Sorting and Searching
With a Faulty
Comparison Oracle

Philip M. Long

UCSC-CRL-92-15
November 9, 1992

Board of Studies in Computer and Information Sciences
University of California at Santa Cruz
Santa Cruz, CA 95064

ABSTRACT

We study sorting and searching using a comparison oracle that "lies." First, we prove that an algorithm of Rivest, Meyer, Kleitman, Winklmann and Spencer for searching in an \( n \)-element list using a comparison oracle that lies \( E \) times requires at most \( O(\log n + E) \) comparisons, improving the best previously known bound of \( \log n + E\log\log n + O(E\log E) \). A lower bound, easily obtained from their results, establishes that the number of comparisons used by their algorithm is within a constant factor of optimal.

We apply their search algorithm to obtain an algorithm for sorting an \( n \) element list with \( E \) lies that requires at most \( O(n\log n + En) \) comparisons, improving on the algorithm of Lakshmanan, Ravikumar and Ganesan, which required at most \( O(n\log n + En + E^2) \) comparisons. A lower bound proved by Lakshmanan, Ravikumar and Ganesan establishes that the number of comparisons used by our sorting algorithm is optimal to within a constant factor.

* I gratefully acknowledge the support of a UCSC Chancellor’s dissertation-year Fellowship. Email address: plong@cs.ucsc.edu.
1 Introduction

Rivest, Meyer, Kleitman, Winklmann, and Spencer [RMK+80] described an algorithm for finding an element \( k \) in \( \{1, \ldots, n\} \) using questions of the form "is \( k \leq a^2 \)" for \( a \) chosen by the algorithm, when up to \( E \) of their algorithm’s questions were answered incorrectly. They showed that the algorithm was guaranteed to output \( k \), and that their algorithm used at most

\[
\log n + E \log \log n + O(E \log E)
\]

comparisons. In this note, we show that the number of comparisons required by their algorithm is at most

\[
O(\log n + E),
\]

improving on their bound when \( E \) grows faster than \((\log n) / (\log \log n)\).

A trivial application of their results provides a lower bound of \( \Omega(\log n + E) \), establishing that their algorithm is optimal to within a constant factor.

We may easily apply this searching bound to show that an insertion sort algorithm, which uses their algorithm as a subroutine to determine where each insertion is to take place, requires at most \( O(n \log n + En) \) comparisons to sort \( n \) keys with \( E \) "lies," improving on the bound of \( O(n \log n + En + E^2) \) proved by Lakshmanan, Ravikumar and Ganesan [LRG91] for a closely related algorithm when \( E \) grows faster than \( n \). A lower bound of \( \Omega(n \log n + En) \) comparisons proved in that paper establishes the fact that our modification of their algorithm is within a constant factor of optimal.

The [RMK+80] paper contained a detailed proof of the following theorem, which is the starting point of our analysis.

**Theorem 1 ([RMK+80]):** For any nonnegative integer \( E \) and positive integer \( n \), let \( Q(n, E) \) denote the number of comparison questions necessary in the worst case to identify an unknown \( k \in \{1, \ldots, n\} \) when up to \( E \) of the questions may receive an erroneous answer. Then

\[
\min \left\{ u : 2^n \geq n \sum_{i=0}^{E} \binom{n}{i} \right\} \leq Q(n, E) \leq \min \left\{ u : 2^n - E \geq n \sum_{i=0}^{E} \binom{n}{i} \right\}.
\]

Our improvement on their upper bound is obtained by applying an unusual approximation to \( \sum_{i=0}^{d} \binom{m}{i} \). It is a direct consequence of Hoeffding's inequality, a bound on the probability that the following two quantities differ by much:

- The probability that a (biased) coin will come up heads,
- The fraction of the time it comes up heads when flipped \( m \) times.

Their approximation improves on the usual approximation of \((\epsilon m / d)^d \) [BEHW89] when \( d \) is large relative to \( m \), which is useful for this application.

Note that if \( E \in \Omega(\log n) \), the \( En \) term in our sorting bound of \( O(n \log n + En) \) dominates. This is especially interesting in light of the result of Ravikumar, Ganesan and Lakshmanan [RGIL87], which says that \((E + 1)n - 1\) comparisons are necessary and

---

1 In this paper, we follow usual convention of denoting the base 2 logarithm by \( \log \) and the natural logarithm by \( \ln \).

2 Bounds of \( O(\log n + E) \) on the first minimum of this theorem were obtained independently by Cesa-Bianchi and Warmuth [CW92] while working on another application.
sufficient to simply find the maximum of \( n \) elements using a comparison oracle that lies \( E \) times. Thus, if \( E \in \Omega(\log n) \), only a constant factor more comparisons are required to sort \( n \) elements than to simply output their maximum. It is also perhaps worth repeating the observation in [LRG91] that any \( O(n \log n) \) sorting algorithm can be trivially modified to cope with \( E \) lies by repeating each comparison \( 2E + 1 \) times, obtaining an algorithm that uses \( O(En \log n) \) comparisons. Thus, for moderately large \( E \), our sorting result can be viewed as knocking off a log factor from what can be obtained trivially.

For those familiar with the Computational Learning Theory literature, as noted by Goldman, Rivest and Schapire [GRS89], another interpretation of the sorting problem is as the problem of learning a total order on \( n \) elements using Angluin’s “membership queries” [Ang88]. Our sorting result can therefore be interpreted as determining (to within a constant factor) the number of membership queries required for learning a total order, when a bounded number of the membership queries are answered incorrectly.

In addition to the aforementioned previous work, sorting and searching with a faulty comparison oracle has been studied under at least two other assumptions about the generation of the faults, including that they are generated independently at random [Pel89, FPRU90], and that there is a constant \( r \) such that for each \( i \), at most \( ir \) of the first \( i \) comparisons are answered incorrectly [Pel89, AD91].

2 Approximating \( \sum_{i=0}^{d} \binom{m}{i} \)

In this section, we state and, for completeness, prove, a useful approximation to \( \sum_{i=0}^{d} \binom{m}{i} \).

The following form of the Hoeffding bounds will be useful.

**Theorem 2 (c.f., [Pol84]):** Let \( Y_1, \ldots, Y_m \) be independent, identically distributed \( \{0,1\} \)-valued random variables such that for each \( i \), \( \Pr(Y_i = 1) = p \). Then, for any \( \alpha \geq 0 \),

\[
\Pr \left( \frac{1}{m} \sum_{i=1}^{m} Y_i \leq p - \alpha \right) \leq e^{-2\alpha^2 m}.
\]

The following Corollary is the useful approximation.

**Corollary 3:** Choose \( d, m \in \mathbb{N} \), \( d \leq m/2 \). Then

\[
\sum_{i=0}^{d} \binom{m}{i} \leq 2^m \exp \left( -\frac{(m - 2d)^2}{2m} \right).
\]

**Proof:** Since if an unbiased coin is flipped independently \( m \) times (call the results of the flips \( Y_1, \ldots, Y_m \)), any subset of the \( m \) tosses is equally likely to be the set of trials in which heads appeared,

\[
\frac{1}{2^m} \sum_{i=0}^{d} \binom{m}{i} = \Pr \left( \frac{1}{m} \sum_{i=0}^{m} Y_i \leq \frac{d}{m} \right) \leq \exp(-2(1/2 - d/m)^2 m),
\]

applying Theorem 2 with \( p = 1/2 \). Multiplying both sides by \( 2^m \) and simplifying yields the desired result. \( \square \)
3 Searching with a faulty comparison oracle

In the section, we present our main result.

**Theorem 4:** For any nonnegative integer $E$ and positive integer $n$, let $Q(n, E)$ denote the number of comparison questions necessary in the worst case to identify an unknown $k \in \{1, ..., n\}$ when up to $E$ of the questions may receive an erroneous answer. Then

$$Q(n, E) = O(\log n + E).$$

**Proof:** First, note that

$$\min \left\{ u : 2^{u-E} \geq n \sum_{i=0}^{E} \binom{u-E}{i} \right\} \leq \min \left\{ u : 2^u \geq n \sum_{i=0}^{E} \binom{u}{i} \right\} + E.$$

Thus, by Theorem 1, a bound of $u + E$ holds for any $u$ such that

$$2^u \geq n \sum_{i=0}^{E} \binom{u}{i}.$$

Fix $E$ and $n$. Let

$$u = \lceil 4 \max\{2\ln n, E\} \rceil.$$  \hspace{1cm} (3.1)

In particular,

$$u \geq 8 \ln n$$
$$\sqrt{u} \geq 2 \sqrt{2 \ln n}$$
$$u \geq 2 \sqrt{2 u \ln n}.$$

Returning to (3.1), we may obtain the following sequence of inequalities,

$$u \geq 2 \max\{2 \sqrt{u \ln n}, 2 E\}$$
$$u \geq 2 \sqrt{u \ln n} + 2 E$$
$$(u - 2 E)^2 \geq 2 u \ln n$$
$$\frac{(u - 2 E)^2}{2 u} \geq \ln n$$
$$\exp\left(\frac{(u - 2 E)^2}{2 u}\right) \geq n$$
$$\exp\left(\frac{-(u - 2 E)^2}{2 u}\right) \leq 1/n.$$

Applying Corollary 3, we obtain

$$\frac{1}{2^u} \sum_{i=0}^{E} \binom{u}{i} \leq 1/n$$
$$n \sum_{i=0}^{E} \binom{u}{i} \leq 2^u.$$
Applying Theorem 1, this implies that the number of comparisons used by their algorithm is at most

\[ u + E = \lceil 4\max\{2\ln n, E\} \rceil + E \leq 8\ln n + 5E + 1, \]

completing the proof. \Box

Next, we turn to lower bounds.

**Theorem 5:** For any nonnegative integer \( E \) and positive integer \( n \geq 2 \), let \( Q(n, E) \) denote the number of comparison questions necessary in the worst case to identify an unknown \( k \in \{1, \ldots, n\} \) when up to \( E \) of the questions may receive an erroneous answer. Then

\[ Q(n, E) = \Omega(\log n + E). \]

**Proof:** Fix \( E \) and \( n \). Note that

\[ 2^n - n \sum_{i=0}^{E} \binom{u}{i} \]

is an increasing function of \( u \), and therefore, by Theorem 1 that any \( u \) for which

\[ 2^n < n \sum_{i=0}^{E} \binom{u}{i} \]

provides a lower bound on \( Q(n, E) \). Let \( u = \lfloor \log n \rfloor + E \). Then

\[ 2^n \leq n2^{E} = n \sum_{i=0}^{E} \binom{E}{i} < n \sum_{i=0}^{E} \binom{\lfloor \log n \rfloor + E}{i} \quad \text{(since } n \geq 2) \]

\[ = n \sum_{i=0}^{E} \binom{u}{i}. \]

This completes the proof. \Box

## 4 Sorting with a faulty comparison oracle

In this section, we describe how to apply the algorithm of the previous section to obtain a sorting algorithm that copes with incorrect answers to comparison questions, and requires a number of comparisons that is within a constant factor of optimal.

We begin by describing a modification of binary insertion sort that uses the robust binary search algorithm of \( [\text{RMK}^+80] \) to determine where to insert. Pseudo-code for this algorithm is given in Figure 4.1.

The following follows trivially from the results of the previous section.
algorithm robust-insertion-sort(A,E,n)
array A; (n elements in A)
integer E;
integer n;

for i = 2 to n
    begin
        use [RMK+80] to determine where A[i] should be
            inserted in A[1],...,A[i-1], assuming at most E lies
            (during this search), say it is before A[k];
        insert A[i] before A[k];
    end;

Figure 4.1: Pseudo-code for a robust sorting algorithm.

**Theorem 6:** The algorithm robust-insertion-sort correctly sorts an array of n elements when at most E of its comparison questions are answered incorrectly, using

\[ O(n \log n + En) \]

comparisons.

The following theorem, due to Lakshmanan, Ravikumar and Ganesan, establishes that the number of comparisons used by robust-insertion-sort is within a constant factor of optimal.

**Theorem 7 ([LRG91]):** Any correct algorithm for sorting n keys, when up to E comparisons may be answered incorrectly, must make

\[ \Omega(n \log n) + E(n - 1) \]

comparisons.

5 Acknowledgements

We’d like to thank Nicolo Cesa-Bianchi, Max Cooperman, Dave Helmbold, Hans Ulrich Simon, K.B. Sriram, Madhukar Thakur and Manfred Warmuth for valuable conversations about this research and related topics.

References


