Optimal Randomized Comparison Based Algorithms for Collision

Riko Jacob*

Computer Science Department, Technische Universität München
jacob@in.tum.de

Abstract. We consider the well known problem of finding two identical elements, called a collision, in a list of n numbers. Here, the (very fast) comparison based algorithms are randomized and will only report existing collisions, and do this with (small) probability p, the success probability. We find a trade-off between 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 $p = \Theta\left(\min\{t/n, 1\}t/(n \log t)\right)$. For expected running time t, the success probability is $p = \Theta\left(t/(n \log n)\right)$.

1 Introduction

We consider the problem of finding two equal elements, called a collision, in a list of numbers. 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 list are different, and is hence called **"element uniqueness"** or **"element distinctness"**. One way to guarantee that no element is repeated is to provide a linear sequence of strict inequalities. Actually, in many models of computation where numbers are treated as atoms, 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 [1], in a general decision tree setting [2], or on a quantum computer [3, 4].

Motivation The investigation in this paper was triggered by a question in cryptography, namely the precise complexity of computing discrete logarithms in a cyclic group of roughly $n \approx 2^h$ numbers, where the input can be specified using $O(h)$ bits. Here, cryptographic primitives are build on the assumption that exponentiation is a good one-way function, i.e., it is easy to compute $x = a^z \mod n$, but it is difficult to invert this function, namely to compute z from x, the discrete logarithm of x to the base a.

One classical algorithm for the discrete logarithm is the so called baby-step giant-step algorithm that produces two lists A, B of $\sqrt{n}$ elements and determines the discrete log from a collision between the lists, i.e., elements $a \in A$ and $b \in B$ with $a = b$, a generalization of the collision problem considered here. If we allow

* Work done at ETH Zurich, Institute of Theoretical Computer Science, Switzerland.

the algorithm to use a hash table and assume that the algebraic operations have unit cost, this algorithm takes $O(\sqrt{n})$ time, whereas a comparison based algorithm, where a hash table is not allowed and is replaced by a balanced search tree, takes $O(\sqrt{n}\log n)$ time. Now, in the cryptographic setting, of course lower bounds for the discrete log computation are even more interesting than upper bounds. Today, no good general lower bounds are known, which led to the investigation of restricted classes algorithms. One such restriction are abstract models of computation [5], that treat the group as a black box that allows only the group operations and a comparison of elements. Now, one can show a $\sqrt{n}$ lower bound to compute the discrete log with constant probability if the group order n is prime [5], matching the performance of the hash-table based version of the baby-step giant-step algorithm. Here, we consider the situation of allowing only comparisons, which makes a hash table impossible. We show that this strengthens the lower bound by the logarithmic factor the algorithm is slowed down, and that this remains the case for small success probabilities. This setting also motivates to consider worst case running times which translate into the assumption that a code-breaking algorithm can run only for a limited time, which might be in contrast to its expected running time.

Randomized Algorithms Here, we consider randomized algorithms (in the Monte-Carlo sense) with one-sided error. Such an algorithm must announce an existing collision with probability p, but never claim a collision for an input consisting of distinct elements. We are interested in how the running time depends 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.

Randomized Input Intuitively, it is clear that a completely random input will make it particularly difficult to find a collision, and, because the input is random, even a deterministic algorithm has access to randomness. This idea directly suggests two particular uniform distributions that are discussed in detail in Section 1.2. First, there is the uniform distribution $\mathbf{q}$ of all inputs with precisely one collision. This kind of input reveals the success probability of the algorithm. Secondly, there is the uniform distribution $\mathbf{p}$ of all inputs without collision. This is useful to analyze the worst case running time of the algorithm. We analyze the asymptotic worst-case and expected running time of deterministic algorithms achieving success probability p. Using Yao's Minimax-principle [6, 7, p. 35], we can transfer the obtained lower bound to randomized algorithms.

Output and allowed errors 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.

It is also meaningful to consider only the decision problem and to allow the algorithm to announce a non-existing collision with a certain probability $q < p$. For a comparison based algorithm, this means that in some situations where no comparison showed equality, the algorithm still announces a collision. By eliminating this behavior one gets an algorithm that announces only existing collisions and has success probability at least $p - q$.

New Results We parametrize the algorithms by a goal running time function $t(n)$. The deterministic Algorithm 1 achieves in worst-case $O(t(n))$ time a success probability $p_t(n) = \min\{\frac{t}{n}, 1\}\frac{t}{n \log t}$ when input is drawn from distribution $\mathbf{q}$. Here, and throughout the paper, log stands for the binary logarithm. In Section 2.3, this algorithm is randomized, resulting in an algorithm with the same worst-case performance on arbitrary input. Additionally, we analyze the above situation for expected running times (instead of worst-case). There we find that success probability $p(n)$ requires running time $t_p(n) = \Theta(p(n) \cdot n \log n)$, or put the other way around, that expected running time $t(n)$ allows for success probability $p_t(n) = \Theta\left(\frac{t \log t}{n \log n}\right)$.

Observe that in both situations, for any positive constant c we have $p_{c \cdot t}(n) = \Theta(p_t(n))$. Vice versa, to change the success probability by a constant factor c, it is always possible to choose a $t'(n) = \Theta(t(n))$ with $p_{t'} = c \cdot p_t$. Hence, the trade-off between running time and success probability can be formulated either way, and it is meaningful to state that the mentioned algorithm is optimal up to a constant factor.

On a side note, we also analyze the maximal number of random bits needed by an algorithm to be $\Theta(-\log p)$.

As is usual, 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.

1.1 Related Work

The inverse setting, where a randomized comparison based algorithm solves **"element uniqueness"** (and not **"collision"**), has been studied by Snir [8]. There, the algorithm needs to recognize that all elements are different with probability p, but is not allowed to misclassify an input with collision. The result is a lower bound of $\lambda n \log n + \lambda \log(p(1 - \lambda))$ for all $0 < \lambda < 1$, which yields, for example, for $p = 1/\log n$ (with $\lambda = 1/2$) 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 [9] 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 the problem **"collision"**, or **"element non-uniqueness"** as they call it, Manber and Tompa [10, 11] 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 [12]. 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 π is given by the number of successor relations of π that coincide with edges of G. This number is also known as π, the number of steps in G [13], and often analyzed as the number of jumps $n - 1 - \pi$.

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 $x \leq y$. A set of inputs that are indistinguishable by such comparisons are called an **order type**, and for $\mathbf{Collision}_n$ there is no loss of generality in assuming that all $x_i \in \{1, \ldots, n\}$. An input without collision is then a permutation π understood as $x_i = \pi(i)$. The variable j with $x_j = \pi(j) = \pi(i)+1$ is called the successor of i. The distribution $\mathbf{p}$ is defined to be the uniform distribution over these $n!$ different inputs.

Choosing uniformly from the inputs with precisely one collision is called the distribution $\mathbf{q}$. There are two interesting procedures to draw from this distribution.

The **collision first** procedure starts by choosing uniformly a pair $i \neq j$ 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, the $(n-1)!n(n-1)/2$ different order types with precisely one collision are created with equal probability.

Alternatively, the **permutation first** procedure chooses a permutation π and a pivot $o \in \{1, \ldots, n-1\}$ of the collision, both with uniform probability. The input is created by changing the value o to the value $o+1$, creating two elements with value $o+1$. 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 π' that is identical to π, only the values of o and $o + 1$ are exchanged. Then the input (π, o) is indistinguishable from the input (π', o). Further, these two representations are the only way to create this order-type. Again, the $n!(n-1)/2$ different inputs are created with equal probability.

We denote the uniform distribution of inputs without collision when the defining permutation is restricted to belong to S as $\mathbf{p}|_S$. With $\mathbf{q}|_S$ we denote the restriction of $\mathbf{q}$ where in the permutation first procedure π is restricted to the

set S. Note that $\mathbf{q}|_S$ is not a uniform distribution because inputs can be created from one or two permutations of S.

The focus of our considerations is the number of comparisons performed by an algorithm. Hence, we model the algorithm as a family of comparison trees of the following type: For every input size n (number of variables), there is a comparison tree T, 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. More precisely, 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.

2 The Algorithms

We start with the slightly simpler deterministic algorithms that rely upon random input, then 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 [14], and 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 hence

Algorithm 1: Deterministic collision find

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

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

Lemma 1. *For a function t that can be computed in $O(t(n))$ time, Algorithm 1 runs in worst-case time $O(t(n))$ and computes $\boldsymbol{Collision}_n$ on input drawn from distribution $\mathbf{q}$ with success probability at least $p = \min\left\{\frac{t(n)}{n}, 1\right\} \frac{t(n)}{n \log t}$.*

Proof. The selection in Line 2 can be achieved in worst-case $O(t(n)$ time [14]. 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, consider the "collision first" procedure to draw an element from distribution $\mathbf{q}$. The probability that the collision is in the first variables ($i < j \leq r$) 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 $\frac{k}{r-1}$. Hence, the success probability of Algorithm 1 is $\frac{r}{n}\frac{r-1}{n-1}\frac{k}{r-1} > \frac{kr}{n^2}$. 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 $p = \frac{k}{n}$. Finally, for $t(n) < n$, we have $r = t(n)$ and hence $\frac{r}{n} < 1$, leading to $p = \frac{r}{n}kn$. □

In Section 3.4 Lemma 8 shows that the trade-off between success probability and worst-case running time is asymptotically optimal if $t(n) < n$. Otherwise, this follows from Section 3.3, Lemma 6 because the worst case running time is an upper bound on the expected running time.

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 $O(1)$. If $p \leq 1/n^2$, comparing x_1 and x_2 suffices.

To achieve this, we use that in distribution $\mathbf{q}$ and $\mathbf{p}$, the outcomes of the **i-th canonical test**, comparing x_{2i} with x_{2i+1} are independent for different $i \in \{1, \ldots, \lfloor n/2 \rfloor\}$, and will be used to emulate random bits. Choose the integer k such that $2^{-k} \geq p > 2^{-k-1}$. For $p > 1/n^2$, we have $k \leq 2 \log n < \lfloor n/2 \rfloor$ if $n > 7$.

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 $O(1 + pn \log n)$.

This running time is asymptotically optimal, for distribution $\mathbf{p}$ this is shown in Section 3.3 (Lemma 6), for distribution $\mathbf{q}$ in Section 3.5 (Lemma 9).

2.3 Randomized Algorithms

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, which is optimal by Theorem 1.

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 $2t/\log t$. Choose 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 choose the two classes where the variables of the collision are located, and it must randomly choose 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 $\frac{t}{\log t}\frac{1}{n}$.

We summarize the above discussion in the following Lemma.

Lemma 2. *Let t be a function that can be computed in $O(t(n))$ time, then there is a randomized algorithm that runs in worst-case time $O(t(n))$ and computes $\textbf{\textit{Collision}}_n$ with success probability at least $p = \min\left\{\frac{t}{n}, 1\right\}\frac{t}{n \log t}$.*

The performance of the described algorithm is asymptotically optimal as discussed in Section 3.

3 The lower bound

The purpose of this section is to show that the four algorithms introduced in Section 2 achieve an asymptotically optimal trade-off between running time and success probability, for all functions t.

By Yao's minimax principle, this task reduces to showing lower bounds for deterministic algorithms working on random input, i.e., the two algorithms of Section 2.1 and Section 2.2. It then follows that also the randomized algorithms cannot be faster. For the sake of completeness, we summarize the results of this section (in particular Lemma 8) in the following theorem. It also states a lower bound on the amount of randomness required.

Theorem 1. *Assume a randomized algorithm $\mathcal{A}$ solves $\textbf{\textit{Collision}}_n$ for all inputs in time t and with positive success probability. Then, with $r = \min\{t, n\}$ and $p \leq p_t = \frac{8r^2}{(n-1)^2 \log(2t)}$, the success probability of $\mathcal{A}$ is at most p and there exists an input where 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 [6, 7, p. 35] and Lemma 8, 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. Any deterministic 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, $2^b \geq 1/p_t$, $b \geq -\log p_t$. □

At the heart of the lower bound is the consideration about a single leaf of the comparison tree that relates the fraction of the input ending at this leaf to the success probability induced by this leaf. This basic trade-off is formulated between success for input from $\mathbf{q}$ versus running time (fraction of input reaching the leaf) of $\mathbf{p}$, which is the worst-case running time of the tree. Transforming this into bounds on the expected running time for $\mathbf{q}$, and taking into account that sublinear algorithms cannot access all the variables, requires some extra work.

3.1 High-probability DAGs

For the purposes of analyzing T, we 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 following the "permutation first" procedure to draw an input with collision. Choose uniformly a permutation π. The corresponding input vector defines a path to a leaf c of T, and π 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 the pivot $o \in \{1, \ldots, n-1\}$ (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 π, which is called the **success probability of the permutation** π 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:

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

The following information theoretic consideration gives a precise relation between the success probability p_c and the number of linear extensions f_c.

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 of the 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 γ. This leads to the claimed bound. □

By Stirling's formula, we get the following lemma.

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

If an algorithm runs in sublinear time, it cannot access all the input elements, leading to a fairly small success probability. We make this precise as a minimum number of accessed variables that is required to achieve a certain success probability.

Lemma 5. *Assume a leaf node c of a decision tree T for* ***Collision***$_n$ *has success probability p_c for* $\mathbf{q}$*. Then the depth d_c of c is at least $d_c \geq \frac{n-1}{2}\sqrt{p_c}$. For $p_c \leq n^{-\frac{1}{6}}$ this implies $d_c \geq \frac{p_c n \log n}{6} - 3p_c n - \frac{\log n}{6} - 1$*

Proof. By the "collision first" procedure to draw an input from $\mathbf{q}$, the pair of variables forming the collision must be accessed, which happens with probability $p_c \leq \frac{r(r-1)}{(n-1)n} < \frac{(2d_c)^2}{n(n-1)}$. This yields $2d_c > \sqrt{p_c n(n-1)} > \sqrt{p_c}(n-1)$, implying $d_c > \frac{n-1}{2}\sqrt{p_c}$.

Now assume $p_c \leq n^{-\frac{1}{6}}$. 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

We want to show a lower bound of $\Omega(pn \log n)$ for the expected running time. As a first step, we consider the running time implied by input without collision (drawn from $\mathbf{p}$). This certainly implies the corresponding lower bound on the worst-case running time, the depth of the tree.

Lemma 6. *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 c of T has success probability p_c, a depth d_c, and a fraction f_c of the permutations from S that end at c. Now, by definition, $D = \sum f_c \cdot d_c$, and $p = \sum f_c \cdot p_c$.

We define two classes of nodes, the high-probability nodes $H = \{c \mid p_c > n^{-\frac{1}{6}}\}$, and the remaining nodes L. Define further for these two classes of permutations in S the split $f_H = \sum_{c\in H} f_c$, and similarly $f_L = \sum_{c\in L} f_c$, 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 $f_H \cdot p_H = \sum_{c\in H} f_c \cdot p_c$, $f_H \cdot D_H = \sum_{c\in H} f_c \cdot d_c$, $f_L \cdot p_L = \sum_{c\in L} f_c \cdot p_c$, $f_L \cdot D_L = \sum_{c\in L} f_c \cdot d_c$. 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 $c \in H$ the relative reach-probability by $f'_c = f_c/f_H$. Note that the f'_c sum to 1, i.e., they form a probability distribution. Define $a_c = 2^{-d_c}$, $A_H = \sum_{c\in H} a_c$, such that the values $a'_c = a_c/a_H$ sum to 1 and form a probability distribution.

With this, we get

$$D_H = -\sum_{c\in H} f'_c \log a_c = -\log a_H - \sum_{c\in H} f'_c \log a'_c \tag{9}$$

$$\geq -\log a_H - \sum_{c\in H} f'_c \log f'_c \tag{10}$$

$$= -\log f_H - \log a_H - \sum_{c\in H} f'_c \log f_c \tag{11}$$

$$\geq \log -f_H + \sum_{c\in H} f'_c \left(\frac{p_c n \log n}{6} - 3p_c n - \frac{\log n}{6} - 1 + \log\frac{|S|}{n!}\right) \tag{12}$$

$$= -\log f_H + \frac{pn}{6}\log n - 3pn - \frac{\log n}{6} - 1 + \log\frac{|S|}{n!}. \tag{13}$$

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 $c \in L$ of T, i.e., with low probability $p_c \leq n^{-\frac{1}{6}}$. By Lemma 5 we have the depth-bound $d_c \geq \frac{p_c n}{6}\log n - 3pn - \frac{\log n}{6} - 1$. This inequality yields $D_L = \sum_{c\in L} f'_c \cdot d_c \geq \sum_{c\in L} f'_c \cdot \left(\frac{p_c n}{6}\log n - 3p_c n - \frac{\log n}{6} - 1\right)$ $\frac{p_L n}{6}\log n - 3p_L n - \frac{\log n}{6} - 1$. Now, the lemma follows by $D = f_H \cdot D_H + f_L \cdot D_L \geq (f_H p_H + f_L p_L)\left(\frac{n}{6}(\log n - 18)\right) - \frac{\log n}{6} - 1 + f_H \log\frac{f_H|S|}{n!} \geq \frac{pn}{6}\log n - 3pn - \frac{\log n}{6} - 1 + \log\frac{|S|}{n!}$. □

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 7. *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 8. *Any comparison tree T with worst-case running time $t \leq n$ has success probability $p \leq p_t = \frac{16t^2}{(n-1)^2\log(2t)}$ when input is drawn from $\mathbf{q}$.*

Proof. With $r = 2t$, by Lemma 7 w.l.o.g. the comparison tree T solves **Collision**$_r$ in worst-case time t and success probability q. From Lemma 6 follows $t \geq \frac{qr}{6}\log r - 3pr - \frac{\log r}{6} - 1$, which yields $q \leq 6(r/2 + \frac{\log r}{6} + 1)/r(\log r - 18) < (3r + \log r + 1)/r\log r < 4/\log r = 4/\log(2t)$. Now, by the argument of Lemma 5 $p \leq \frac{r(r-1)}{n(n-1)}q \leq \frac{16t^2}{(n-1)^2\log(2t)}$. □

3.5 Expected Time for Random Input with Collision

Finally, we can also conclude an asymptotic lower bound of $\Omega(pn \log n)$ for the expected running time when input is drawn according to $\mathbf{q}$. By Yao's Minimax Principle, the same lower bound holds for the expected running time of randomized algorithms on input with collision.

Lemma 9. *Let T be a linear decision tree solving **Collision**$_n$. Assume that the success probability for input drawn according to distribution $\mathbf{q}$ is p, and the expected running time is $C_{\mathbf{q}}$. Then, $C_{\mathbf{q}} \geq \frac{pn}{48}(\log n - 18) - \frac{\log n}{12} - 4$.*

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 6, 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 $p'_S \geq 1/4 \geq p_S/4$. Now, because the expected running time of T' is less than the worst-case running time $2C_S$ of T', Lemma 6 yields $2C_S \geq \frac{p_S n}{4 \cdot 6}(\log n - 18) - \frac{\log n}{6} - 1 + \log f_S$. It remains to take the weighted sums of the bounds on $2C_S$ and $2C_{\bar{S}}$, yielding $2C \geq \frac{pn}{4 \cdot 6}(\log n - 18) - \frac{\log n}{6} - 2$. Here, the last term stems from $f_S \log f_S + (1 - f_S)\log(f_S - 1) > -1$. □

Acknowledgment

I would like to thank Ueli Maurer and Dominik Raub for introducing me to the problem and for several fruitful discussions, and an anonymous referee for suggestions improving the introduction.

References

1. Ben-Or, M.: Lower bounds for algebraic computation trees. In: Proc. 15th Annual ACM Symposium on Theory of Computing. (1983) 80–86
2. Boppana, R.B.: The decision-tree complexity of element distinctness. Inf. Process. Lett. **52**(6) (1994) 329–331
3. Buhrman, H., Dürr, C., Heiligman, M., Høyer, P., Magniez, F., Santha, M., de Wolf, R.: Quantum algorithms for element distinctness. SIAM J. Comput. **34**(6) (2005) 1324–1330 (electronic)
4. Aaronson, S., Shi, Y.: Quantum lower bounds for the collision and the element distinctness problems. J. ACM **51**(4) (2004) 595–605 (electronic)

5. Maurer, U.M.: Abstract models of computation in cryptography. In Smart, N.P., ed.: IMA Int. Conf. Volume 3796 of Lecture Notes in Computer Science., Springer (2005) 1–12
6. Yao, A.C.C.: Probabilistic computations: towards a unified measure of complexity. In: Proc. 18th FOCS, IEEE (1977) 222–227
7. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge, UK (1995)
8. Snir, M.: Lower bounds on probabilistic linear decision trees. Theoret. Comput. Sci. **38**(1) (1985) 69–82
9. Grigoriev, D., Karpinski, M., Meyer auf der Heide, F., Smolensky, R.: A lower bound for randomized algebraic decision trees. Comput. Complexity **6**(4) (1996/97) 357–375
10. Manber, U., Tompa, M.: Probabilistic, nondeterministic, and alternating decision trees (preliminary version). In: STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing, New York, NY, USA, ACM Press (1982) 234–244
11. Manber, U., Tompa, M.: The complexity of problems on probabilistic, nondeterministic, and alternating decision trees. J. Assoc. Comput. Mach. **32**(3) (1985) 720–732
12. Manber, U., Tompa, M.: The effect of number of Hamiltonian paths on the complexity of a vertex-coloring problem. SIAM J. Comput. **13**(1) (1984) 109–115
13. Chein, M., Habib, M.: The jump number of DAGs and posets: An introduction. Annals of Discrete Mathematics **9** (1980) 189–194
14. Blum, M., Floyd, R., Pratt, V., Rivest, R., Tarjan, R.: Time bounds for selection. Journal of Computer and System Sciences **7** (1973) 448–461