# Optimal Randomized Comparison Based Algorithms for Collision

Technical Report, Department of Computer Science ETH Zurich,

Number 541

Riko Jacob*

November 28, 2006

**Abstract**

We consider the problem of finding two identical elements in a list of length $n$ by randomized comparison based algorithms. We find a trade-off between the success probability $p$ and the running time $t$, and show that this trade-off is optimal up to a constant factor. For worst-case running time $t$, the optimal success probability is $p = \Theta\left(\min\{\frac{t}{n}, 1\}\frac{t}{n \log t}\right)$. For expected running time $t$, the success probability is $p = \Theta\left(t/(n \log(n))\right)$.

As part of this, but of independent interest, we determine the complexity of finding collisions in randomly chosen input according to three natural (uniform) distributions. Again, we determine the optimal (up to a constant factor) trade-off between success probability (relying on the input distribution) and running time, both worst-case and expected.

## 1 Introduction

We consider the problem of finding two equal elements, called a collision, in a list. This is a fundamental problem in computational complexity theory and among the first decision problems for which non-trivial lower bounds were known. It is usually formulated as the decision if all elements of a

*ETH Zurich, Institute of Theoretical Computer Science, 8092 Zurich, Switzerland. E-mail: rjacob@inf.ethz.ch.

list are different, and is hence called **"element uniqueness"** or **"element distinctness."** Intuitively, a positive answer, and hence the decision, requires that the total order of the elements is determined. Hence, in many models of computation, the lower bound of $\Omega(n \log n)$ for sorting carries over to this decision problem. The problem has been studied in all kinds of machine models, for example in the stronger algebraic decision tree model [BO83], in a general decision tree setting [Bop94], or on a quantum computer [BDH+05, AS04].

The investigation in this paper was triggered by a question in cryptography. There, we would like to know the precise complexity of computing discrete logarithms in a cyclic group. One promising approach to prove lower bounds for this problem are abstract models of computation [Mau05]. For example, an abstract model with comparisons is appropriate to analyze the algorithm of Pohlig and Hellman [PH78]. In this setting, we are interested in the comparison based complexity of finding a collision with a certain probability.

For a comparison based algorithm, it is actually equivalent to find a collision or to state the existence of a collision without making an error. Certainly, if the algorithm needs to show a collision to have success, it can as well just state that there is a collision. But also if a comparison based algorithm announces the existence of a collision only if it is actually present in the input, then it must have compared two identical elements, and hence is in the position to report this collision.

Here, we consider randomized algorithms (in the Monte-Carlo sense) with one-sided error. Such an algorithm must announce an existing collision only with probability $p$, but never claim a collision for an input consisting of distinct elements. We are interested in the dependence of the running time on the success probability $p$. More precisely, we consider both the success probability $p$ and the running time $t$ as functions of $n$, and are mainly interested in the asymptotic behavior for large $n$. The achieved trade-off can be formulated in two ways. Either we try to achieve a certain $p$ and optimize the running time, or we fix the running time and try to achieve the best possible success probability. As we will see, changing the running time requirement by a constant factor results in a constant factor in the bound on the probability, and vice versa. Therefore, these two ways of looking at the problem are equivalent. We also analyze the number of random bits needed by the algorithm. All in all, we give an algorithm with worst-case running time $t$ that achieves for $t(n) > 2$ success probability $p_t = \min\{\frac{t}{n}, 1\}\frac{t}{n \log t}$, and uses $\log \frac{1}{p_t}$ random bits. Both these values are optimal up to a constant

factor.

Note that for in one step we can test a single edge and achieve a success probability of $1/n^2$. From now on, we assume $p \geq 1/n^2$.

Our approach uses Yao's Minimax-principle [Yao77, MR95, p. 35], leading us to consider deterministic comparison based algorithms and their behavior when the input is drawn from a uniform distribution. Relying upon the input distribution, the algorithm should detect a collision with probability $p$. More precisely, we look at three different distributions: uniformly without collision, and uniformly with precisely one collision. We are interested in the necessary (asymptotic) running time, both as the worst-case running time of the algorithm, and as the expected running time over the randomly drawn input.

### 1.1 Related Work

The inverse setting, where a randomized comparison based algorithm solves **"element uniqueness"** (and not **"collision"**), has been studied by Snir [Sni85]. Here, the algorithm needs to recognize that all elements are different with probability $p$, but is not allowed to misclassify an input with collision. He shows a lower bound of $\lambda(n \log n - \log(p(1-\lambda))$ for all $0 < \lambda < 1$, which yields, for example, for $p = 1/\log n$ an $\Omega(n \log n)$ lower bound. This shows an important difference to collision, where this success probability can be achieved in $O(n)$ time. Grigoriev, Karpinski, Meyer auf der Heide, and Smolensky [GKMadHS97] show that if the algorithm is allowed to misclassify with probability $p < 1/2$ in both directions (two-sided errors), even for algebraic decision trees of constant degree, there is an $\Omega(n \log n)$ lower bound.

For **"collision"**, or **"element non-uniqueness"** as they call it, Manber and Tompa [MT82, MT85] give a lower bound of $\Omega(n \log n)$ for comparison based algorithms that are not allowed to announce a collision if there is none, and need to announce an existing collision with probability $1/2$. This is the special case of our result with $p = 1/2$. Similar to our main technical Lemma 3, they bound the number of linear extensions of a graph, given that at least half of the edges must be used, an idea going back to [MT84]. In contrast to the results presented here, their estimate heavily depends on the success probability being $1/2$, and does not give a lower bound for smaller probabilities like $1/\log n$.

Our Lemma 3 is an observation about the structure of the directed acyclic graph $G_c$ describing the outcomes of comparisons at a (leaf) node $c$ of the comparison tree. The observation is that a high success probability can only

be achieved if the number of linear extensions is small. The permutations ending at $c$ describe linear extensions of $G_c$, and the success probability of one permutation/linear extension $\pi$ is given by the number of successor relations of $\pi$ that coincide with edges of $G$. This number is also known as the number of steps of $\pi$ in $G$ [CH80], and is $n$ minus the number of jumps. There is some recent interest in determining the average jump number of a grid [Coo06], which is equivalent to determining the success probability of the grid. The problem of computing the number of linear extensions of a given graph is known to be #P-complete [BW91], but can be approximated in polynomial time. According to [BT03], the idea of estimating the number of linear extensions by using the entropy method goes back to Kahn [Kah02].

The connection between sorting and element distinctness is well known [BO83]. Recently, there has been some interest in approximative sorting [GSS06] with natural measures of sortedness. One can understand the DAGs produced by our algorithms as some kind of sorting, and the success probability as the measure of sortedness. Technically, one key difference is that the task in [GSS06] is transforming the input, whereas our setting asks only for a collision. Hence, their lower bounds can be achieved by reductions and hence hold also in the algebraic setting [BO83], whereas this is not clear for the problems considered here.

Similar to results on sorting, the lower bounds are valid in a strong, non-uniform model of computation (comparison trees), whereas the algorithms are quite general and can be implemented in many uniform models of computation.

### 1.2 Preliminaries

The input to $\mathbf{Collision}_n$ consists of a list of $n$ numbers $x_1, \ldots, x_n$. The answer is YES if there exist $i \neq j$ such that $x_i = x_j$, and otherwise NO. The numbers are assumed to be atomic to the algorithms, and the only operation on numbers is the comparison $x \leq y$. Hence, the main feature of an input is the order type (with identities), and without loss of generality we can assume that all $x_i \in \{1, \ldots, n\}$. The inputs without collision are then characterized by a permutation $i \mapsto x_i$. The distribution $\mathbf{p}$ is defined to be the uniform distribution over these $n!$ different inputs.

The inputs with precisely one collision can be characterized by a permutation $\pi$ and a pivot $o \in \{1, \ldots, n-1\}$ of the collision, meaning that the value $o$ is changed to the value $o+1$, creating two elements with this value. More precisely, the input is given by $x_i = \pi(i)$ if $\pi(i) \neq o$ and $x_i = o+1 = \pi(i)+1$ if $\pi(i) = o$. Consider the permutation $\pi'$ that is identi-

cal to $\pi$, only the values of $o$ and $o+1$ are exchanged. Then the input $(\pi, o)$ is indistinguishable from the input $(\pi', o)$. Further, these two representations are the only way to create this order-type. The distribution $\mathbf{q}$ is defined as the uniform distribution over these $n!(n-1)/2$ different inputs. With $\mathbf{q}|_S$ we denote the restriction to permutations from some arbitrary set $S$ with uniform probability.

There is an alternative random experiment that draws an input from distribution $\mathbf{q}$. First, choose uniformly a pair $i < j$ of indices. Set $x_i = x_j$, and then choose a random ordering (permutation) of the values of the variables $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$. In this way, $(n-1)!n(n-1)/2$ different order types are created, hence this is the same as drawing from $\mathbf{q}$.

The focus of our considerations is the number of comparisons performed by an algorithm. Hence, we model the algorithm as comparison trees of the following type: A comparison tree $T$ is a rooted tree where every internal node $c$ has a left and a right child and is annotated by a pair $(i, j)$. Every input $\mathbf{x} = x_1, \ldots, x_n$ (with or without collision) defines a path $P(\mathbf{x})$ in $T$, the next node is given by the outcome of the comparison, for $x_i < x_j$ the left child, for $x_i > x_j$ the right child. If the comparison yields $x_i = x_j$, the path stops at the corresponding node. Here, we assume that all nodes of the tree are reached for some input.

This is a non-uniform model of computation, the tree for $n$ elements need not be similar in any way to the trees for other $n$.

The success probability $p_T$ of $T$ is the probability for an input $\mathbf{x}$ (with collision) drawn from distribution $\mathbf{q}$ that the last node of $P(\mathbf{x})$ is an internal node, and hence the collision is indeed detected. Formally, we define the random variable $S(\mathbf{x})$ to be 1 if $P(\mathbf{x})$ ends at an internal node, and 0 otherwise, i.e., $P(\mathbf{x})$ ends at a leaf of $T$. For a given probability distribution on the input, we define the success probability as $E[S(\mathbf{x})]$.

This modeling reflects that we require our algorithms to have found the collision if they answer YES. Indeed, any well defined comparison based algorithm with this property can be described by a comparison tree $T$, with the same success probability and performing not more comparisons than the original algorithm.

By $|P(\mathbf{x})|$ we denote the number of internal nodes on $P(\mathbf{x})$, which is for randomly chosen $\mathbf{x}$ a random variable, representing the number of comparisons or running time on input $\mathbf{x}$. We are interested in the expected running time $C_{\mathbf{q}} = E_{\mathbf{q}}[|P(\mathbf{x})|]$ and $C_{\mathbf{p}} = E_{\mathbf{p}}[|P(\mathbf{x})|]$. We are also interested in the **worst-case running time** of $T$ which is the maximal running time (number of comparisons) for a possible input, which is given by the height of $T$.

For the purposes of analyzing $T$, we can annotate every node $c$ of $T$ by the directed graph $G_c$ that reflects the already performed comparisons. The vertices of $G'_c$ are the variables, and there is a directed arc from $x_i$ to $x_j$ if the variables were compared, and $x_i < x_j$. By this definition, $G'_c$ is a directed acyclic graph. Since the order relation is transitive, and because we are interested in single collisions we consider the irreducible core $G_c$ of $G'_c$, i.e., the graph with the fewest edges and the same transitive closure as $G'_c$.

This leads to an alternative way of computing the success probability of $T$ that explains the connection to the number of jumps of a random linear extension of a DAG. Choose uniformly a permutation $\pi$. The corresponding input vector defines a path to a leaf $c$ of $T$, and $\pi$ can be understood as a linear extension of $G_c$. Actually, the node $c$ is the only leaf of $T$ with this property. Now, uniformly choose a collision $o \in \{1, \ldots, n-1\}$ (identifying the value $o$ and $o+1$). This collision is detected if and only if there is an arc between the vertex $i$ with $\pi(i) = o$ of $G_c$ and its successor in $\pi$, which is called the **success probability of the permutation** $\pi$ in $T$, and hence in $G_c$.

Define the success probability $p_c$ of $G_c$ by the probability that a uniformly chosen linear extension of $G_c$ and a uniformly chosen collision is detected by $G_c$. Let $u_c$ be the number of linear extensions of $G_c$, and define $f_c = u/n!$. Then, the probability of reaching (ending at) $c$ with a uniformly chosen permutation is $f_c$, and we can express the success probability of $T$ as the weighted sums of the success probabilities of the leaves:

$$p_T = \sum_{c \text{ is leaf of T}} f_c \cdot p_c \,.$$

Alternatively, the success probability of a graph $G$ is given as the average over all vertices of the following success probability: The probability of the vertex $v$ to not be the last in the linear extension times the probability that the successor in the linear extension is connected to $v$ with an arc.

## 2 The Algorithms

We start with the slightly simpler deterministic algorithms that rely upon random input. Later we also consider randomized algorithms. The connection between element uniqueness and sorting is well known for comparison based models of computation. Not surprisingly, all algorithms presented here are based on sorting some subset of the input values. We use that the rank $k$ element of a list can be found in linear time [BFP+73]. Remember that sorting $k$ values takes $O(k \log k)$ comparisons.

## 2.1 Deterministic or Worst-Case Time

Consider the following Algorithm 1 that is designed to run in worst-case $O(t(n))$ time. For $t(n) = n \log n$ time, this algorithm sorts the complete input and

**Algorithm 1**: Deterministic collision find

1.1 Determine $r = \min\{n, t(n)\}$ and $k = \min\{r, t(n)/\log t(n)\}$;
1.2 Select the $k$-smallest element $x_j$ of $R = \{x_1, \ldots, x_r\}$;
Determine $S := \{x \in R \mid x \leq x_j\}$ /*$|S| = k$ */
1.3 Sort $S$;
**return** the collision if two elements of $S$ are equal;

hence finds the collision with probability 1. For $t(n) = O(1)$ the success probability is $O(1/n^2)$, comparable to testing a single edge. Note that for the interesting case $t(n) \leq n \log n$ we always have $k = t(n)/\log t(n)$.

**Lemma 1.** *Algorithm 1 runs in worst-case time $O(t(n))$ and has for distribution* $\mathbf{q}$ *success probability at least $p = \min\left\{\frac{t}{n}, 1\right\} \frac{t}{n \log t}$.*

*Proof.* The selection in Line 1 can be achieved in worst-case $O(t(n)$ time [BFP+73]. Sorting $k$ elements can be achieved in $O(k \log k) = O\left(\frac{t}{\log t} \log \frac{t}{\log t}\right) = O(t)$ worst-case time, for example with heap sort.

To compute the success probability of Algorithm 1, we consider the alternative procedure to draw an element from distribution $\mathbf{q}$, where we first decide upon the two indices $i < j$ of variables that form the collision, and then on the order of the values. The probability that both $i \leq r$ and $j \leq r$ is $\frac{r}{n} \frac{r-1}{n-1}$. The rank of the value $x_i = x_j$ within $x_1, \ldots, x_r$ is uniform between 1 and $r - 1$, hence the probability for it to be $\leq k$ is The probability that the collision gets a rank $\leq k$ is $k/(r-1)$.

Hence, the success probability of Algorithm 1 is $\frac{r}{n} \frac{r-1}{n-1} \frac{k}{r-1} > kr/n^2$. If $t(n) > n \log n$, we have $r = k = n$ and hence $p = 1$, as stated in the lemma. For $n \leq t(n) \leq n \log n$ we have $r = n$ and hence $p = k/n$, the statement of the lemma. Finally, for $t(n) < n$, we have $r = t(n)$ and hence $r/n < 1$, leading to $p = (r/n)(k/n)$, again the statement of the lemma. □

## 2.2 Expected Time

In comparison to the worst-case time, it is easier to achieve good expected running times because on some inputs the algorithm may be slow if it is fast on others. More precisely, if a fraction $p$ of the inputs is sorted completely,

the success probability is $p$, and the expected running time is $O(pn \log n)$, as long as the expected running time of a non-successful input is $O(1)$.

To achieve this, we use that in distribution $\mathbf{q}$ and $\mathbf{p}$, the outcomes of the ***i*-th canonical test**, comparing $x_{2i} \overset{?}{<} x_{2i+1}$ are independent for different $i \in \{1, \ldots, \lfloor n/2 \rfloor\}$. Chose the integer $k$ such that $2^{-k} \geq p > 2^{-k-1}$. By the assumption $p > 1/n^2$, we have $k \leq 2 \log n < \lfloor n/2 \rfloor$ for $n > 7$. The algorithm performs the canonical tests in the natural order. As soon as one of the tests fails, the algorithm stops. Once test $k$ succeeds, the algorithm sorts the input and hence finds all collisions. Hence, the success probability is at least $2^{-k} \geq p$. The expected running time until a failing test is reached is bounded by $\sum_{i=1}^{k} i2^{-i} = O(1)$. Hence, the expected running time is $O(1 + pn \log n)$.

## 2.3 Randomized Algorithms

### 2.3.1 Expected time

Again, if only a good expected running time should be achieved, the algorithm is very simple, if we allow to toss an arbitrarily biased coin. With probability $p$, we solve the problem deterministically in $O(n \log n)$ time, otherwise we do nothing and declare that we find no collision. This algorithm has expected success probability $p$ and expected running time $O(pn \log n)$ on any input. If only unbiased binary coins are allowed, $p$ should be overestimated as $2^{-k}$, leading to the same asymptotic performance.

By Yao's minimax principle [Yao77, MR95, p. 35] and Lemma 9 this is asymptotically optimal.

### 2.3.2 Worst-case time

Now consider the case where the randomized algorithm should never exceed a certain running time, and still find a collision with reasonably high probability. The idea here is to use few random bits to "simulate" distribution $\mathbf{q}$ in Algorithm 1.

Let $t = t(n) \leq n \log n$ be the goal for a asymptotic worst-case running time. We design an algorithm with running time $O(t(n))$ and high success probability. For the case $t < n/2$ the variables are divided equally into $k = \lfloor n/t \rfloor$ classes, such that the size is $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$. Now, choose uniformly at random two different such classes and call the resulting set of variables $R$ and define $r = |R|$. Observe that $\lceil n/k \rceil \leq n/k + 1 \leq n/(n/t - 1) + 1 = t/(1 - t/n) + 1 \leq 2t + 1$, and hence $r = O(t(n))$. Divide the set $[r]$ equally into ranges of length at least $t/\log t$ and at most $2t/\log t$. Chose one such

range $[a, b]$, determine the rank-$a$ element of $R$ and the rank-$b$ element, such that another scan over $R$ yields the set $S$ of elements whose rank is in the range $[a, b]$, and sort this set $S$. By a similar calculation as for Algorithm 1, the worst case running time of this algorithm is $O(t(n))$.

To have success, the algorithm needs to randomly chose the two classes where the variables of the collision are located, and it must randomly chose the rank of this collision within the set $R$.

For the case $t < n$, there are $k \leq n/t$ classes, and at most $\log t$ ranges, such that this success probability is at least $p = (t/n)^2(1/\log t)$. Otherwise, only the choice of the range is random, the range with the collision is chosen with probability at least $\frac{t}{\log t}\frac{1}{n}$.

The following theorem formulates that this algorithm is optimal in terms of running time up to a constant factor, and in terms of number of random bits used up to an additive constant.

**Theorem 2.** *Assume a randomized algorithm $\mathcal{A}$ solves $\boldsymbol{Collision}_n$ for all inputs in time $t$ and with positive success probability. Then, with $r = \min\{t, n\}$ and $p_t \leq \frac{8r^2}{(n-1)^2 \log(2t)}$, the success probability of $\mathcal{A}$ is at most $p_t$ and it uses at least $-\log p_t$ random bits.*

*Proof.* We see the randomized algorithm $\mathcal{A}$ as first using a certain number $b$ of random bits (biased or unbiased) to select one of at most $2^b$ deterministic algorithms with worst-case running time $t$. By Yao's minimax principle [Yao77, MR95, p. 35] and Lemma 11, any deterministic algorithm has success probability at most $p_t$, giving the bound on the success probability of the algorithm. Let $X = n!(n-1)/2$ be the number of different inputs. The algorithm has success on at most $X \cdot p_t$ inputs, such that the total number of successful inputs is $Xp_t2^b$. If $p_t2^b < 1$, there would exist an input that fails for all random choices. Hence, $2^b \geq 1/p_t$, $b \geq -\log p_t$. □

## 3 The lower bound

The lower bound proof starts by giving an upper bound on the number of linear extensions that are compatible with leafs of the comparison tree that have high success probability. This allows to conclude different worst-case and expected running times.

The main ingredient is an upper bound on the number of permutations that are linear extensions of a DAG with high success probability. Additionally, very fast algorithms cannot have a too high success probability because they must leave some of the input unconsidered. Finally, to argue about all possible algorithms, the two settings need to be carefully combined.

### 3.1 High-probability DAGs

The following information theoretic consideration shows that not too many permutations can take many arcs of a given DAG as successors.

**Lemma 3.** *Let $G = (V, E)$ be a directed acyclic graph with $n$ vertices, $V = \{1, \ldots, n\}$. Then, the number of linear extensions of $G$ with at least $k$ arcs in $G$ is at most*

$$\binom{n}{k} \cdot \frac{n!}{k!}$$

*Proof.* Any linear extension with at least $k$ arcs in $G$ can be described by the set $A$ of additional arcs that need to be inserted into $G$ to yield a directed path (thinking of $G$ as an order-relation, the additional comparisons that are necessary to make all elements comparable). Since at least $k$ arcs of $G$ are used we have $|A| \leq n - k - 1 < n - k$. Define $T_A \subseteq V$ to be the starting points arcs in $A$. Now, the arcs form an injective mapping $\gamma\colon T_A \to \{1, \ldots, n\}$. There are at most $\binom{n}{n-k} = \binom{n}{k}$ possibilities for $T_A$, and at most $\frac{n!}{k!}$ possibilities for $\gamma$. This leads to the claimed bound. □

**Lemma 4.** *Given a graph $G$ on $n$ vertices and a set $S$ of permutations on $[n]$. Assume that the success probability of $G$ when drawing uniformly permutations from $S$ that are linear extensions of $G$ and uniformly the collision is at least $p \geq n^{-\frac{1}{6}}$. Then the number $u$ of permutations in $S$ that are linear extensions of $G$ is bounded by $-\log \frac{u}{|S|} \geq \frac{pn}{6} \log n - 3pn - \frac{\log n}{6} - 1 + \log \frac{|S|}{n!}$.*

*Proof.* For a given linear extension/permutation, the success probability $q$ is given by the number $k$ of arcs of $G$ it is using as $q = k/(n-1)$. Let $R \subseteq S$ be the set of permutations with at least $k = \lceil pn/2 \rceil$ arcs of $G$, i.e., permutations that have success probability at least $p/2$. By a Markov inequality we have $r = |R| \geq pu/2$, and Lemma 3 yields $r \leq \binom{n}{k} \cdot \frac{n!}{k!}$. Hence,

$$u \leq \binom{n}{\lceil pn/2 \rceil} \cdot \frac{2n!}{p\lceil pn/2 \rceil!}.$$

We use the well known bounds on the binary logarithm of the factorial $x(\log x - 2) \leq \log(x!)$ and $\log \binom{x}{y} \leq 3y \log(x/y)$ for the case $y < x/2$ that

derive from Stirling's formula. Define $x$ by $2^{-x}n! = u$. This yields

$$\begin{aligned}
x &= \log(n!/u) && (1)\\
&\geq \log(p\lceil pn/2\rceil!) - \log\binom{n}{\lceil pn/2\rceil} - \log 2 && (2)\\
&\geq \log p + \left\lceil\frac{pn}{2}\right\rceil\Big(\log pn/2 - 2\Big) - 3\left\lceil\frac{pn}{2}\right\rceil\log\left(\frac{n}{\lceil pn/2\rceil}\right) - 1 && (3)\\
&\geq \log p + \frac{pn}{2}\Big(\log n + \log(p/2) - 3\log(2/p) - 2\Big) - 1 && (4)\\
&= \log p + \frac{pn}{2}\Big(\log n - 4\log(2/p)\Big) - pn - 1 && (5)\\
&= \log p + \frac{pn}{2}\Big(\log n - 4\log(1/p) - 4\Big) - pn - 1 && (6)\\
&\geq -\frac{\log n}{6} + \frac{pn}{2}\Big((1-4/6)\log(n)\Big) - 3pn - 1 && (7)\\
&= \frac{pn\log n}{6} - 3pn - \frac{\log n}{6} - 1 && (8)
\end{aligned}$$

In (4), we omit the ceiling in the denominator of a negative term (inside the log), only lessening the term, and omit the ceiling in a positive product. In (7) we use the bound on $p \geq n^{-\frac{1}{6}}$ yielding $\log(1/p) \leq \frac{\log n}{6}$ twice.

□

### 3.2 Easy Low-Probability Bound

Now, we need to turn to the low-probability situations.

**Lemma 5.** *Consider the random experiment of drawing a permutation, giving rise to the rank-variables $x_1, \ldots, x_n$. Assume $r > 1$, that the set of possible permutations is given by disallowing some permutations on $x_1, \ldots, x_r$, and that all allowed permutations are equally likely. Then, for any fixed variable $x_i$, $1 \leq i \leq r$, the probability that the successor of $x_i$ is in $x_{r+1}, \ldots, x_n$ is $(n - r + 1)/n$.*

*Proof.* Consider the experiment of first choosing the permutation of $x_1, \ldots, x_r$, and then, one by one, the relative positions of $x_{r+1}, \ldots, x_n$ in that order. Now, the probability for $x_j$ to not become the successor (at its stage) of $x_i$ is $1 - 1/(j-1) = (j-2)/(j-1)$. Hence, the probability that the successor of $x_i$ remains unchanged is $\frac{r-1}{r} \cdot \frac{r}{r+1} \cdots \frac{n-2}{n-1} \cdot \frac{n-1}{n} = \frac{r-1}{n}$. □

**Lemma 6.** *Assume a leaf node $i$ of a decision tree $T$ for **Collision**$_n$ has success probability $p_i$ for $\mathbf{q}$. Then the depth $d_i$ of $i$ is at least $d_i \geq \frac{n-1}{2}\sqrt{p_i}$. For $p_i \leq n^{-\frac{1}{6}}$, this implies $d_i \geq \frac{p_i n\log n}{6} - 3p_i n - \frac{\log n}{6} - 1$*

*Proof.* Let $G_i$ be the comparison graph of node $i$ and $G'$ the subgraph of $G_i$ that consists of the non-isolated vertices $R$. By the depth restriction $r < 2d_i$.

Recall the definition of success probability: First we choose uniformly a permutation $\pi$ that is a linear extension of $G_i$. Then we uniformly choose a vertex $o$ from the $n-1$ vertices that have a successor, i.e., that are different from the last position. The experiment is successful if there is an edge in $G_i$ from $o$ to its successor in $\pi$.

Here, three events must happen:

(1) $o \in R$

(2) the $\pi$-successor $q$ of $o$ is in $R$

(3) there is an arc from $o$ to $q$ in $G'$

No vertex can become $o$ with probability greater than $1/(n-1)$. Hence, the probability of Event (1) is at most $r/(n-1)$. By Lemma 5 the probability for Event (2) is $(r-1)/n$. We trivially bound the probability for Event (3) by 1.

Hence, $p_i \leq \frac{r(r-1)}{(n-1)n} < \frac{(2d_i)^2}{n(n-1)}$. This is $2d_i > \sqrt{p_i n(n-1)} > \sqrt{p_i}(n-1)$, implying $d_i > \frac{n-1}{2}\sqrt{p_i}$.

Now, assume $p_i \leq n^{-\frac{1}{6}}$. We will show $\frac{n-1}{2}\sqrt{p_i} \geq \frac{pn\log n}{6} - 3pn - 1$, which implies the statement of the lemma. To see this, we observe that $-\frac{1}{2}\sqrt{p} > -1$, and hence it is sufficient to argue for $\frac{n}{2}\sqrt{p} \geq pn\left(\frac{\log n}{6} - 3\right)$, or equivalently $1/\sqrt{p} \geq \frac{\log n}{3} - 6$. To this end, we consider the function $f(n) = n^{\frac{1}{12}} - \frac{\log n}{3} + 6$, and argue that $f(n) > 0$ for all $n > 0$. Because for $\log n \leq 18$ we certainly have $f(n) > 0$, and because $\lim_{n\to\infty} f(n) = +\infty$, the existence of a point $\hat{x}$ with $f(\hat{x}) < 0$ would imply a local minimum $x > 0$ with $f(x) < 0$. At such a local minimum we would have $f'(x) = 0$, i.e., $1/(12x^{\frac{11}{12}}) - 1/(3x\ln 2) = 0$, $x^{\frac{1}{12}} = 4/\ln 2$, yielding $x = (4/\ln 2)^{12}$. Now, $f(x) = (4/\ln 2) - 12\log(4/\ln 2)/3 + 6 > 4/\frac{3}{4} - 4\log 4/\frac{2}{3} + 6 = 16/3 - 4\log 6 + 6 > 5 - 4\frac{8}{3} + 6 = 11 - 32/3 > 0$, a contradiction that shows $f(n) > 0$ for all positive $n$. Here we used the estimate $2/3 < \ln 2 < 3/4$ and $\log 6 < 8/3$. □

### 3.3 Expected Running Time without Collisions

If we consider the expected running time of an algorithm, even a fast algorithm can have nodes at considerable depth.

**Lemma 7.** *Let $T$ be a comparison tree solving* ***Collision***$_n$*, and $S$ a set of permutations. Assume an input drawn from $\mathbf{q}|_S$ (uniform permutation in $S$ and uniform collision) has success probability at least $p$ and expected running time $D$ for input from $\mathbf{p}|_S$, i.e., a uniformly chosen permutation from $S$ without collision. Then $D \geq \frac{pn}{6}\log n - 3pn - \frac{\log n}{6} - 1 + \log\frac{|S|}{n!}$*

*Proof.* Every leaf $i$ of $T$ has success probability $p_i$, a depth $d_i$, and a fraction $f_i$ of the permutations from $S$ that end at $i$. Now, by definition, $D = \sum f_i \cdot d_i$, and $p = \sum f_i \cdot p_i$.

We define two classes of nodes, the high-probability nodes $H = \{i \mid p_i > n^{-\frac{1}{6}}\}$, and the remaining nodes $L$. Define further for these two classes the split $f_H = \sum_{i\in H} f_i$, and similarly $f_L = \sum_{i\in L} f_i$, such that $f_H + f_L = 1$. The restricted probabilities $p_H$ and $p_L$, and the restricted expected running times $D_L$ and $D_H$ are defined by $f_H \cdot p_H = \sum_{i\in H} f_i \cdot p_i$, $f_H \cdot D_H = \sum_{i\in H} f_i \cdot d_i$, $f_L \cdot p_L = \sum_{i\in L} f_i \cdot p_i$, $f_L \cdot D_L = \sum_{i\in L} f_i \cdot d_i$. These values satisfy $p = f_H \cdot p_H + f_L \cdot p_L$, and $D = f_H \cdot D_H + f_L \cdot D_L$.

Define for $i \in H$ the relative reach-probability by $f'_i = f_i/f_H$. Note that the $f'_i$ sum to 1, i.e., they form a probability distribution.

Define $a_i = 2^{-d_i}$, $A_H = \sum_{i\in H} a_i$, such that the values $a'_i = a_i/a_H$ sum to 1 and form a probability distribution.

With this, we get

$$
\begin{aligned}
D_H &= -\sum_{i\in H} f'_i \log a_i = -\log a_H - \sum_{i\in H} f'_i \log a'_i && (9)\\
&\geq -\log a_H - \sum_{i\in H} f'_i \log f'_i && (10)\\
&= -\log f_H - \log a_H - \sum_{i\in H} f'_i \log f_i && (11)\\
&\geq \log -f_H + \sum_{i\in H} f'_i \left(\frac{p_i n \log n}{6} - 3p_i n - \frac{\log n}{6} - 1 + \log\frac{|S|}{n!}\right) && (12)\\
&= -\log f_H + \frac{pn}{6}\log n - 3pn - \frac{\log n}{6} - 1 + \log\frac{|S|}{n!}. && (13)
\end{aligned}
$$

Where the inequality (10) is Gibbs' inequality and the inequality (12) is the statement of Lemma 4, together with the fact that $-\log a_H \geq 0$.

Now, consider a node $i \in L$ of $T$, i.e., with low probability $p_i \leq n^{-\frac{1}{6}}$. By Lemma 6 we have the depth-bound $d_i \geq \frac{p_i n}{6}\log n - 3pn - \frac{\log n}{6} - 1$.

Using this inequality, we get

$$D_L = \sum_{i \in L} f'_i \cdot d_i \quad \geq \quad \sum_{i \in L} f'_i \cdot \left( \frac{p_i n}{6} \log n - 3 p_i n - \frac{\log n}{6} - 1 \right) \tag{14}$$

$$= \quad \frac{p_L n}{6} \log n - 3 p_L n - \frac{\log n}{6} - 1 \tag{15}$$

Now, the lemma follows by

$$D \quad = \quad f_H \cdot D_H + f_L \cdot D_L \tag{16}$$

$$\geq \quad (f_H p_H + f_L p_L) \left( \frac{n}{6} (\log n - 18) \right) - \frac{\log n}{6} - 1 + f_H \log \frac{f_H |S|}{n!} \tag{17}$$

$$\geq \quad \frac{pn}{6} \log n - 3pn - \frac{\log n}{6} - 1 + \log \frac{|S|}{n!} \tag{18}$$

□

**Lemma 8.** *Let $T$ be a linear decision tree solving **Collision**$_n$. Assume that the expected running time $D$ for uniformly chosen permutations without collision ($\mathbf{p}$) is $D = n/2$. Then the success probability $p$ (for $\mathbf{q}$) is bounded by $p \leq 4/\log n$.*

*Proof.* Lemma 7 gives $n/2 \geq \frac{pn}{6} \log n - 3pn - \frac{\log n}{6} - 1$. $p \leq 6(n/2 + \frac{\log n}{6} + 1)/n(\log n - 18) < (3n + \log n + 1)/n \log n < 4/\log n$. □

**Theorem 9.** *Assume that the linear decision tree $T$ solves **Collision**$_n$ with success probability at least $p$ (for $\mathbf{q}$) and expected running time $D$ (for $\mathbf{p}$). Then $D = \Omega(pn \log n)$.*

For a function $n < t(n) < n \log n$ Algorithm 1 runs in time $O(t(n))$ and achieves success probability $p_t = \frac{t(n)}{n \log t(n)} \geq \frac{t(n)}{n \log n}$. By Theorem 9, $p_t$ requires asymptotic running time $\Omega(\frac{t(n)}{n \log n} n \log n) = \Omega(t(n)$. Hence Algorithm 1 is asymptotically optimal for success probability $p_t$.

### 3.4 Strong Low Probability Bound for Worst-Case Time

Certainly, the lower bound on the expected running time is also lower bound on the worst-case running time. Still, for sub-linear time algorithms the success probability is significantly lowered by the impossibility to touch all vertices.

**Lemma 10.** *Let $T$ be a comparison tree for* ***Collision***$_n$ *with maximal depth $r < n/2$, and that input is drawn from* $\mathbf{p}$ *or* $\mathbf{q}$*. Then there is a comparison tree $T'$ with the same success probability, expected and worst-case running time as $T$, and $T'$ uses only the variables $x_1, \ldots, x_{2r}$.*

*Proof.* Structurally, $T$ and $T'$ are the same, they differ only in the variable names. Rename variables (recursively from root to leafs) in a way that any new variable (so far not part of any comparison) is changed to the new variable with the lowest index. Now, the comparison graphs at the corresponding nodes of $T$ and $T'$ are isomorphic and hence reached with the same probability if input is drawn from $\mathbf{p}$ or $\mathbf{q}$. □

**Lemma 11.** *Any comparison tree $T$ with worst-case running time $t \leq n$ has (average) success probability $p \leq p_t =\leq \frac{16t^2}{(n-1)^2 \log(2t)}$.*

*Proof.* With $r = 2t$, by Lemma 10 w.l.o.g. the comparison tree $T$ solves $\mathbf{Collision}_r$ in worst-case time $t$ and success probability $q$, which is by Lemma 8 bound by $q \leq 4/\log r = 4/\log(2t)$. Now, by the argument of Lemma 6, $p = \frac{r(r-1)}{(n-1)n} q \leq \frac{16t^2}{(n-1)^2 \log(2t)}$. □

For some function $t(n) < n$, Algorithm 1 runs in worst-case time $O(t(n))$ and achieves success probability $\frac{t(n)^2}{n^2 \log t(n)}$, by this asymptotically matching the bound of Lemma 11.

### 3.5 Expected Time for Random Input with Collision

**Theorem 12.** *Let $T$ be a linear decision tree solving* ***Collision***$_n$*. Assume that the success probability for input distribution* $\mathbf{q}$ *is $p < 1/4$, and the expected running time for* $\mathbf{q}$ *is $C_\mathbf{q}$. Then, $C_\mathbf{q} \geq \frac{pn}{12}(\log n - 18) - \frac{\log n}{12} - 3$.*

*Proof.* Let $S$ be the set of permutations that have success probability $> 1/2$. With $f_S := |S|/n!$ we can express running time and probability as $C = f_S C_S + (1 - f_S) C_{\bar{S}}$, where $C_S$ is the expected running time for permutations in $S$, and $C_{\bar{S}}$ the expected running time for permutations not in $S$. Similarly, we can write the success probability as $p = f_S p_S + (1 - f_S) p_{\bar{S}}$.

For permutations not in $S$, half the contribution to the average running times stems from undetected collision. Hence, it can be estimated using $C_\mathbf{p}$, by Lemma 7, we have $C_{\bar{S}} \geq \left(\frac{np_{\bar{S}}}{6}(\log n - 18) - \frac{\log n}{6} - 1 + \log(1 - f_S)\right)/2$.

By a Markov inequality, at least half of the inputs in $\mathbf{q}|_S$ stop at times before $2C_S$. Cut $T$ at depth (time) $2C_S$, leading to $T'$ with success probability $p'_S \geq 1/4$. Now, because the expected running time is less than the

worst-case running time of $T'$, Lemma 7 yields $2C_S \geq \frac{pn}{6\cdot 4}(\log n - 18) - \frac{\log n}{6} - 1 + \log f_S$.

It remains to take the weighted sums of the bounds on $2C_S$ and $2C_{\bar{S}}$. Because for $p \leq 1/4$ we have $n/4 \geq pn$, we get $2C_S \geq \frac{n}{6}(\log n - 18) - \frac{\log n}{6} - 2$. Here, the last term stems from $f_S \log f_S + (1-f_S)\log(f_S - 1) > -1$, implying the statement of the lemma. □

**Corollary 13.** *Let $T$ be a linear decision tree solving **Collision**$_n$. Assume that the success probability for input distribution* $\mathbf{q}$ *is $p$. Then the expected running time for* $\mathbf{q}$ *is $\Omega(pn \log n)$.*

## Acknowledgment

I would like to thank Ueli Maurer and Dominik Raub for introducing me to the problem and for several fruitful discussions.

## References

[AS04] Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM*, 51(4):595–605 (electronic), 2004. 2

[BDH+05] Harry Buhrman, Christoph Dürr, Mark Heiligman, Peter Høyer, Frédéric Magniez, Miklos Santha, and Ronald de Wolf. Quantum algorithms for element distinctness. *SIAM J. Comput.*, 34(6):1324–1330 (electronic), 2005. 2

[BFP+73] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973. 6, 7

[BO83] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annual ACM Symposium on Theory of Computing*, pages 80–86, 1983. 2, 4

[Bop94] Ravi B. Boppana. The decision-tree complexity of element distinctness. *Inf. Process. Lett.*, 52(6):329–331, 1994. 2

[BT03] Graham R. Brightwell and Prasad Tetali. The number of linear extensions of the boolean lattice. *Order*, 20(4):333+, 2003. 4

[BW91] Graham Brightwell and Peter Winkler. Counting linear extensions is #P-complete. In *STOC*, pages 175–181. ACM, 1991. 4

[CH80] M. Chein and M. Habib. The jump number of DAGs and posets: An introduction. *Annals of Discrete Mathematics*, 9:189–194, 1980. 4

[Coo06] Joshua Cooper. Random linear extensions of grids, 2006. 4

[GKMadHS97] Dima Grigoriev, Marek Karpinski, Friedhelm Meyer auf der Heide, and Roman Smolensky. A lower bound for randomized algebraic decision trees. *Comput. Complexity*, 6(4):357–375, 1996/97. 3

[GSS06] J. Giesen, E. Schuberth, and M. Stojakovic. Approximate sorting. In *Proceedings of the 7th Latin American Theoretical Informatics Symposium (LATIN)*, 2006. 4

[Kah02] Jeff Kahn. Entropy, independent sets and antichains: a new approach to Dedekind's problem. *Proc. Amer. Math. Soc.*, 130(2):371–378 (electronic), 2002. 4

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography. In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005. 2

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995. 3, 8, 9

[MT82] Udi Manber and Martin Tompa. Probabilistic, nondeterministic, and alternating decision trees (preliminary version). In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 234–244, New York, NY, USA, 1982. ACM Press. 3

[MT84] Udi Manber and Martin Tompa. The effect of number of Hamiltonian paths on the complexity of a vertex-coloring problem. *SIAM J. Comput.*, 13(1):109–115, 1984. 3

[MT85] Udi Manber and Martin Tompa. The complexity of problems on probabilistic, nondeterministic, and alternating decision trees. *J. Assoc. Comput. Mach.*, 32(3):720–732, 1985. 3

[PH78] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, Jan 1978. 2

[Sni85] Marc Snir. Lower bounds on probabilistic linear decision trees. *Theoret. Comput. Sci.*, 38(1):69–82, 1985. 3

[Yao77] A. C. C. Yao. Probabilistic computations: towards a unified measure of complexity. In *Proc. 18th FOCS*, pages 222–227. IEEE, 1977. 3, 8, 9