The Bayesian Learner is Optimal for Noisy Binary Search (and Pretty Good for Quantum as Well) *

Michael Ben Or † Avinatan Hassidim ‡

February 5, 2009

Abstract

We use a Bayesian approach to optimally solve problems in noisy binary search. We deal with two variants:

- Each comparison is erroneous with independent probability $1-p$.
- At each stage k comparisons can be performed in parallel and a noisy answer is returned.

We present a (classical) algorithm which solves both variants optimally (with respect to p and k), up to an additive term of $O(\text{loglog}\, n)$, and prove matching information-theoretic lower bounds. We use the algorithm to improve the results of Farhi et al. [12], presenting an exact quantum search algorithm in an ordered list of expected complexity less than $(\log_2 n)/3$.

1 Introduction

Noisy binary search has been studied extensively (see [25, 26, 22, 2, 11, 5, 13, 3, 19, 20, 21, 23]). In the basic model we attempt to determine a variable $s \in \{1, \ldots, n\}$ by queries of the form "$s > a$"? for any value a. In the noisy model, we get the correct answer with probability p, with independent errors for each query. In the adversarial model, an opponent chooses which queries are answered incorrectly, up to some limit. Our work focuses on the noisy non-adversarial model.

Generalizing noiseless binary search to the case when k questions can be asked in parallel is trivial: recursively divide the search space into $k+1$ equal sections. This model, and its noisy variant, are important (for example) when one can send a few queries in a single data packet, or when one can ask the second query before getting an answer to the first.

*Research supported by the Israel Science Foundation.

†Incumbent of the Jean and Helena Alfassa Chair in Computer Science. benor@cs.huji.ac.il, The Hebrew University, Jerusalem, Israel

‡avinatanh@gmail.com, The Hebrew University, Jerusalem, Israel

1.1 Previous Results

The problem of binary search with probabilistic noise was first introduced by Rény [25], but for a stronger type of queries. Ulam [28] restated this problem, allowing only comparisons. An algorithm for solving Ulam's game was first proposed by Rivest et. al. in [26]. They gave an algorithm with query complexity $O(\log n)$ which succeeds if the number of adversarial errors is constant. Following their results, a long line of researchers have tried to handle a constant fraction r of errors. Dhagat, Gács and Winkler [11] showed that this is impossible for $r \geq 1/3$, and gave an $O(\log n)$ algorithm for any $r < 1/3$. The constant in the O notation was improved by Pedrotti in [21], to $\frac{8 \ln 2}{3} \frac{\log n}{(1-3r)^2(1+3r)}$. Another variant of the adversarial problem is the Prefix-Bounded model. In this model, any initial sequence of i answers has at most ri adversarial mistakes for some constant r. Borgstrom and Kosaraju [5] gave an $O(\log n)$ algorithm for any $r < 1/2$ fraction of errors for this case.

Assuming probabilistic noise, Feige et. al. showed [13] that one can perform binary search using $\Theta(\log n/(1 - H(p)))$ queries, where $H(p)$ is the entropy function. The algorithm proceeds by repeating every query many times to obtain a constant error probability, and then traverses the search tree, backtracking when needed. This leads to constants which are too large for our applications, and has no easy generalization when multiple queries are made simultaneously (the batch learning model). Aslam showed a reduction of probabilistic errors to an adversarial Prefix-Bounded model [3]. Aslam's algorithm has the same multiplicative factor that arises in the adversarial algorithm, and might not be applicable to generalizations of noisy search.

Noisy binary search has also been defined in the k-batch model (see Orr [20] for applications of batch learning in a more general model), but much less is known there. Cicalese, Mundici and Vaccaro [9] gave an optimal solution for a constant number of adversarial errors, and two batches of queries. We are not aware of any results regarding the probabilistic batch model. For an extensive survey on the subject see Pelc [23], who also states the k-batch model (in probabilistic and adversarial flavors) as an important open problem.

Quantum Binary Search An equivalent formulation of binary search is giving the algorithm oracle access to a threshold function $f_s(x)$ for some unknown s. Applying $f_s(x)$ returns 1 if $x > s$, and 0 otherwise. The algorithm can apply the function on inputs x (in a noisy manner), and its goal is to find s. This model can be generalized by turning the oracle into a quantum one. Formally, the algorithm is given access to an oracle O_s for some fixed but unknown s, where $O_s|x,t\rangle = |x, f_s(x) \oplus t\rangle$. Determining the exact complexity of quantum binary search is an interesting open problem.

Farhi et al. presented in [12] two quantum algorithms for searching an ordered list. They first presented a "greedy" algorithm with small error probability that clearly outperformed classical algorithms. However, they could not analyze its asymptotic complexity, and therefore did not use it. Instead, they devised another algorithm, which can find the correct element in a sorted list of length 52 using just 3 queries. Applying this recursively gives a $0.53 \log_2 n$ quantum search algorithm. This was later improved by Jacokes, Landahl and Brooks [17] by searching lists of 434 elements using 4 queries. Another improvement by Childs, Landahl and Parrilo [7] enables searching lists of 605 elements using 4 comparisons, and gives a query complexity of $0.433 \log_2 n$ queries.

We note that these algorithms are exact. Since Farhi et al.'s greedy algorithm has small error probability, iterating it on a fixed size list results in a noisy binary search algorithm. However, without an exact analysis of noisy binary search, the resulting bounds are not strong enough.

The lower bounds for binary search were first treated by Ambainis, who showed [1] that quantum binary search has complexity $\Omega(\log n)$. Using the quantum adversary method of Ambainis, but applying unequal weights to different inputs, Hoyer, Neerbek and Shi gave a lower bound of $\frac{1}{\pi}\ln(n) \approx 0.202 \log n$ queries [16]. Childs and Lee showed that using the generalized adversary method cannot improve this by much [8].

1.2 Our Results

Intuitively, our algorithm, for the classical binary search problem, always asks the query which yields the maximal amount of information. This is done using a Bayesian learner which tries to determine the place of the element we are looking for. Usually, myopic learning algorithms are not optimal, but in this case we show that greedy behavior is in fact optimal.

We give an informal description of the algorithm. Assume that s is chosen uniformly from the list. Choose an element x in the list such that $\Pr(x > s) \approx 0.5$. Compare x and s, and give the updated Bayesian probabilities to all the elements in the list. Repeat the process until one element has relatively high probability, and then test its surroundings recursively. Note that if $p = 1$ then the algorithm performs standard binary search - after the first step half the elements have probability zero, and the rest will have uniform distribution.

Letting $I(p) = 1 - H(p)$, each noisy query provides at most $I(p)$ bits of information. Thus the best time bound we can hope for is $(1-\epsilon)\log n / I(P)$, where the factor $(1-\epsilon)$ comes from Shanon. We show

Theorem 1.1. *There exists a (classical) algorithm which finds s in a sorted list of n elements with probability $1-\delta$ using an expected*

$$(1-\delta)\left[\frac{\log n}{I(p)} + O\left(\frac{\textit{loglog } n}{I(p)}\right) + O\left(\frac{\log(1/\delta)}{I(p)}\right)\right]$$

noisy queries, where each query has (independent) probability $p > 1/2$ of being answered correctly. This is tight up to an additive term which is polynomial in loglog n, $\log(1/\delta)$ and $I(p)$.

We present a similar Bayesian strategy when we are allowed to perform composite comparisons. In this model, we choose a constant number of consecutive segments, and the oracle tells us in which segment is the special element (binary search uses two segments each query). When the answer given by the oracle is noisy, a generalization of the noisy binary search algorithm remains optimal (see Subsection 3).

A surprising application of this classical noisy search algorithm is a faster quantum algorithm for binary search. Using the generalized variant, when we can divide the array into a set of segments, we can recursively use the greedy quantum binary search of Farhi et al. [12]. Measuring after r queries in their algorithm corresponds to sampling

the intervals according to a probability distribution which is concentrated near the correct interval. If the entropy of this distribution over the k equal probability intervals is H_r, then the average information is $I_r = \log(k) - H_r$, and the expected number of queries is $\frac{r \cdot \log n}{I_r}$. Using this we show:

Theorem 1.2. *The expected quantum query complexity of searching an ordered list without errors is less than* $0.32 \log n$.

We prove a quantum lower bound, which shows that quantum algorithms faced with a noisy oracle cannot be much better than their classical counterparts, by showing that they require at least $\frac{\ln(n)}{\pi\sqrt{p(1-p)}} \approx 0.202 \frac{\log n}{\sqrt{p(1-p)}}$ queries. Allowing the quantum algorithm to err with probability δ reduces the complexity of the upper bound and the lower bound by the same classical factor $(1 - \delta)$.

1.3 Applications of Noisy Binary Search

Practical uses for optimal noisy search can occur (for example) in biology. This is especially important when each "noisy comparison" is a biological experiment, which is being used to find the value of some quantity, by comparing it to different thresholds. Experiments have an error probability, and performing them can be very time consuming. One example for this scenario is trying to determine the supermolecular organization of protein complexes and isolating active proteins in their native form [27, 14]. In both cases, the 3-dimensional conformation of the proteins should be conserved, and solubilization methods are based on different percentages of mild detergents. Determining the right percentage can be done by noisy binary search, running a gel for each query.

The algorithm has theoretical applications as well. For example, it can be used to achieve the results of Karp and Kleinberg [18]. Our main application is to obtain better bounds on the query complexity of quantum binary search.

2 Classical Noisy Search

2.1 Problem Setting

Let $x_1 \geq \ldots \geq x_n$ be n elements, and assume we have a value s such that $x_1 \geq s \geq x_n$, and want to find i such that $x_i \geq s \geq x_{i+1}$. Comparing x_i and s is done with $f(i) \to \{0, 1\}$, which returns 1 if $x_i > s$ and 0 if $x_i \leq s$. Each evaluation of f returns the correct answer with probability $p > 1/2$. Note that calculating f twice at the same place may return different answers.

In this noisy environment, we must let our algorithm err. We bound the error probability by a given $\delta > 0$.

2.2 Algorithm

The algorithm uses an array of n cells $a_1, \ldots, a_n$, where a_i denotes the probability that $x_i \geq s \geq x_{i+1}$. The initialization of the array is $a_i = 1/n$, as if we could assume

that we have a uniform prior for s. Each step, the algorithm chooses an index i such that $t = \sum_{j=1}^{i} a_j < 0.5$ but $\sum_{j=1}^{i+1} a_j \geq 0.5$ and compares s to x_i. According to the result of the comparison (which could be wrong) the algorithm *updates* the probabilities $a_1, \ldots, a_n$. If the result of the comparison was that $x_i > s$, we multiply a_j for $j \leq i$ by $(1-p)$, multiply a_j for $j > i$ by p and normalize so that the values $a_1, \ldots, a_n$ sum up to 1. The normalization depends on the sum $t = \sum_{j=1}^{i} a_j$. Assuming the result of the comparison was $x_i > s$, the normalization is[1]

$$a_j = \begin{cases} \frac{(1-p)a_j}{(1-p)t+p(1-t)}, j \leq i \\ \frac{pa_j}{(1-p)t+p(1-t)}, j > i \end{cases}$$

If the result was that $x_i \leq s$, the normalization is $pt + (1-p)(1-t)$. Note that if $|t - 1/2|$ is small, as will be the case in our algorithm, the normalization is roughly a multiplication by 2. In order to use this idea we need to address two technical issues:

1. It is not always possible to find an element such that $\Pr(x_i > s) = 1/2$. Therefore, we use a constant called ϵ_{par} ("par" stands for partition) which is an upper bound for $|\sum_{j=1}^{i} a_j - 1/2| = |t - 1/2| \leq \epsilon_{par}$. Its value will be chosen later.

2. Given the bound ϵ_{par}, we may not be able to partition the array if there is an element x_i such that $a_i > \epsilon_{\text{par}}$. However, if a_j is not much bigger than ϵ_{par} for any j, we can not yet find s with success probability $1-\delta$, as ϵ_{par} is too small. Instead, we show that with high probability, there are at most l_{sur} (for "surroundings") elements between x_i and s. We can then iterate the algorithm, this time searching the elements $x_{i-l_{\text{sur}}}, \ldots, x_{i+l_{\text{sur}}}$. Making sure l_{sur} is $O(\text{polylog}(n))$ gives the right running time, adding the additive $O(\text{loglog } n)$ term.

The exact values for l_{sur} will be chosen later.

[1] Previous noisy search algorithms have already used weights, see for example [26, 5, 18]. However, we choose weights optimally, and use information even when p is very close to $1/2$ (see for example the usage of ϵ_{good} in [18]). This gives us better results, and enables optimal generalization to the batch model.

1. *Halt condition* If the number of elements is smaller than $O(\log\log n)$, search the array using [13].
2. Let $a_i = \max_j a_j$, breaking ties arbitrarily. If $a_i \geq \epsilon_{\text{par}} = (24\log n)^{-1/2}$:
   - Check using $O((\log(1/\delta) + \text{loglog } n)/I(p))$ repeated comparisons on both sides if $x_{i-l_{\text{sur}}} \geq s \geq x_{i+l_{\text{sur}}}$.
     (a) If it is in the surroundings, search $x_{i-l_{\text{sur}}} \ldots x_{i+l_{\text{sur}}}$ recursively.
     (b) Else (s is not there), restart the algorithm, from scratch, searching n elements
3. Else, find an index i such that
$$1/2 - \epsilon_{\text{par}} \leq \sum_{j=1}^{i} a_j < 1/2$$
4. Compare s and x_i; update the probabilities. Go to 2.

Algorithm 1, for $\delta \leq 1/\log n$

Theorem 2.1. *The expected query complexity of Algorithm 1 is*

$$\frac{\log n}{I(p)} + O\left(\frac{\textit{loglog } n}{I(p)}\right) + O\left(\frac{\log(1/\delta)}{I(p)}\right) \tag{1}$$

The proof will occupy the rest of this section. We note, in passing, that the lower bound is very similar to the upper bound; see 2.8.

Assuming Theorem 2.1, we now prove Theorem 1.1.

Proof. Remember Theorem 1.1 gives query complexity of

$$(1-\delta)\left[\frac{\log n}{I(p)} + O\left(\frac{\text{loglog } n}{I(p)}\right) + O\left(\frac{\log(1/\delta)}{I(p)}\right)\right]$$

If $\delta < 1/\log n$, the difference between the bounds in Theorems 1.1, 2.1 is absorbed by the big$-O$ notation of the low order terms.

If $\delta > 1/\log n$, we modify the algorithm as follows: with probability $c = \delta - 1/\log n$ we choose a random element (i.e., we fail immediately); with probability $1-c$, run Algorithm 1 with $\delta = 1/\log n$. The failure probability is $c + \frac{1-c}{\log n} < c + \frac{1}{\log n} = \delta$ and the expected query complexity is

$$\begin{aligned}
&(1-c)\left[\frac{\log n}{I(p)} + O\left(\frac{\text{loglog }(n)}{I(p)}\right) + O\left(\frac{\log\log n}{I(p)}\right)\right] \\
&= \left[1 - \delta + \frac{1}{\log n}\right]\left[\frac{\log n}{I(p)} + O\left(\frac{\text{loglog }(n)}{I(p)}\right)\right]
\end{aligned}$$

$$(1-\delta)\left[\frac{\log n}{I(p)} + O\left(\frac{\text{loglog } n}{I(p)}\right) + O\left(\frac{\log(1/\delta)}{I(p)}\right)\right]$$

Again the O notation in the low order terms gives the bound. □

Uniform Prior. In order to simplify the presentation, the statement of the algorithm is worst case, and does not use any prior information on the place of s. To simplify the analysis, we show a reduction to the case where the *a priori* distribution is uniform, following [12]. Our problem is equivalent to the following: given a monotone nondecreasing boolean array A, find its first 1 by querying elements. Pick k randomly and uniformly between 1 and n. Define a new array B of length n as follows:

$$B[i] = \begin{cases} A[i+k], & i+k \leq n \\ 1 - A[i+k-n], & i+k > n \end{cases}$$

The transition point in B is uniformly distributed, since k is. Apply the algorithm to the array B (it is easy to see how to translate queries of B into queries of A). From this we can find the transition point of B, and deduce the transition point of A. Note that B must be monotone; however, it can be either increasing or decreasing. To distinguish these two cases, we start with $O(\log(1/\delta)/I(p))$ queries of $B[1]$; this reduces the error probability sufficiently so as not to impact the overall error probability, and the query cost is swallowed by the big-O term in Theorem 1.1.

It is also possible to reduce the problem to the uniform prior case using by searching in an array of length $2n$. This is done by extending the functions as in [12]. In this reduction, we search for a transition point, and the direction is not important.

2.3 Analysis of the Algorithm

The following lemma is immediate:

Lemma 2.2. *If the algorithm reached Step (3) then there is an index i such that $1/2 - \epsilon_{par} \leq \sum_{j=1}^{i} a_j < 1/2$.*

We now need to prove two main claims—that the algorithm terminates quickly, and that when it does, s will, with high probability, be near i. The first claim is stated as Lemma 2.5 and is based on Lemma 2.3 and Lemma 2.4. To state these lemmas we need to use the entropy function $H(a_1, \ldots, a_n) = \sum_{i=1}^{n} -a_i \log(a_i)$ and the information function $I(a_1, \ldots a_n) = \log n - H(a_1, \ldots a_n)$. From the convexity of entropy, it follows that:

Lemma 2.3. *If $\forall i$, $a_i < \epsilon_{par}$ then $H(a_1, \ldots, a_n) \geq \log(1/\epsilon_{par})$.*

This means that if $H(a_1, \ldots, a_n) < \log(1/\epsilon_{\text{par}})$ then $\exists i : a_i \geq \epsilon_{\text{par}}$.

Lemma 2.4. *Let $a_1, \ldots, a_n$ be the probabilities before the comparison in step 4, and $b_1, \ldots, b_n$ be the updated probabilities after the comparison. Then:*

$$E[I(b_1, \ldots, b_n)] - I(a_1, \ldots, a_n) \geq I(p) - 4\epsilon_{par}^2(1-2p)^2$$

and taking $\epsilon_{par} = (1/24 log n)^{-1/2}$, this is at least $I(p)(1 - \frac{1}{3 \log n})$.

Proof. Assume that the partition was between k and $k+1$. Call the result of the comparison r, i.e., $r = 0$ if the result was that $x_k > s$, and $r = 1$ otherwise. Define $t = \sum_{i=1}^{k} a_i$, and let $\alpha = \frac{1}{pt+(1-p)(1-t)}$ be the normalization constant used by the algorithm if $r = 0$. We look at the information in this case:

$$I(b_1, \ldots, b_n | r = 0) = \log n + \sum_{i=1}^{k} \alpha p \cdot a_i \log(\alpha p \cdot a_i) + \sum_{i=k+1}^{n} \alpha(1-p) \cdot a_i \log(\alpha(1-p) \cdot a_i)$$

Analyzing the first sum:

$$\sum_{i=1}^{k} \alpha p \cdot a_i \log(\alpha p \cdot a_i) s =$$

$$\alpha p \log(\alpha p) \sum_{i=1}^{k} a_i + \alpha p \sum_{i=1}^{k} a_i \log(a_i) = \alpha p t \log(\alpha p) - \alpha p H(a_1, \ldots, a_k)$$

Substituting into the original equation:

$$\begin{aligned} &I(b_1, \ldots, b_n | r = 0) = \\ &\quad \log n + \alpha p t \log(\alpha p) - \alpha p H(a_1, \ldots, a_k) + \\ &\quad \alpha(1-p)(1-t) \log(\alpha(1-p)) - \alpha(1-p) H(a_{k+1}, \ldots, a_n) \end{aligned}$$

To analyze the expected information gain, we need to know the distribution of r. Fortunately, $\Pr(r = 0) = pt + (1-p)(1-t) = 1/\alpha$. When $r = 1$ the calculation is similar, but the normalization factor changes to $\beta = \frac{1}{p(1-t)+(1-p)t}$. Thus,

$$E[I(b_1, \ldots, b_n)] = I(b_1, \ldots, b_n | r = 0)/\alpha + I(b_1, \ldots, b_n | r = 1)/\beta$$

Calculating:

$$\begin{aligned} &I(b_1, \ldots, b_n | r = 0)/\alpha \\ &\quad = \log n/\alpha + pt \log(\alpha) - tp \log(p) \\ &\quad + pH(a_1, \ldots, a_k) + (1-p)(1-t) \log(\alpha) \\ &\quad + (1-t)(1-p) \log(1-p) - (1-p) H(a_{k+1}, \ldots, a_n) \end{aligned}$$

Noting the normalization sums to one,

$$1/\alpha + 1/\beta = tp + (1-p)(1-t) + p(1-t) + (1-p)t = 1$$

We have:

$$\begin{aligned} &I(b_1, \ldots, b_n | r = 0)/\alpha + I(b_1, \ldots, b_n | r = 1)/\beta \\ &\quad = \log n - H(p) - H(a_1, \ldots, a_n) + pt \log(\alpha) \\ &\quad + (1-p)(1-t) \log(\alpha) + p(1-t) \log(\beta) + (1-p)t \log(\beta) \end{aligned}$$

So the expected information increase after the query is

$$pt\log(\alpha) + (1-p)(1-t)\log(\alpha) + p(1-t)\log(\beta) + (1-p)t\log(\beta) - H(p)$$

We will soon simplify this further and choose a value for ϵ_{par} to make it close enough to $I(p)$. However, we already showed that expected increase does not depend on the actual values of $a_1, \ldots, a_n$, or on the information before the query, other than how balanced the partition is (given by t). Now:

$$\begin{aligned} &pt\log(\alpha) + (1-p)(1-t)\log(\alpha) + p(1-t)\log(\beta) + (1-p)t\log(\beta) \\ &\quad = (pt + (1-p)(1-t))\log(\alpha) + (p(1-t) + (1-p)t)\log(\beta) \\ &\quad = -(1/\alpha)\log(1/\alpha) - (1/\beta)\log(1/\beta) = H(1/\alpha) \end{aligned}$$

We now need to bound $H(1/\alpha)$. For an ideal partition $t = 1/2$ we will have $H(1/\alpha) = 1$, and the expected information increase in each query would be $I(p)$, which is optimal. However, t deviates from $1/2$ by at most ϵ_{par}, and we should now choose ϵ_{par} small enough to get the desired runtime. As $t \geq 1/2 - \epsilon_{\text{par}}$, we have

$$\begin{aligned} H(1/\alpha) &\geq H(p + 1/2 + \epsilon_{\text{par}} - 2p(1/2 + \epsilon_{\text{par}})) \\ &= H(1/2 + \epsilon_{\text{par}}(1-2p)) \geq 1 - 4\epsilon_{\text{par}}^2(1-2p)^2 \end{aligned}$$

where the last inequality follows from the fact that for $x \in [-1/2, 1/2]$ we have $1 - 2x^2 \geq H(1/2 + x) \geq 1 - 4x^2$. Manipulating this inequality gives $x^2 < \frac{1-H(1/2+x)}{2}$. Using this and substituting $\epsilon_{\text{par}} \leq (1/24\log n)^{-1/2}$,

$$\begin{aligned} 4\epsilon_{\text{par}}^2(1-2p)^2 &= 16\epsilon_{\text{par}}^2(p-1/2)^2 \leq \frac{16(p-1/2)^2}{24\log n} \\ &= \frac{2(p-1/2)^2}{3\log n} \leq \frac{1-H(p)}{3\log n} = I(p)/3\log n \end{aligned}$$

Putting it all together, the expected information gain is at least

$$\begin{aligned} H(1/2 + \epsilon_{\text{par}}(1-2p)) - H(p) &\geq 1 - 4\epsilon_{\text{par}}^2(1-2p)^2 - H(p) \\ &\geq I(p) - I(p)/3\log n = I(p)\left(1 - \frac{1}{3\log n}\right) \end{aligned}$$

which completes the proof. □

Note that ϵ_{par} is not a function of p. This is important if $p = o(1)$.

Lemma 2.5. *The expected number of comparisons before reaching the recursion condition in stage 2 is at most* $\log n/I(p) + O(1/I(p))$.

Proof. By Lemma 2.3, if $H(a_1, \ldots, a_n) < \log(1/\epsilon_{\text{par}})$ then we have reached the recursion condition. As the initial entropy is $\log n$ and the expected information gain per

comparison is $I(p)(1 - 1/3\log n)$ (by Lemma 2.4), the expected number of comparisons is at most

$$\begin{aligned}\frac{\log n - \log(1/\epsilon_{\text{par}})}{I(p)(1-1/3\log n)} &\leq \frac{\log n}{I(p)(1-1/3\log n)} \\ &\leq \frac{\log n}{I(p)} + \frac{2}{3I(p)}\end{aligned}$$

using the fact that $1/(c-x) < 1/c + 2x/c$ for $c > 2x \geq 0$. □

We now prove that with high probability, when the algorithm halts, s is in the correct surroundings. We begin by showing that the probability for a large majority of wrong answers in a small consecutive section is bounded. We then apply a union bound on all small consecutive sections, to get the result. Finally, we show that if the right element is not in the recursive surroundings then such an improbable section exists.

Lemma 2.6. *Let $r = \frac{12p(1-p)\log\log n}{2p-1}$, and assume $x_{i-1} \geq s \geq x_i$. Let $q_1, \ldots, q_t$ denote all the comparisons made until the algorithm stopped, sorted in descending order by the element which was compared to s (repetitions are possible). Let $A(k)$ denote the answer given by the oracle to q_k, that is $A(k) = 1$ with probability p if the k'th largest comparison compared s to an element which is larger than it. Let $q_j, \ldots q_t$ denote the comparisons to elements smaller than s. Then*

$$\Pr(\exists x : \exists y \geq j : \sum_{k=y}^{y+2x+r} A(k) > x + r) < 1/\log n$$

To continue, we need some bound on t, the number of queries. We know the expected number of comparisons until we halt. Let Q be the random variable measuring the query complexity of the algorithm, and let $\epsilon > 0$ is fixed. By Markov's inequality, we have that

$$\Pr\left(Q > (c+\epsilon)\frac{\log^2 n}{I(p)}\right) < \frac{1}{c\log n}$$

We can assume that $t < 2(\log n)^2/I(p)$, by paying a probability cost of $0.5/\log n$. We now use a union bound. Note that x is bounded by t. Fix x and y, and consider their contribution to the sum. This is bounded by the probability that $B(2x+r, 1-p) \geq x + r$, where B is the binomial distribution. Approximating by the normal distribution (which is applicable because $r = \Omega(\text{loglog } n)$ is not a constant), we get a standard deviation of $\sqrt{p(1-p)(2x+r)}$ and an expectancy of $(2x+r)(1-p)$, so the bound is roughly $2\exp\left(-\left(\frac{x+r-(2x+r)(1-p)}{2\sqrt{p(1-p)(2x+r)}}\right)^2\right)$. To find the worst case, we differentiate the exponent by x and find the minimum (without the minus sign). This yields $x = \frac{r(1-p)}{2p-1}$, and substituting gives the exponent as $\frac{r(2p-1)}{4p(1-p)}$. If the expression is less than $0.5/\log^3 n$, the total contribution from the $\log^2 n$ pairs is bounded by $0.5/\log n$, as

desired. Taking the logarithm of both sides, we get

$$\begin{aligned} \frac{r(2p-1)}{4p(1-p)} &= 3\text{loglog } n + O(1) \\ r &\approx \frac{12p(1-p)\text{loglog } n}{2p-1} \end{aligned}$$

Lemma 2.7. *Suppose $a_j \geq \epsilon_{par}$ in step 2. Let $r = \frac{12 \log\log(n) p(1-p)}{2p-1}$, and $l_{sur} = (\frac{p}{1-p})^r \frac{1}{\epsilon_{par}}$. Then with probability $\geq 1 - 1/\log n$ we have $x_{j-l_{sur}} \geq s \geq x_{j+l_{sur}}$.*

Proof. Assume otherwise, and let $x_{i-1} \geq s \geq x_i$. As the lemma is symmetric we can assume without loss of generality that $j > i + l_{sur}$. By the pigeonhole principle, there is some $k \in [i, i + l_{\text{sur}}]$ such that $a_k < 1/l_{\text{sur}}$. So $a_j/a_k \geq \frac{\epsilon_{\text{par}} p^r}{\epsilon_{\text{par}}(1-p)^r} = (\frac{p}{1-p})^r$. Since every update consists of multiplying some of the elements by p and some by $1-p$ (up to normalization), we can see that this implies that, for some x, there were $2x + r$ comparisons made which differentiated between a_j and a_k, of which $x + r$ pointed towards a_j (remember that the correct direction is towards a_k, which is the direction in which a_i, the correct answer, is found). We now use Lemma 2.6 to bound the probability of this event. □

In order to calculate the query complexity of the entire algorithm, we need an estimate for l_{sur}, as we recursively examine a neighborhood of size $2l_{\text{sur}} + 1$. Note that for $1/2 < p < 1$ and $a > 0$, we have

$$\left(\frac{p}{1-p}\right)^{ap(1-p)/(2p-1)} \leq e^{a/2}$$

So we get $l_{\text{sur}} < e^{6\text{loglog } n}/\epsilon_{\text{par}} = O(\log n^{6 \log e}/\epsilon_{\text{par}})$.

We can now bound the expected query complexity of the algorithm for $\delta < 1/\log n$. Denote the expected query complexity until the test in step 1 succeeds by T. Once the test succeeds, we pay a cost of $O(\log(1/\delta) + \log\log n)$ queries; then, with probability $1/\log n$ the algorithm will restart, and with probability $1 - 1/\log n$ it will continue recursively, operating on a polylogarithmic number of elements (of size $2l_{\text{sur}} + 1$). This means that the query complexity cost added by the case where the algorithm restarts is $O((T + \log\log n + \log(1/\delta))/\log n)$, which is negligible (following the same idea as in the proof of Theorem 1.1 from Theorem 2.1).

By Lemma 2.5 the expected runtime until $I(a_1, \ldots a_n) > \log n - \log(1/\epsilon_{\text{par}})$ is $\log n/I(p) + c/I(p)$ for some constant c. This concludes the proof of Theorem 2.1.

2.4 Lower Bounds

Theorem 2.8. (Lower bound) *Let A be a classical noisy binary search algorithm with success probability greater than $1 - \delta$, then its expected number of comparisons is at least*

$$(1-\delta)\frac{\log n}{I(p)} - 10/I(p)$$

To prove the lower bound, we first show a reduction from binary search to a channel coding problem, and then present a new information theoretic lower bound for this problem. We say that Alice and Bob communicate over a binary symmetric channel with feedback if

1. Alice has a binary symmetric channel towards Bob, with noise probability p for some $p < 1/2$
2. Bob has a perfect channel towards Alice, called the feedback channel. Communication in this channel is unlimited and free

Alice wishes to send Bob $\log n$ bits, with success probability $1 - \delta$. The players are allowed to use variable length coding, and we are only interested in the expected number of channel uses they require.

Lemma 2.9. *Let A be a noisy binary search algorithm, and success probability $1 - \delta$. Let Q denote the expected number of comparisons A requires. Then Alice can send Bob $\log n$ bits of information, over a binary channel with feedback, with success probability $1 - \delta$, such that the expected number of channel uses is Q.*

Proof. The players simulate A, over the channels they have. Denote by i the $\log n$ bits which Alice wishes to transmit to Bob. Alice considers the hypothetical case in which she has a sorted array, such that $x_i > s > x_{i+1}$, and Bob tries to find the place of s using comparisons. Alice and Bob now simulate A, where Bob tells that he wishes to compare s and x_j (by sending j in the feedback channel), and Alice responds with the corresponding answer to the comparison in the forward noisy channel. Thus, each comparison A makes is mapped to a single use of the forward channel, and the expected number of channel uses is Q. Bob decodes i correctly if and only if A's output was that $x_i > s > x_{i+1}$, and therefore the success probability doesn't change.

Finally, note that there is no real need for Bob to send the index j - it is enough that he sends Alice back the output of the forward channel, and she can compute j by herself. This is a general property for feedback channels. □

We can now transform lower bounds on variable length coding with feedback (when there is an error probability) to lower bounds on the expected runtime of A. Theorem 2.8 now follows from Theorem B.1 in Appendix B. □

Using this theorem, we can show that the probability that our algorithm halts prematurely (when we run it with small δ) is very low. As we know the expected runtime of our algorithm, a generalized Markov gives us some concentration on its query complexity.

2.5 Implementation Notes

We are interested in the query complexity of the algorithm, rather than its runtime. However, we note that a naive implementation is polylogarithmic in n (actually $O(\log n^2)$). This is done by uniting cells of the array $a_1, \ldots, a_n$ when there was no query which discriminates between them. We begin the algorithm with a single segment which consists of the entire array. Every query takes a segment, and splits it into two segments (so

in the end of the algorithm we are left with $O(\log n)$ segments). After each query the weight of each segment is updated ($O(\log n)$ time) and choosing where to ask the next query consists of going over the segments (again $O(\log n)$ time). This can be improved to $O(\log n \text{loglog } n)$ by saving the segments in a binary search tree. Every edge on the tree has an associated probability on it, such that multiplying the numbers on a path between the root to a certain vertex gives the weight of all the segments which are under the vertex (the leaves each represent a single segment). Suppose we need to query x_j after having already queried x_k and x_l, where $k < j < l$ and no other elements were queried between x_k and x_l. In this case the leaf which represents the segment $a_k, \ldots, a_l$ will have two sons, one representing $a_k, \ldots, a_j$ and the other representing $a_{j+1}, \ldots, a_l$. According to the result of the query, one son will have probability p, and the other $1 - p$. The data structure will then fix the probabilities on the path between the root and the vertex $a_k, \ldots, a_l$ according to the answer of the query. Both finding the right element and updating the probabilities takes time which is proportional to the depth of the tree. Each query increments the number of leaves, so there will be $O(\log n)$ leaves at termination. Keeping the search tree balanced (such as by using red-black trees) gives a tree depth of $O(\text{loglog } n)$ as required. It is also possible to implement an approximation of the search, in time $O(\log n)$, see [5, 4] for details.

3 Generalized Noisy Binary Search

In a binary search, the algorithm partitions a sorted array of items into two parts, and the oracle returns which part contains the desired element. Our generalization is to let the algorithm partition the sorted array according to k elements, and the oracle returns which interval between them contains the correct element.

Generalizing the noise model can be done in a few ways. One way is to assume that the algorithm actually makes k different comparisons in parallel, where each of them is noisy with probability p, and the probabilities for noise in different comparisons are independent. This model may be useful for biological applications. We use a different model, which is more suited to the quantum case. Instead of assuming that we get k bits (which is redundant in the noise-free case), we assume that we get one answer, which tells us which are the two elements (out of the k chosen elements) such that s is between these elements. There is now one correct answer, and k wrong ones, so we need to specify the probability for each kind of error. This is done by taking $k + 1$ probabilities (which add to 1), where the hth probability is the probability that the oracle returns $j + h \bmod (k + 1)$ instead of j, the correct reply.

Formally, let $g : \{1, \ldots, n-1\}^k \to \{0, \ldots, k\}$. If g is given k indices, $i_1 > i_2 > \ldots > i_k$, it outputs the answer j if $x_{i_j} \geq s \geq x_{i_{j+1}}$, where we take $i_0 = 1$ and $i_{k+1} = n$. The error probability is taken into account by associating $k + 1$ known numbers $p_0, \ldots, p_k$ to g, such that if $x_{i_j} \geq s \geq x_{i_{j+1}}$ then the result $j+h \mod (k+1)$ would appear with probability p_h.

The optimal algorithm for this case is very similar to the case $k = 1$ (which is f). In each step, partition the array into $k+1$ parts with (almost) equal probability, and ask which part contains the desired element. The only difference will be in the recursion condition. Instead of taking the surroundings of the most likely element, we pass to

the next stage all the elements with weight greater than ϵ_{pass}, which will be determined later. Let $a_1, \ldots, a_n$, ϵ_{par} be as before (albeit with different values this time).

1. If there is a value i such that $a_i > \epsilon_{\text{par}}$, halt. If the algorithm halts, take a set of all the elements with weight greater than ϵ_{pass} and run on it recursively. Note that it is possible to run recursively on a set of cells which are not a continues segment by ignoring all the cells which are not in the set. If s is not in this set, restart the algorithm.

2. Else, let $i_1, \ldots i_k$ be indices such that the sum of the elements between two indices does not deviate from $1/k$ by more than ϵ_{par}:

$$1/k - \epsilon_{\text{par}} \leq \sum_{h=i_{j-1}}^{i_j} a_h \leq 1/k + \epsilon_{\text{par}}$$

3. Apply $g(i_1, \ldots, i_k)$ and update the probabilities according to Bayes's rule, using the p_j's.

We use $\epsilon_{\text{par}} = 1/(6k+6)\log n$, and $\epsilon_{\text{pass}} = \frac{I(p)^2}{k(6k+6)\log^6 n}$.
Remember $I(p_0, \ldots, p_k) = \log(k+1) - H(p_0, \ldots, p_k)$.

Theorem 3.1. *The algorithm presented finds the right element with probability* $1-\delta$ *in an expected query complexity of*

$$\frac{\log n}{I(p_0, \ldots, p_k)} + O(\frac{\textit{polyloglog}\, n \log(1/\delta)}{I(p_0, \ldots, p_k)})$$

The proof of this theorem greatly resembles the one of Theorem 1.1. In particular, it is based on analyzing the expected entropy of the distribution $a_1, \ldots a_n$, and a similar technique for $\delta > \text{loglog}\, n/\log n$ gives the factor of $1-\delta$. There are two main differences:

1. We show an analog of Lemma 2.4, to show that the entropy of $a_1, \ldots a_n$ decreases fast enough

2. We show an analog of Lemmas 2.6,2.7, proving that with high probability, when we apply the recursion condition we pass the element to the next stage.

3.1 The Entropy Decrease for k Segments

Before we prove the main lemma, we formally present the update procedure. Let $a_1, \ldots, a_n$ be the probabilities before the comparison. Let $a_{i_0}, \ldots a_{i_k}$ be the $k+1$ elements involved in the comparison. Denote $B_r = \{i : i_r \leq i < i_{r+1}\}$ denote the r'th segment, and $W_r = \sum_{i \in B_r} a_i$.

Remember that p_{j+h} is the probability to get the answer h if the correct answer is j, i.e. s is in the j'th segment. Equivalently, p_{j-r} is the probability for getting the result j if $a_{i_r} \leq s < a_{i_{r+1}}$. In this case

$$\Pr(\text{output } j) = \sum_r \Pr(\text{output } j | a_{i_r} \leq s < a_{i_{r+1}}) \Pr(a_{i_r} \leq s a_{i_{r+1}}) = \sum_r p_{j-r} W_j$$

Let b_i be the updated value of a_i. Then the Bayesian update for $a_i \in B_r$ if the result was j is

$$b_i = a_i p_{j-r} / N_j$$

and

$$N_j = \sum_r W_r p_{j-r}$$

is the normalization factor for this result. Note that $N_j = 1/\Pr(\text{output } j)$.

Lemma 3.2. *Assume $\forall i : a_i < \epsilon_{par}$. Then:*

$$E[I(b_1, \ldots, b_n)] - I(a_1, \ldots, a_n) \geq I(\overline{p})(1 - (2k+2)\epsilon_{par})$$

Proof. Assume that we partitioned the cells into blocks $B_0, \ldots, B_k$, with total weights $W_0, \ldots, W_k$ respectively. We can assume that $\forall i : |W_i - 1/k| < \epsilon_{\text{par}}$. Therefore

$$N_i = \sum_j W_j p_{j-i} \geq \sum_j \left(\frac{1}{k+1} - \epsilon \right) p_{j-i} = \left(\frac{1}{k+1} - \epsilon \right) \sum_j p_{j-i} = \frac{1}{k+1} - \epsilon$$

and similarly we get $N_i \leq \frac{1}{k+1} + \epsilon$.

We now bound $E(H(b_1, \ldots, b_n))$, where the expectancy is over the result j.

$$E(H(b_1, \ldots, b_n)) = \sum_j H(b_1, \ldots, b_n | \text{output } j) \Pr(\text{output } j)$$

Considering the event that the result was j

$$\begin{aligned} H(b_1, \ldots, b_n | \text{output } j) &= \sum_i \sum_{t \in B_i} p_{j-i} a_t / N_j \log(p_{j-i} a_t / N_j) \\ &= \frac{1}{N_j} \sum_i \sum_{t \in B_i} p_{j-i} a_t \log(p_{j-i} a_t / N_j) \end{aligned}$$

Fortunately, $1/N_j = \Pr(\text{output } j)$, so we have

$$\begin{aligned} E(H(b_1, \ldots, b_n)) &= \sum_j \sum_i \sum_{t \in B_i} p_{j-i} a_t \log(p_{j-i} a_t / N_j) \\ &= \sum_j \sum_i \sum_{t \in B_i} p_{j-i} a_t \left[\log p_{j-i} + \log a_t - \log N_j \right] \end{aligned}$$

We now look at each of the three terms separately.

$$\begin{aligned}\sum_j \sum_i \sum_{t\in B_i} p_{j-i} a_t \log p_{j-i} &= \sum_j \sum_i p_{j-i} \log p_{j-i} \sum_{t\in B_i} a_t = \sum_j \sum_i p_{j-i} \log p_{j-i} W_i \\ &= \sum_i W_i \sum_j p_{j-i} \log p_{j-i} = H(p)\end{aligned}$$

As for the second term

$$\sum_j \sum_i \sum_{t\in B_i} p_{j-i} a_t \log a_t = \sum_i \sum_{t\in B_i} a_t \log a_t \sum j p_{j-i} = \sum_i \sum_{t\in B_i} a_t \log a_t = H(a_1, \ldots, a_n)$$

And the third

$$-\sum_j \sum_i \sum_{t\in B_i} p_{j-i} a_t \log N_j = -\sum_j \log N_j \sum_i p_{j-i} W_i = -\sum_j N_j \log N_j$$

Thus the expected decrease in entropy is

$$-\sum_j N_j \log N_j - H(p)$$

Using the fact that $\sum_j N_j = 1$, Taylor's approximation gives that if $|N_j - 1/k| < \frac{\epsilon}{2k+2}$ then

$$\sum_j N_j \log N_j > \log(k+1) - \epsilon$$

Specifically, choosing $\epsilon_{\text{par}} = 1/((6k+6)\log n)$ then the expected information gain at each step is at least $I(p)(1 - 1/(3\log n))$ and applying Lemma 2.5 (*mutatis mutandis*) shows that we will reach the recursion condition within $\log n / I(p) + O(1/I(p))$ steps. □

3.2 Correctness Proof for the Halt Condition

Lemma 3.3. *When we reach the recursion condition, with probability greater than* $1 - 1/\log n$ *the correct cell has high weight, specifically more than*

$$\epsilon_{par} I(p)^2 / k \log^5 n \geq \frac{I(p)^2}{k(6k+6)\log^6 n}$$

As we pass all the cells with higher weight to the next stage, and as the failure probability is too small to effect the main term in the runtime, this lemma finishes the correctness proof.

Proof. Note first that with probability at least $1 - 1/2\log n$ we reach the recursion condition within $t = E\log n = \log^2 n/I(p)$ steps, from a trivial Markov bound (as the number of steps is always nonnegative).

We now want to prove that with high probability, no cell is a lot larger than the right one. The probability that after s queries, a specific (wrong) element has weight larger than c times the weight of the right one is at most $1/c$. Note that for this to hold we only need that the distribution $p_0, \ldots, p_k$ is not uniform[2]. However, there are $n-1$ wrong cells, so applying a union bound is not possible. Fortunately, if two cells were never separated by a question, they have the same probability (as they passed the same update process - see Subsection 2.5).

After s queries, there are only $ks + 1$ unique weights, as each query divides k segments, each into two subsegments. Using the Markovian bound, we only need to apply a union bound on $(\log n)^2/I(p)$ stages of the algorithm. In each stage, there are at most $k(\log n)^2/I(p)$ different cells. Setting the ratio $c = 2k\log^5 n/I(p)^2$, the probability that a specific cell (or segment) to be c times more heavy than the right one at any stage of the algorithm is at most $1/c$. Taking a union bound on the different segments in each stage of the runtime gives a failure probability of at most

$$\frac{(\log n)^2}{I(p)} \cdot \frac{k(\log n)^2}{I(p)} \cdot \frac{1}{c} = 1/2\log n$$

Adding the probability of $1/2\log n$ to fail the Markovian argument on the number of stages gives the result. □

Note that this proof also lets us deduce the result for $k = 2$, which appeared separately as Lemmas 2.6 and 2.7. However, $1/l_s ur$ is larger than ϵ_{pass}, since we can use a more exact approximation of the update process and also have a better bound for ϵ_{par}.

4 Quantum Search With a Non-Faulty Oracle

Farhi et al. presented in [12] a "greedy" algorithm which, given t queries and an array of size K, attempts to find the correct element but has some error probability. In fact, their algorithm actually does more. Assume that the elements given to their algorithm are $y_0, .., y_{K-1}$ and the special element s. Again we are trying to find i which satisfies $y_i \geq s \geq y_{i+1}$. Their algorithms outputs a quantum register with the superposition $\Sigma_{j=0}^{K-1}\beta_j|(j+i)\rangle$ (with all indices taken mod K) for fixed $\beta_0, \ldots, \beta_{K-1}$ which are not a function of s. Let $p_j = |\beta_j|^2$. When measuring the register we obtain the correct value with probability p_0. The exact numbers $p_0, \ldots p_{K-1}$ are determined by the number of oracle queries t. We now use their algorithm (with proper values for K and t) as a subroutine in our generalized search algorithm with $k = K$.

[2]One could get a better approximation by looking at the exact values. At first glance, it may seem surprising that it is possible to say something which is independent of the distribution. Note however that if the distribution is very close to uniform, then because of the update procedure it would have to favor the wrong element over the right one many times to get the factor of c. On the other hand, if it is far from uniform, the probability each time to get a "wrong" answer is small. These two effects cancel out, and we get the trivial bound.

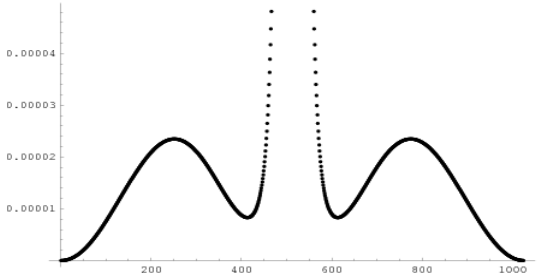

Figure 1: The probability for measuring each element out of 1024 elements after 3 queries. The probability to find the right element is 0.598. The entropy of this distribution is 2.817 bits, the information gain is 7.182, and it yields a quantum search algorithm with complexity $0.417 \log n$

We present a table which describes the algorithm for $K = 2^{26}$. The second column gives the probability of finding the right element after t queries, while third column gives the information of the distribution. The last column gives the resulting query complexity of searching a sorted list of n elements using t queries on 2^{26} elements as a subroutine, and is calculated by dividing the information gain by t.

Number of Queries t	Success Probability	Information Gain $I(t)$	Query Complexity
1	$2.3 \cdot 10^{-6}$	1.45	0.687 $\log n$
2	0.000088	3.77	0.53 $\log n$
3	0.0014	7.5	0.400 $\log n$
4	0.0134	11.2	0.357 $\log n$
5	0.0727	15.01	0.333 $\log n$
6	0.2513	18.802	0.319 $\log n$
7	0.57	22.3138	0.3137 $\log n$
8	0.877	24.921	0.321 $\log n$

For each fixed size K, increasing t above some threshold does not help the algorithm. Figure 2 describes the expected runtime, as a function of the logarithm of the size of the search space, for 5 to 8 queries. The figure gives evidence towards the statement that increasing K always improves the query complexity (for the optimal choice of t). This raises the question of whether an exact analysis of the greedy algorithm gives the optimal quantum algorithm.

Using $K = 2^{26}$ and $t = 7$ gives a distribution Q with $I(p_0, \ldots, p_k) = 22.3138$. This gives us an algorithm which requires less than $0.314 \log n$ oracle questions with $o(1)$ failure probability, proving Theorem 1.2

For every size of K we checked, the success probability for the optimal t was quite low (about 0.6). This means that the measurement distribution is important, and not just the probability of finding the right element. Figure 1 shows this distribution for 1024 elements and 3 queries. Figure 3 shows this distribution (on a log plot) for 1 to 6 queries. The number of side lobes is proportional to the number of questions.

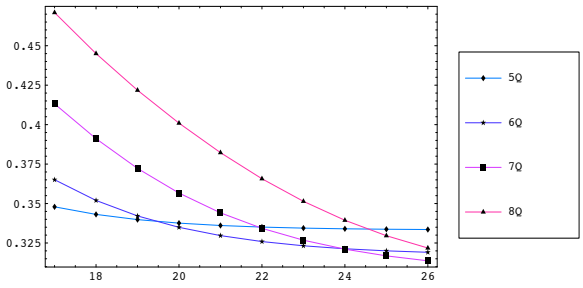

Figure 2: The expected runtime of binary search, in a log-linear plot, when the quantum subroutine uses 5,6,7 or 8 queries

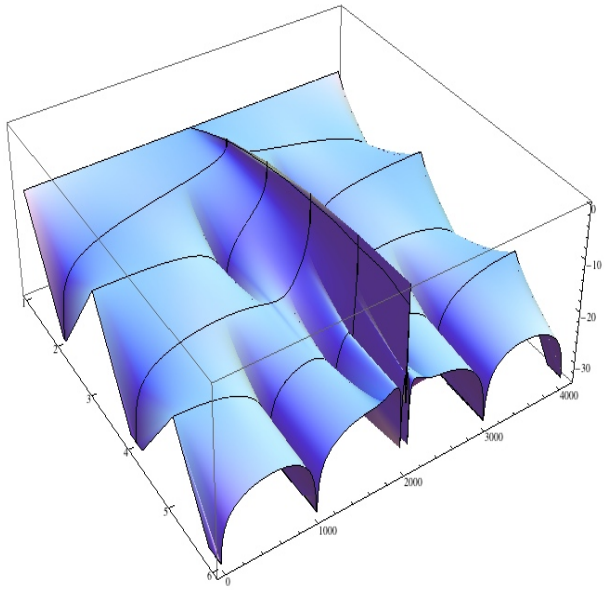

Figure 3: The probability of getting each element out of 4096, assuming that the correct element is 2048, for 1 to 6 queries. The probability is depicted in a logarithmic plot

5 Quantum Lower Bounds

5.1 Noisy Quantum Search

Let O be a quantum search oracle, $O(|x,c\rangle) = |x,(0 \oplus c)\rangle$ if $x \geq s$ and $|x,(1 \oplus c)\rangle$ if $x < s$. We want to define a noisy version of the oracle, which will generalize the classical noisy oracle, that is, we want to it to have a probability for the correct answer, as well as a probability for the wrong one[3]. We thus define $O(|x,c\rangle) = \sqrt{p}|x,(c \oplus f(x))\rangle + \sqrt{1-p}|x,(c \oplus f(x) \oplus 1)\rangle$ where $f(x) = 0$ if and only if $x > s$ (see [16, 6, 15]). Clearly the complexity of the optimal algorithm, as a function of p, cannot be worse than in the classical case. We show that, up to a constant factor, the dependence is identical in the quantum and classical cases.

In [16], Lemma 5, it is stated that

$$|\langle \psi_x^j | \psi_y^j \rangle - \langle \psi_x^{j+1} | \psi_y^{j+1} \rangle| \leq 2 \sum_{i, x_i \neq y_i} ||P_i|\psi_x^j\rangle|| \cdot ||P_i|\psi_x^j\rangle||$$

using the Cauchy-Schwarz inequality for the proof. In the case of a noisy oracle, an identical proof shows that

$$\begin{aligned} &|\langle \psi_x^j | \psi_y^j \rangle - \langle \psi_x^{j+1} | \psi_y^{j+1} \rangle| \\ &\quad \leq 2\sqrt{p(1-p)} \sum_{i, x_i \neq y_i} ||P_i|\psi_x^j\rangle|| \cdot ||P_i|\psi_x^j\rangle|| \end{aligned}$$

Using this tighter bound in the rest of [16] we get:

Theorem 5.1. *Any noisy quantum algorithm requires at least* $\frac{\ln(n)}{\pi\sqrt{p(1-p)}} \approx 0.202 \frac{\log n}{\sqrt{p(1-p)}}$ *queries.*

5.2 Lower Bounds for Quantum Search with a (high) Probability of Error

Our techniques enable us to give a better lower bound for the number of queries that a quantum noiseless algorithm needs to the find the right element in a sorted list with probability at least $1 - \delta$.

Theorem 5.2. *Any quantum algorithm which finds the right element in an array of length* k *with success probability greater than* $1 - \delta$ *requires at least* $t \geq \frac{\ln(2)}{\pi}((1 - \delta)\log(k)) - O(1)$ *queries.*

Proof. Given a quantum algorithm on an array of size k with success probability $1 - \delta$, we can use it as a basis for the recursive step for the algorithm in Section 3, by taking probabilities

$$p_i = \begin{cases} 1 - \delta & i = 0 \\ \frac{\delta}{k-1} & i \neq 0 \end{cases}$$

[3] It is possible to define noisy quantum oracles in several other ways. For example, one can define oracles which sometimes do not act on the state at all [24], or oracles which present us with a state which is close (in some norm) to the desired state.

Here

$$I(p_0, \ldots, p_k) = \log(k) + (1-\delta)\log(1-\delta) + \delta\log(\delta/(k-1))$$

and we gain $I(p_0, \ldots, p_k)/t$ bits of information per query. However, we know from [16] that any perfect quantum search algorithm for an ordered list needs at least $\frac{\ln n}{\pi}$ queries. This means that the average information gain per query is at most $\pi/\ln(2)$ bits per query, so

$$\frac{1}{t}(\log(k) + (1-\delta)\log(1-\delta) + \delta\log(\delta/(k-1))) \leq \frac{\pi}{\ln(2)}$$

Manipulating this gives the result. □

This bound is nontrivial as long as $\delta < 1 - \frac{1}{k}$, which is much better than the previously best lower bound of

$$t \geq (1 - 2\sqrt{\delta(1-\delta)})\frac{1}{\pi}(H_k - 1)$$

by [16] which is trivial for $\delta > 1/2$.

6 Open Problems

An interesting classical open problem is to study the classical generalization of noisy binary search with independent answers. Giving upper and lower bounds is important, especially as a function of k.

We believe that tight asymptotic analysis of the greedy algorithm can lead to algorithms that are better than the one presented here. Also, trying to decrease the entropy at each stage (and not just maximize the probability to get the correct answer if we measure immediately), could help decrease the complexity.

7 Acknowledgments

We thank Dorit Aharonov for many stimulating discussions. A. H. thanks Haran Pilpel and Oded Regev for their help and comments.

References

[1] Ambainis. A better lower bound for quantum algorithms searching an ordered list. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.

[2] Aslam and Dhagat. Searching in the presence of linearly bounded errors (extended abstract). In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1991.

[3] Javed A. Aslam. *Noise tolerant algorithms for learning and searching*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1995.

[4] Bent, Sleator, and Tarjan. Biased search trees. *SICOMP: SIAM Journal on Computing*, 14, 1985.

[5] Borgstrom and Kosaraju. Comparison-based search in the presence of errors (preliminary version). In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1993.

[6] Harry Buhrman, Ilan Newman, Hein Röhrig, and Ronald de Wolf. Robust quantum algorithms and polynomials. *CoRR*, quant-ph/0309220, 2003. informal publication.

[7] Andrew M. Childs, Andrew J. Landahl, and Pablo A. Parrilo. Improved quantum algorithms for the ordered search problem via semidefinite programming, August 21 2006. Comment: 8 pages, 4 figures.

[8] Andrew M. Childs and Troy Lee. Optimal quantum adversary lower bounds for ordered search, August 24 2007. Comment: 13 pages, 2 figures.

[9] Cicalese, Mundici, and Vaccaro. Least adaptive optimal search with unreliable tests. In *SWAT: Scandinavian Workshop on Algorithm Theory*, 2000.

[10] T.M. Cover, J.A. Thomas, J. Wiley, and W. InterScience. *Elements of Information Theory*. Wiley-Interscience New York, 2006.

[11] Aditi Dhagat, Péter Gács, and Peter Winkler. On playing "twenty questions" with a liar. In *SODA*, pages 16–22, 1992.

[12] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Invariant quantum algorithms for insertion into an ordered list, January 19 1999. Comment: 19 pages, LaTeX, amssymb,amsmath packages; email to farhi@mit.edu.

[13] Feige, Raghavan, Peleg, and Upfal. Computing with noisy information. *SICOMP: SIAM Journal on Computing*, 23, 1994.

[14] J. Heinemeyer, H. Eubel, D. Wehmhönre, L. Jänsch, and H. Braun. Protemic approach to characterize the supramulecular organization of photosystems in higher plants. *Phytochemistry*, 65:1683–1692, 2004.

[15] Hoyer, Mosca, and de Wolf. Quantum search on bounded-error inputs. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 2003.

[16] Hoyer, Neerbek, and Shi. Quantum complexities of ordered searching, sorting and element distinctness. *ALGRTHMICA: Algorithmica*, 34, 2002.

[17] Jacokes, Landahl, and Brooks. An improved quantum algorithm for searching an ordered list. *In preparation*, 2006.

[18] Karp and Kleinberg. Noisy binary search and its applications. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 2007.

[19] S. Muthukrishnan. On optimal strategies for searching in presence of errors. In *SODA*, pages 680–689, 1994.

[20] Genevieve B. Orr. Removing noise in on-line search using adaptive batch sizes. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 232. The MIT Press, 1997.

[21] A. Pedrotti. Searching with a constant rate of malicious lies. In Elena Lodi, Linda Pagli, and Nicola Santoro, editors, *Proceedings of the International Conference on Fun with Algorithms (FUN-98)*, pages 137–147, Waterloo, Ontario, June 18–20 1999. Carleton Scientific.

[22] Pelc. Searching with known error probability. *TCS: Theoretical Computer Science*, 63, 1989.

[23] Pelc. Searching games with errors–fifty years of coping with liars. *TCS: Theoretical Computer Science*, 270, 2002.

[24] Oded Regev and Liron Schiff. Impossibility of a quantum speed-up with a faulty oracle. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 773–781. Springer, 2008.

[25] Alfred Rényi. On a problem in information theory, 1961.

[26] R. L. Rivest, A. R. Meyer, D. J. Kleitman, and K. Winklmann. Coping with errors in binary search procedures. *J. Comput. Sys. Sci.*, 20:396–405, 1980.

[27] H. Schägger, W. Cramer, and G. von Jagow. Analysis of molecular masses and oligomeric states of protein complexes by blue native electrophoresis and isolation of membrane protein complexes by two-dimensional native electrophoresis. *Analytical Biochemiostry*, 217:220–230.

[28] S. M. Ulam. *Adventures of a Mathematician*. Scribner's, New York, 1976.

A A Review of the Greedy Algorithm

In this appendix we give a short presentation of the quantum algorithm of Farhi et al. (in [12]), which is being thoroughly used in our paper. As a part of this description we present the reduction to translationally invariant algorithms.

Farhi et al. solve a different problem which is equivalent to search. They define n functions $f_j(x)$ by

$$f_j(x) = \begin{cases} -1, x < j \\ 1, x \geq j \end{cases}$$

for $j \in \{1, \ldots, n\}$. A query in this problem is giving the oracle a value x, and getting $f_j(x)$ for some fixed but unknown j. The goal of the algorithm is to find j. They then double the domain of the functions and define $F_j(x)$ by

$$F_j(x) = \begin{cases} f_j(x), 0 \leq x \leq n-1 \\ -f_j(x-n), n \leq x \leq 2n-1 \end{cases}$$

and use the fact that $F_{j+1}(x) = F_j(x-1)$ to analyze their algorithm only for $j = 0$. They also define $G_j|x\rangle = F_j(x)|x\rangle$ and $T|x\rangle = |x+1\rangle$. This means that their algorithm can be described as

$$V_k G_j V_{k-1} \ldots V_1 G_j V_0 |0\rangle$$

followed by a projective measurement which decides the result. Noticing that $T^j G_j T^{-j} = G_0$, Farhi et al. found a base which they denote $|0+\rangle, \ldots, |n-1+\rangle, |0-\rangle, \ldots |n-1-\rangle$ such that $T^j|0\pm\rangle = |j\pm\rangle$, and when the measurement results in $j\pm$, the algorithm outputs that it got the jth oracle as an input[4].

Demanding that $V_l = TV_{l-1}T^{-1}$, it is possible to calculate the success probability of any given algorithm, by looking at the inner product $\langle V_k G_0 V_{k-1} \ldots V_1 G_0 V_0 |0\rangle |0\pm\rangle$. For any given state $|\psi\rangle$, it is possible to calculate which V will maximize $\langle V G_0 \psi | 0\pm\rangle$. Farhi et al. define the greedy algorithm recursively starting from V_0, such that each V_l is chosen to maximize the overlap of $|V_{l-1}G_0, \ldots V_1 G_0 V_0\rangle$ with $|0\pm\rangle$. Analyzing analytically the behavior of the greedy algorithm is an interesting open problem. Its behavior for finite (albeit large) size is the basis for our algorithm. It is interesting to note that the full distribution is important and not just the probability for a correct answer (which is what the algorithm maximizes).

B Information Theoretical Lower Bound

In this section we prove a lower bound for sending information over a noisy channel, when the algorithm is allowed large failure probability, which can approach one as the amount of information we require to send grows. This type of results is not common, as we are usually interested in sending information such that the failure probability tends to zero as the amount of information grows. We present the lower bounds in a very strong model, where the sender and the receiver have a noiseless feedback channel, and are only interested in the expected number of channel uses (they are allowed variable length coding). This model serves as a lower bound for the noisy binary search problem.

We introduce some notation. Alice wishes to send Bob $\log n$ bits of information, over a binary symmetric channel with noise probability p. Bob has a perfect communication channel towards Alice, and communication in this channel is free. Equivalently, we can assume that Alice knows what was the bit received by Bob every time she sends something. Let **C** be a communication protocol for sending $\log n$ bits, with success probability at least $1-\delta$. The rest of the section is devoted to proving the following theorem:

[4]Actually the result should be $|j+\rangle$ if k is even and $|j-\rangle$ if k is odd. We ignore this point as it is not necessary for the understanding of the algorithm.

Theorem B.1. *Let k be the expected number of times* $\mathbf{C}$ *uses the noisy communication channel, and $\alpha = 10$. Then*

$$k > (1-\delta)\frac{\log n}{1-H(p)} - \alpha$$

For n large enough.

This bound is meaningful even when $\delta = 1 - O\left(1/\log n\right)$, and the success probability tends to zero as n grows.

Let C be a protocol with success probability $1-\delta$, which uses the channel k times. We can build another protocol $\mathbf{C}'$ out of it by halting every run of C after $k\log^2 n$ steps. If we halt the run, $\mathbf{C}'$ just outputs 1. Let E_x denote the expected number of channel uses $\mathbf{C}'$ requires, and $1-\tau$ its failure probability. The following properties hold

1. $\mathbf{C}'$ expected runtime lower bounds that of C, or $k \geq E_x$
2. The failure probability of $\mathbf{C}'$ is not much greater than that of C, $1-\tau > 1-\delta-1/\log^2 n > 1-\delta-\alpha/2\log n$

Properties 1,2 imply that to prove B.1 it is enough to show that

$$E_x > (1-\tau)\frac{\log n}{1-H(p)} - \alpha/2 \tag{2}$$

It is easy to see that the variance of the number of channel uses $\mathbf{C}'$ requires is bounded by $kE_x\log^2 n < 2E_x^2\log^2 n$.

We now show how by iterating $\mathbf{C}'$ one can create a new code, which would pass $n\log n$ bits with very high success probability, using the channel at most $\frac{n}{1-\tau} + n^{0.52})(E_x+4)$ times. The result will then follow from standard bounds on fixed-length codes with high success probability. We consider the $n\log n$ input bits as n symbols, each of size $\log n$ bits, and let T denote the series of the symbols, $|T| = n$.

1. Initialize $T_0 = T$
2. While $|T_i|$ is greater than zero,
   (a) *Data Phase:* Alice sends Bob each of the symbols in T_i, using $\mathbf{C}'$. Note that in any attempt Alice knows exactly what symbol Bob decoded.
   (b) *Control Phase:* Let S_i be a vector of length $|T_i|$, such that $S[j] = 1$ if the j'th attempt failed. Alice encodes S_i using a good error correcting code using $(4|T_i|/(1-H(p)) + 4\sqrt{n}/(1-H(p)))$ bits. This means that the probability that Bob decodes S_i incorrectly is at most $1/n$.
   (c) *Internal update:* Let T_{i+1} be a vector of all the symbols (of size $\log n$) which Bob didn't receive correctly (corresponds to the places where $S_i[j] = 1$).

Protocol **P** passes $n\log n$ bits

If at any time Bob fails to understand Alice's message in step (2b), we say that $\mathbf{P}$ failed, and Alice and Bob halt. In order to bound the probability for this event, we first prove a few properties of $\mathbf{P}$.

Lemma B.2. *With probability greater than $1 - 1/n$, the number of iterations of $\mathbf{P}$ is at most $\frac{(\log n)^2}{1-\tau}$.*

Proof. In the first iteration, Alice tries to send n symbols, such that the success probability of each transmission is $1 - \tau$. The probability that a certain symbol isn't transmitted successfully after $\frac{(\log n)^2}{1-\tau}$ rounds is at most $1/n^2$, as this is just a geometric variable. a union bound gives the result. □

Lemma B.2 bounds the number of cycles Alice and Bob need. We now bound the number of times $\mathbf{C}'$ was used

Lemma B.3. *With probability greater than $1 - 1/\log^2 n$, $\mathbf{P}$ uses $\mathbf{C}'$ at most $\frac{n}{1-\tau} + \frac{\tau\sqrt{n}\log n}{(1-\tau)^2}$ times.*

Proof. The proof is derived by computing the sum of n independent geometric random variables, each with parameter $1 - \tau$. The variance of such a variable is

$$\frac{\tau}{(1-\tau)^2}$$

According to Tchebychev's Inequality, the probability that this sum exceeds $\frac{n}{1-\tau} + \frac{\tau\sqrt{n}\log^2 n}{(1-\tau)^2}$ is at most $1/\log^2 n$, as required. □

We now have the with probability at least $1 - 2/\log^2 n$, the number of rounds as well as the number of uses of $\mathbf{C}'$ is bounded. We bound from above the number of channel uses, assuming this event.

Lemma B.4. *With probability at least $1 - 4/\log^2 n$, Alice and Bob used the channel at most $(\frac{n}{1-\tau} + n^{0.52})(E_x + 4)$ times.*

Proof. We bound the number of channel uses by bounding two independent terms:

1. The number of times $\mathbf{C}'$ is used, times the number of bits it requires each time
2. The number of channel uses required to pass the control data (the vectors S_i)

According to Lemma B.3, with probability at least $1 - 1/\log^2 n$, $\mathbf{C}'$ was used at most $\beta = \frac{n}{1-\tau} + 2\frac{\tau\sqrt{n}\log^2 n}{(1-\tau)^2}$ times. The number of channel uses each time is a random variable, with expectancy E_x and variance at most $2E_x^2 \log^2 n$. Conditioned on this event, Tchebychev's Inequality gives that with probability $\geq 1 - 1/\log^2 n$ the total number of channel uses was

$$\beta E_x + \sqrt{\beta} 2E_x^2 \log^4 n \leq (\frac{n}{1-\tau} + n^{0.51})E_x$$

Where the $\log^4 n$ comes from the variance of $\mathbf{C}'$ times the slack required for Tchebychev's inequality, and the constant 0.51 is somewhat arbitrary.

We now bound the number of channel uses required for the control phase of $\mathbf{P}$, which consists of $(4|T_i|/(1-H(p)) + 4\sqrt{|T_0|}/(1-H(p)))$ bits in the i'th iteration. To bound the $\sum_i |T_i|$, note that any application of $\mathbf{C}'$ appears in one such T_i (whether it was successful or not). With probability $1 - 1/\log^2 n$ the number of times $\mathbf{C}'$ is applied is bounded by β, and thus this requires 4β bits to bound the first term. As for the second term, according to Lemma B.2, with probability at least $1-1/n$ the number of rounds is at most $\frac{(\log n)^2}{1-\tau}$. Multiplying this by $4\sqrt{|n|}/(1-H(p))$ and adding the first term gives that with probability at least $1-4/\log^2 n$ the second phase of the algorithm incurs at most

$$4\beta + \frac{4\sqrt{|n|}\log^2 n}{(1-H(p))(1-\tau)}$$

channel uses. The lemma stems from from

$$(\frac{n}{1-\tau}+n^{0.51})E_x+4\frac{n}{1-\tau}+8\frac{\tau\sqrt{n}\log n}{(1-\tau)^2}+\frac{4\sqrt{n}\log^2 n}{(1-H(p))(1-\tau)} \leq (\frac{n}{1-\tau}+n^{0.52})(E_x+4)$$

□

To apply known coding inequalities, we require a protocol which has a constant block size. We therefore define a protocol $\mathbf{P}'$ which passes $n\log n$ bits, by applying protocol $\mathbf{P}$, and halting if the number of channel uses exceeds $(\frac{n}{1-\tau} + n^{0.52})(E_x + 4)$ The error probability of $\mathbf{P}'$ can be bounded as follows

Lemma B.5. *The failure probability of* $\mathbf{P}'$ *is at most* $5/\log^2 n$

Proof. According to Lemma B.2With probability at least $1 - 1/n$, there are at most $\frac{(\log n)^2}{1-\tau}$ communication cycles. In each cycle the failure probability is at most $1/n$. Taking a union bound gives that the probability that Bob will not know if a bit was passes correctly in any round is

$$\frac{(\log n)^2}{n(1-\tau)} \leq 1/\log^2 n$$

With probability at least $1 - 4/\log^2 n$ the number of channel uses is not too large. Applying a union bound on theses sources of failure finishes the proof. □

Finally, we present bounds for codes with feedback channel, but with small failure probability P_e. The following bound is taken from [10] (see chapter 8 Equation 139), and is derived from Fano's inequality:

$$mR \leq P_e mR + 1 + m(1-H(p))$$

where the channel is used m times, to pass mR bits. Note that this inequality is exact, and not just asymptotic. Manipulating this equation gives

$$m \geq \frac{n\log n - P_e n\log n - 1}{1-H(p)} > \frac{n\log n}{1-H(p)} - \frac{n}{(5-H(p))\log n}$$

Where the last inequality comes from substituting P_e. Finally, we substitute the value $m = (\frac{n}{1-\tau} + n^{0.52})(E_x + 4)$, pass sides and divide by n to get

$$E_x + 4 > (1-\tau)\left(\frac{\log n}{1 - H(p)} - \frac{5}{(1 - H(p))\log n} - n^{-0.48}\right) > (1-\tau)\frac{\log n}{1 - H(p)} - 1$$

or equivalently, $E_x > (1-\tau)\frac{\log n}{1-H(p)} - 5$. This gives Equation 2, and finishes the proof of Theorem B.1