \frac{29}{37}$	
	$2m_1 - 2m_0 = \frac{24}{37}$	
	$p_1 - 2m_0 = \frac{32}{37}$	

Table 6: More comparisons of two singletons

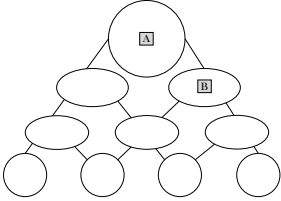
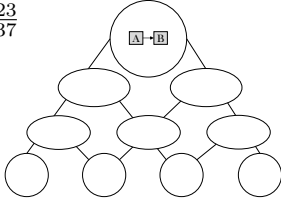
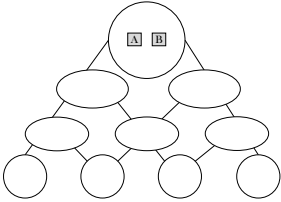
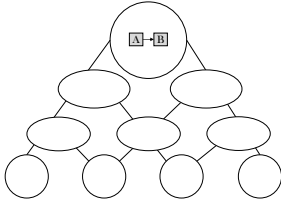
	$p_0 - m_0 = \frac{23}{37}$	
	$p_0 = \frac{32}{37}$	

Table 7: Yet more comparisons of two singletons

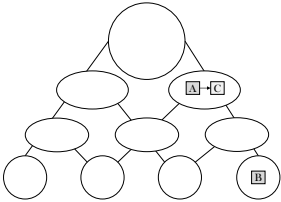
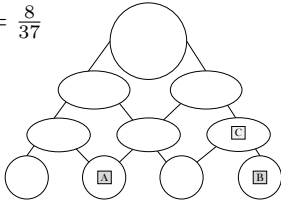
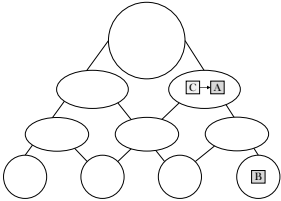
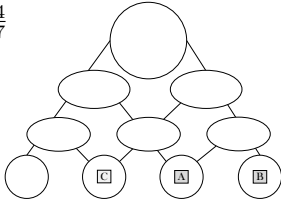
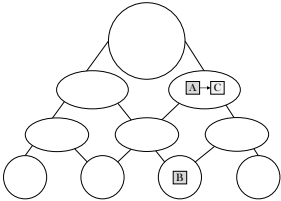
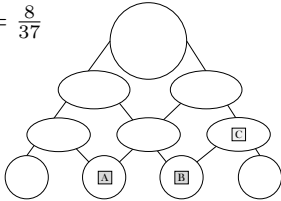
	$1 + m_1 - p_1 = \frac{8}{37}$	
	$2 - p_1 = \frac{24}{37}$	
	$1 + m_1 - p_1 = \frac{8}{37}$	

Table 8: Comparisons involving a lower pair and a singleton

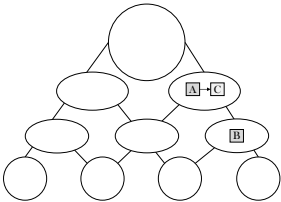
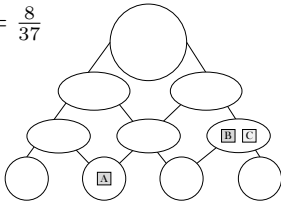
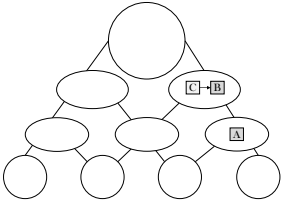
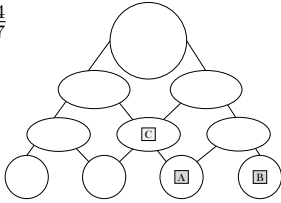
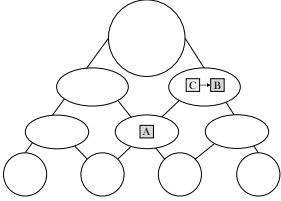
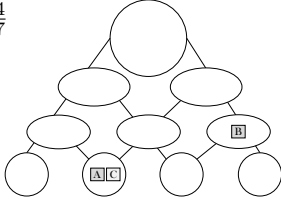
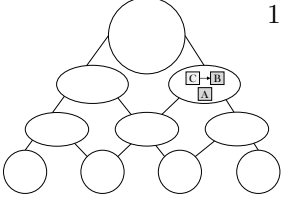
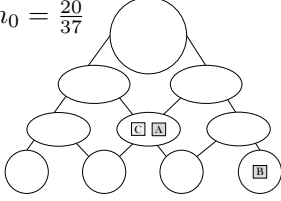
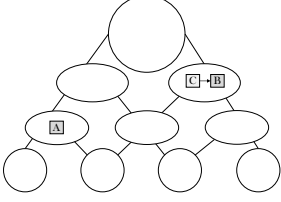
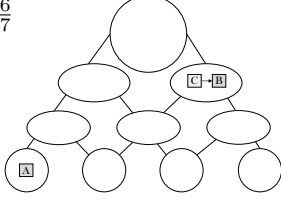
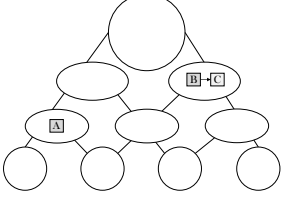
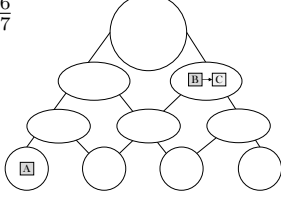
	$1 + m_1 - p_1 = \frac{8}{37}$	
	$2 - p_1 = \frac{24}{37}$	
	$2 - p_1 = \frac{24}{37}$	
	$1 + 2m_1 - p_1 - m_0 = \frac{20}{37}$	
	$1 - m_1 = \frac{16}{37}$	
	$1 - m_1 = \frac{16}{37}$	

Table 9: Comparisons involving a lower pair and a singleton

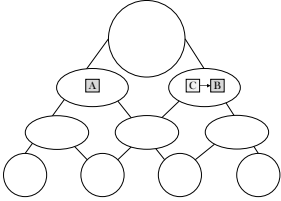
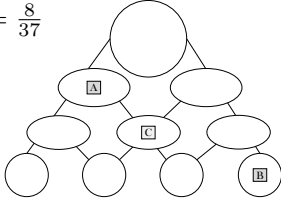
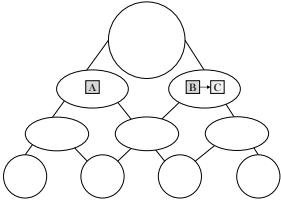
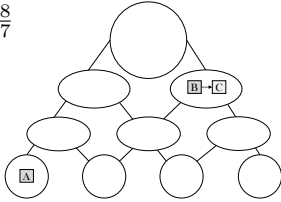
	$1 + m_1 - p_1 = \frac{8}{37}$	
	$1 - m_0 = \frac{28}{37}$	

Table 10: Comparisons involving a lower pair and a singleton

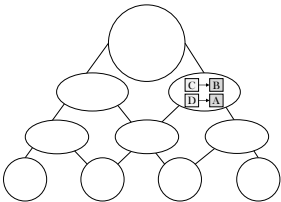
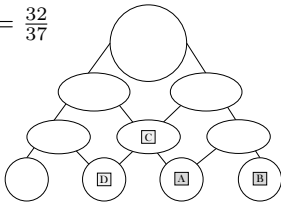
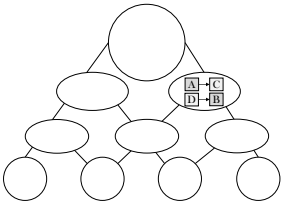
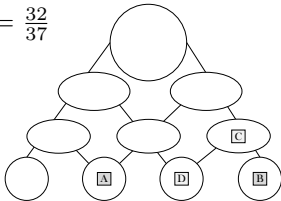
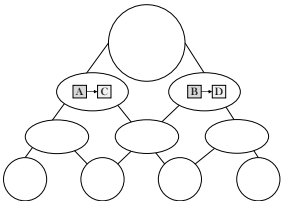
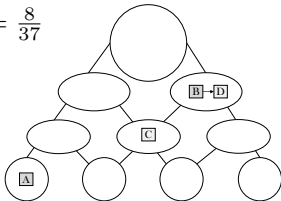
	$3 + m_1 - 2p_1 = \frac{32}{37}$	
	$3 + m_1 - 2p_1 = \frac{32}{37}$	
	$1 + m_1 - p_1 = \frac{8}{37}$	

Table 11: Comparisons involving two lower pairs

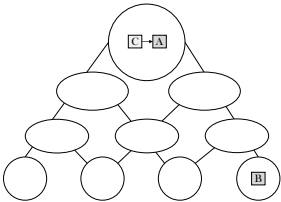
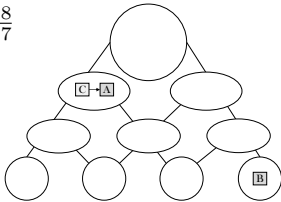
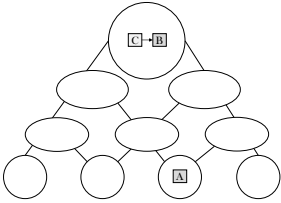
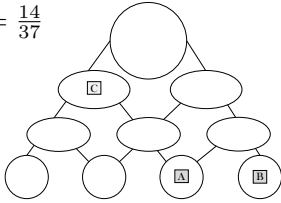
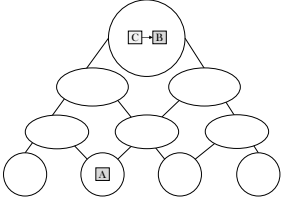
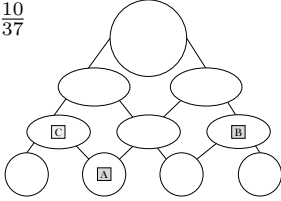
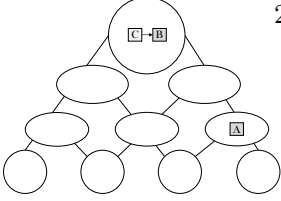
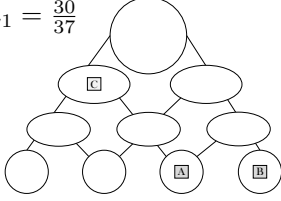
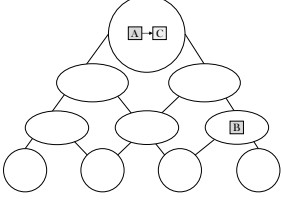
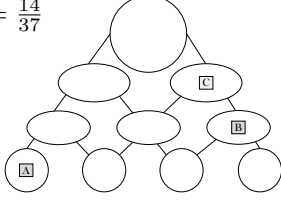
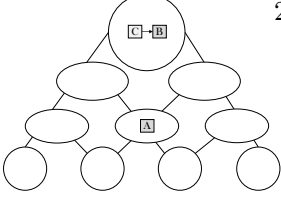
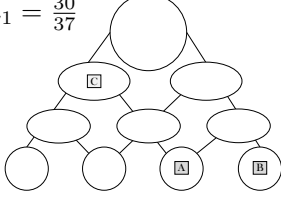
	$p_1 - p_0 = \frac{18}{37}$	
	$1 + m_0 - p_0 = \frac{14}{37}$	
	$2m_1 - p_0 = \frac{10}{37}$	
	$2 + m_0 - p_0 - m_1 = \frac{30}{37}$	
	$1 + m_0 - p_0 = \frac{14}{37}$	
	$2 + m_0 - p_0 - m_1 = \frac{30}{37}$	

Table 12: Comparisons involving an upper pair and a singleton

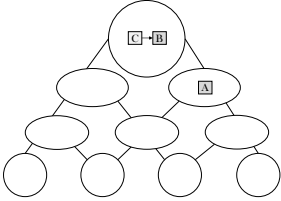
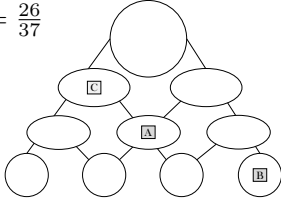
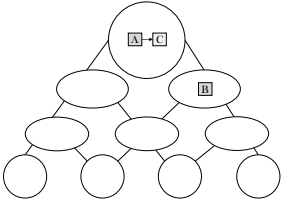
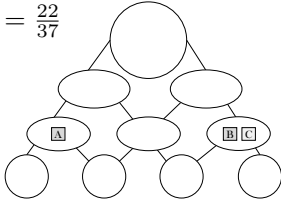
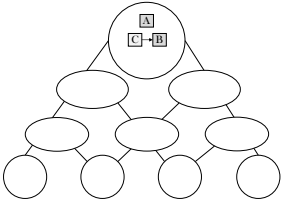
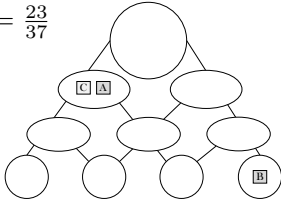
	$1 + m_1 - p_0 = \frac{26}{37}$	
	$3m_1 - p_0 - m_0 = \frac{22}{37}$	
	$1 + 2m_0 - p_0 = \frac{23}{37}$	

Table 13: Comparisons involving an upper pair and a singleton

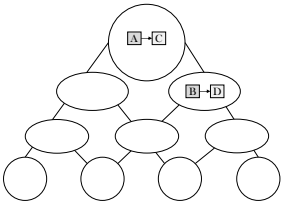
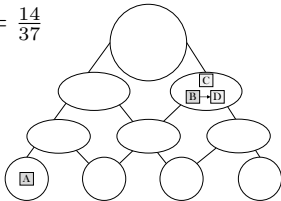
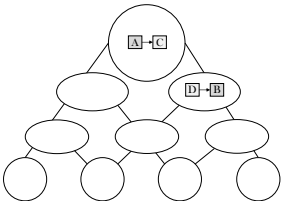
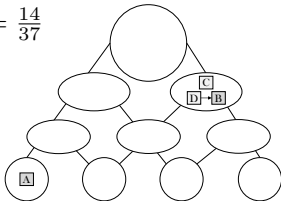
	$1 + m_0 - p_0 = \frac{14}{37}$	
	$1 + m_0 - p_0 = \frac{14}{37}$	

Table 14: Comparisons involving an upper pair and a lower pair

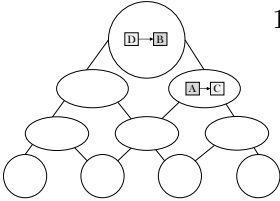
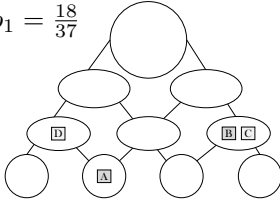
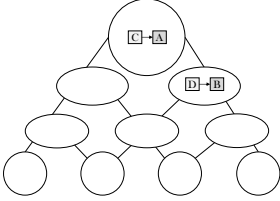
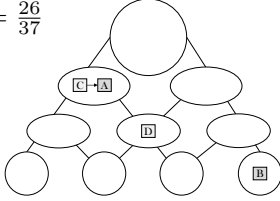
	$1 + 3m_1 - p_0 - p_1 = \frac{18}{37}$	
	$1 + m_1 - p_0 = \frac{26}{37}$	

Table 15: Comparisons involving an upper pair and a lower pair

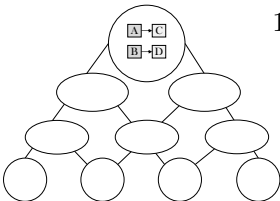
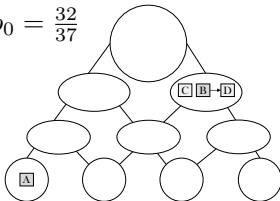
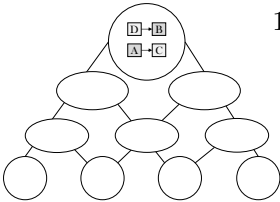
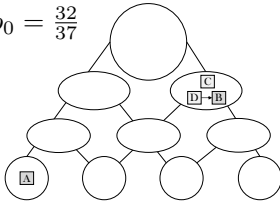
	$1 + m_0 + p_1 - 2p_0 = \frac{32}{37}$	
	$1 + m_0 + p_1 - 2p_0 = \frac{32}{37}$	

Table 16: Comparisons involving two upper pairs