An Improved Quantum Algorithm for Searching an Ordered List

Brian Jacokes

under the direction of
Dr. Andrew Landahl
Massachusetts Institute of Technology

Research Science Institute
July 31, 2007

Abstract

The best known quantum algorithm for searching an ordered list of N items is recursive and requires $3\log_{52} N \approx 0.526\log_2 N$ queries. We find an improved base case for this recursion, yielding an algorithm that requires only $4\log_{434} N \approx 0.457\log_2 N$ queries. This algorithm is more than twice as fast as the best possible classical algorithm, which requires $\log_2 N$ queries.

1 Introduction

A common computing problem is to find a target element in a list of size N, returning the index of that element as the output. The goal of algorithms for this problem is to find the target element as quickly as possible. One measurement of speed for such algorithms is how many queries the algorithm makes, where a query is an operation which gathers information on elements in the list.

The class of searching problems is divided into the problems of searching unordered and ordered lists. In both problems, querying an element of the list results in obtaining a one-bit answer. However, in searching unordered lists, a query only returns whether the element queried is the target element or not, while in searching ordered lists, a query returns whether or not the element queried is after the target element in the list. Thus, extra information is returned by queries to an ordered list, resulting in faster algorithms than those for unordered lists.

Although searching an unordered list can be performed in a provably optimal $O(N)$ queries on classical computers, it was shown in [1] that quantum computers can perform the same computation in $O(\sqrt{N})$ queries, which was shown in [2] to be the quantum lower bound for the problem. Searching an ordered list can be performed classically in $\log_2 N$ queries, which is also provably optimal. Quantum algorithms have been found which solve the same problem in both $\log_3 N \approx 0.631 \log_2 N$ queries [3] and $3 \log_{52} N \approx 0.526 \log_2 N$ queries [4]. A quantum lower bound for searching an ordered list was shown in [3] to be $\frac{1}{\pi}(\ln N - 1) \approx 0.221 \log_2 N$ queries, suggesting that a better quantum algorithm could exist for the problem.

In this paper, we build on methods presented in [4], which were used to obtain the previous upper bound of $0.526 \log_2 N$ queries for searching an ordered list. These methods include a series of equivalences, the final of which is to find a function of multiple variables

which obeys certain properties. We define a cost function whose minimization will result in finding such a function, and then present methods for minimizing this cost function. Using our methods, we find a new algorithm which solves the problem in $4 \log_{434} N \approx 0.457 \log_2 N$ queries, providing an improvement over the previous upper bound.

2 Background

Although quantum computing is still in its infancy, quantum algorithms have provided speedups over classical algorithms for many problems. This speedup is due in part to the ability of quantum systems to be in superpositions of multiple states, a phenomenon known as working in *quantum parallel* [5].

2.1 Quantum computation

Classical computers encode information using bits, which may take on values of either 0 or 1. Thus, any system of n classical bits represents exactly one state: a string of 0s and 1s chosen from among 2^n possibilities. However, if we were instead using quantum bits, or *qubits*, then the state of this n-qubit system, denoted $|\Psi\rangle$, would be given by a superposition of the 2^n classical states, each having a complex coefficient a_i:

$$|\Psi\rangle = \sum_{i=0}^{2^n-1} a_i |i\rangle, \tag{1}$$

with

$$\sum_{i=0}^{2^n-1} |a_i|^2 = 1. \tag{2}$$

In this notation, $|i\rangle$ is the classical state with a string of N 0s and 1s corresponding to the binary representation of the number i. The 2^n states in this sum are mutually orthonormal, so the *inner product*, or overlap, of these states is defined to be $\langle i|j\rangle = \delta_{ij}$. Therefore, the

set of states $\{|0\rangle, |1\rangle, ..., |2^n - 1\rangle\}$ forms a basis for the state of the n qubits, which we call the *computational basis* [6].

Although a system of qubits can store information on multiple states at once through the a_is, we can only obtain information about the the state $|\Psi\rangle$ by performing a measurement on the system. For the purpose of this paper, we will only deal with measurements that project $|\Psi\rangle$ onto the computational basis, although a projection onto any set of 2^n orthonormal states may be performed by applying the correct operations before and after the measurement. When a measurement is performed, the probability of obtaining the state $|i\rangle$ from the measurement is $|\langle\Psi|i\rangle|^2 = |a_i|^2$. Since every such measurement results in obtaining exactly one of the computational basis states, this directly implies (2). However, any measurement also collapses the state of the system: if the state of the system is measured as $|i\rangle$, then after the measurement, the new state of the system is $|i\rangle$, and the old state $|\Psi\rangle$ is lost [5, 6].

Quantum gates are unitary operators that take a state $|\Psi\rangle$ to a new state $U|\Psi\rangle$. An operator U is unitary if and only if it preserves inner products, meaning that

$$\langle U\Psi|U\phi\rangle = \langle\Psi|\phi\rangle. \tag{3}$$

Thus, these operators may also be seen as rotations in 2^n-space. It follows from (3) that unitary operators are invertible and therefore preserve all of the information from the original state, which is a requirement for quantum operators which aren't measurements. Quantum algorithms use unitary operators to transform an initial state into a final state whose measurement will yield a useful result to high or unit probability [5, 6].

2.2 Ordered Search

The ordered search problem is defined as follows:

ORDERED SEARCH: *Given a ordered list of N elements, and an element which appears exactly once in the list, find the index w of this element.*

A simple way to characterize the complexity of this problem is to ask how many times one must query the list to determine w. Thus, one may ignore how the list is actually stored, and instead consider a model of the problem in which a black box, called an *oracle*, tells one whether or not a position is after w. In classical computing, the oracle is a function defined as

$$f_w(i) = \begin{cases} -1 & \text{if } i < w; \\ 1 & \text{if } i \geq w. \end{cases} \tag{4}$$

In quantum computing, this oracle is elevated to an operator F_w such that

$$F_w|i\rangle = f_w(i)|i\rangle \tag{5}$$

for any computational basis state $|i\rangle$. The goal of an algorithm which uses the oracle model is not to construct the oracle, but rather to minimize the number of queries to the oracle once it is given [4].

Classical computers can solve the ordered search problem in $\log_2 N$ queries by the well-known binary search algorithm. In this algorithm, a query is made to f_w at the center of the list, dividing the space to be searched by 2, and the process is repeated. Quantum computers, working in quantum parallel, can use F_w to operate on a superposition of states. This may be used to improve on the $\log_2 N$ queries required by the binary search.

There are many applications in which ordered search is used, the most prominent of which is database search. The ordered search problem has also been shown to have implications in the related problems of inserting elements into and testing for element distinction in ordered lists [3]. Because of how often this problem appears, it is of interest to find an improved algorithm for it, despite the fact that only constant factor improvements are possible. It is

also of theoretical interest, as methods for finding ordered search algorithms may carry over to improved algorithms for other problems.

2.3 Translationally Invariant Algorithms

In [4], Farhi *et al.* give a method for finding quantum algorithms for the ordered search problem, and find exact algorithms which solve the problem for 6 elements in 2 queries and for 52 elements in 3 queries. One may use these base cases to create a recursive solution to the general ordered search problem. Using the algorithm for ordered search of 52 elements in 3 queries, one may repeatedly divide a list of arbitrary size into 52 equal groups and perform the algorithm on the first element of each group, dividing the search space by 52. By repeatedly performing this process, the ordered search problem can thus be solved in $3 \log_{52} N \approx 0.526 \log_2 N$ queries. Note that the base case algorithms must be exact, because if they fail with some probability $P \neq 0$, then the general algorithm will only succeed with probability $(1 - P^{\log_{52} N})$, which goes to zero for N large.

The method used for finding such algorithms considers a slightly different oracle, which is given by

$$F'_w|x\rangle = \begin{cases} F_w|x\rangle & \text{if } 0 \leq x \leq N-1 \\ -F_w|x\rangle & \text{if } N \leq x \leq 2N-1. \end{cases} \tag{6}$$

Clearly, this oracle contains no more information than the one in (5), and can be easily constructed from the first. Using the oracle operator, all k-query algorithms which start in a state $|s\rangle$ will end in a state

$$V_k F'_w V_{k-1} F'_w ... V_1 F'_w |s\rangle. \tag{7}$$

where the V_ℓ are unitary operators applied by the algorithm. Since it is the goal of our algorithm to determine the value of w, we must choose the V_ℓ such that the ending states for (7) are mutually orthogonal for $w = 0, 1, ..., N-1$, as this is the only way to distinguish the

oracles from each other with unit probability. Thus, the task is to find operators V_ℓ which will satisfy this orthogonality condition.

The key idea in [4], which motivates the definition of the oracle in (6), is to restrict the search to operators V_ℓ which are *translationally invariant*. Consider a translation operator T which is defined as

$$T|x\rangle = \begin{cases} |x+1\rangle & \text{if } 0 \le x \le 2N-2 \\ |0\rangle & \text{if } x = 2N-1. \end{cases} \tag{8}$$

We define an operator A to be translationally invariant if $TA|\Psi\rangle = AT|\Psi\rangle$, or in other

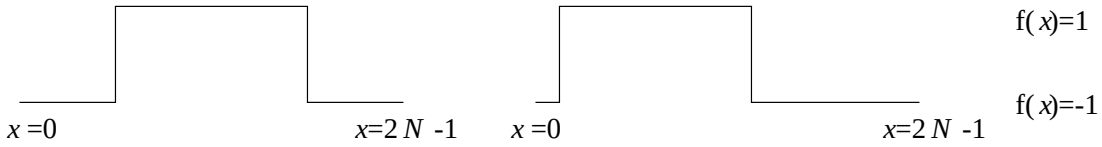

Figure 1: Two possible oracles. Each is equivalent to the other if shifted cyclically, demonstrating that they are translates of each other. The jumps in each are located at $x = w$, and the drops are located at $x = w + N$.

words, if $TAT^{-1} = A$. Note that the oracles are translates of each other, because $TF'_wT^{-1} = F'_{w+1}$ (Figure 1). One of the advantages of restricting attention to translationally invariant algorithms is that if one demands that a successful algorithm starts in the state

$$|s\rangle = \frac{1}{\sqrt{2N}} \sum_{x=0}^{2N-1} |x\rangle \tag{9}$$

and ends in the state

$$\begin{aligned} |w+\rangle &= \frac{1}{\sqrt{2}}(|w\rangle + |w+N\rangle) \quad \text{if } k \text{ is odd}, \\ |w-\rangle &= \frac{1}{\sqrt{2}}(|w\rangle - |w+N\rangle) \quad \text{if } k \text{ is even}, \end{aligned} \tag{10}$$

then $T|s\rangle = |s\rangle$ and $T^w|0\pm\rangle = |w\pm\rangle$, so that if the algorithm is successful for $w = 0$, namely

$$|0\pm\rangle = V_k F_0' V_{k-1} F_0' ... V_1 F_0' |s\rangle, \tag{11}$$

then the algorithm holds for any w using the same V_ℓ:

$$|w\pm\rangle = V_k F_w' V_{k-1} F_w' ... V_1 F_w' |s\rangle. \tag{12}$$

Each translationally invariant operator V_ℓ must be diagonal in the momentum basis

$$|\mathbf{p}\rangle = \frac{1}{\sqrt{2N}} \sum_{x=0}^{2N-1} e^{i\mathbf{p}x\pi/N} |x\rangle, \tag{13}$$

because

$$T|\mathbf{p}\rangle = e^{i\alpha}|\mathbf{p}\rangle \tag{14}$$

for some α. Hence, writing the state of the system after ℓ steps as

$$|\psi_\ell\rangle = \begin{cases} |s\rangle & \text{if } \ell = 0; \\ V_\ell F_w' |\psi_{\ell-1}\rangle & \text{if } \ell > 0, \end{cases} \tag{15}$$

one finds that

$$|\langle \mathbf{p}|\psi_\ell\rangle| = |\langle \mathbf{p}|V_\ell F_w'|\psi_{\ell-1}\rangle|. \tag{16}$$

By (14), $|\langle \mathbf{p}|V_\ell|\Psi\rangle| = |\langle \mathbf{p}|\Psi\rangle|$ for any $|\Psi\rangle$, so (16) is equivalent to

$$|\langle \mathbf{p}|\psi_\ell\rangle| = |\langle \mathbf{p}|F_w'|\psi_{\ell-1}\rangle|. \tag{17}$$

In [4], (17) is expanded in terms of each computational basis vector $|x\rangle$, so that it takes

the form

$$P_\ell(z) = P_{\ell-1}(z) \quad \text{at } |z| = 1 \tag{18}$$

for a set of polynomials $P_\ell(z)$ of degree $N-1$. In other words, finding polynomials P_ℓ satisfying (18) corresponds to finding an exact quantum algorithm for ordered search.

To simplify this search, Farhi *et al.* also define a new set of polynomials $Q_\ell(z) = P_\ell(z)[P_\ell(\frac{1}{z^*})]^*$, which decompose on $|z| = 1$ as

$$Q_\ell(e^{i\theta}) = 1 + A_\ell(\theta) + B_\ell(\theta), \tag{19}$$

where

$$A_\ell(\theta) = \sum_{r=1}^{N-1} a_{l,r} \cos r\theta, \quad a_{l,r} = a_{l,N-r} \tag{20}$$

$$B_\ell(\theta) = \sum_{r=1}^{N-1} b_{l,r} \cos r\theta, \quad b_{l,r} = -b_{l,N-r}. \tag{21}$$

The conditions in (17) in terms of the polynomials Q_ℓ take the form

$$Q_\ell(z) \geq 0 \quad \text{on } |z| = 1 \tag{22}$$

and

$$\begin{aligned} B_0(\theta) &= B_1(\theta) \\ A_1(\theta) &= A_2(\theta) \\ B_2(\theta) &= B_3(\theta), \\ &\vdots \end{aligned} \tag{23}$$

while the starting and ending values of the algorithm in (9) and (10) have become

$$\begin{aligned} A_0(\theta) &= \sum_{r=1}^{N-1} \cos r\theta \\ B_0(\theta) &= \sum_{r=1}^{N-1} \left(1 - \frac{2r}{N}\right) \cos r\theta \end{aligned} \tag{24}$$

and

$$A_k(\theta) = B_k(\theta) = 0. \tag{25}$$

Thus, finding a k-query quantum algorithm for an ordered search of N elements is equivalent to finding functions A_ℓ and B_ℓ which satisfy the starting and ending conditions in (24) and (25), and the equality condition in (23) [4].

3 Improving the Algorithm

Using the equalities in (23), we may rewrite the set of inequalities in (22) as a set of inequalities for A_ℓ and B_ℓ by substituting $A_\ell = A_{\ell+1}$ for ℓ even, and $B_\ell = B_{\ell+1}$ for ℓ odd:

$$\begin{aligned} 1 + A_1(\theta) + B_0(\theta) &\geq 0 \\ 1 + A_1(\theta) + B_2(\theta) &\geq 0 \\ 1 + A_3(\theta) + B_2(\theta) &\geq 0 \\ &\vdots \end{aligned} \tag{26}$$

Since either A_ℓ or B_ℓ (but not both) is needed for each value of ℓ, we may define a new set of functions $\mathbf{C}$ as

$$C_\ell(\theta) = \begin{cases} A_\ell(\theta) & \text{if } \ell \text{ is odd,} \\ B_\ell(\theta) & \text{if } \ell \text{ is even,} \end{cases} \tag{27}$$

with the inequalities in (26) becoming

$$1 + C_\ell(\theta) + C_{\ell+1}(\theta) \geq 0; \quad 0 \leq \ell \leq k-2. \tag{28}$$

Each function C_ℓ has approximately $\frac{N}{2}$ free variables because of the equalities in (20) and (21), so the total dimension of the search space is appromixately $\frac{N}{2}(k-1)$. As N must grow exponentially with k to achieve an improvement over the best known algorithms, we must refine our search in order to find this set of functions for values of $k > 3$.

3.1 Defining a Cost Function

We define a cost function whose minimization will result in finding a solution $\mathbf{C}$ to the conditions in (28). Let $f_\ell(\theta) = 1 + C_\ell(\theta) + C_{\ell-1}(\theta)$. We define our cost function as twice the total negative area under the curves $f_0, f_1, ..., f_{k-2}$ from 0 to π, or

$$cost(\mathbf{C}) = \sum_{\ell=0}^{k-2} \int_{\theta=0}^{\pi} (|f_\ell(\theta)| - f_\ell(\theta))\, d\theta. \tag{29}$$

If each $f_\ell(\theta)$ is everywhere non-negative, then $cost(\mathbf{C}) = 0$. Thus, if the cost is minimized to 0 for a set of functions C, then (22) holds, and these functions will provide an algorithm for the ordered search problem.

An important observation about the search space in which we are looking for $\mathbf{C}$ is that if there exists a set of functions $\mathbf{C}^*$ for which (28) holds, then there are no local minima for $\mathbf{C}$. We may see this by considering some $\mathbf{C}$ for which $cost(\mathbf{C})$ is positive. If we move in the search space towards the solution set $\mathbf{C}^*$, taking $\mathbf{C}$ to $(1-\epsilon)\mathbf{C} + \epsilon\mathbf{C}^*$ with $0 < \epsilon \leq 1$, we will decrease this cost function (Figure 2). This is clear because moving towards $\mathbf{C}^*$ will add a non-negative value to $C_\ell(\theta)$ for every ℓ and θ. Note that the change in the cost function when moving towards $\mathbf{C}^*$ will always be non-zero, as $C^*_\ell(\theta)$ is zero at only finitely

many points for each ℓ. Thus, from any $\mathbf{C}$, if a solution exists for N elements and k queries we may find it by moving in a direction of decreasing cost through the search space until the cost function has been minimized to zero. Additionally, if a local minimum for the cost function has been found and the cost is not equal to zero, we may conclude that there is no solution for k queries and N elements.

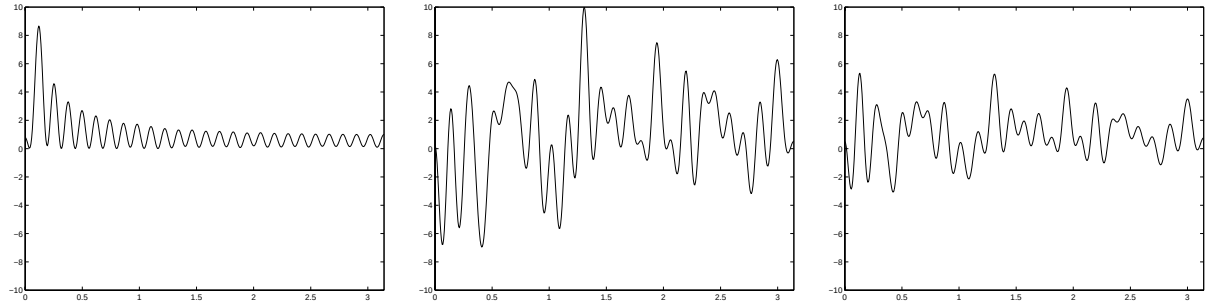

Figure 2: Plots of $f_\ell(\theta)$ for $\mathbf{C}^*$, $\mathbf{C}$, and $\frac{1}{2}(\mathbf{C}^* + \mathbf{C})$ vs. θ (left to right), with $k = 3$ queries and $N = 53$ elements ($\epsilon = \frac{1}{2}$). It is evident from the graphs that the average of the random set $\mathbf{C}$ and the solution set $\mathbf{C}^*$ has a smaller negative area than $\mathbf{C}$, and thus a lower cost function. This supports our observation that moving towards C^* in the search space always decreases the cost function.

3.2 Minimizing the Cost Function

We found that standard methods such as genetic algorithms, simulated annealing, and the simplex method performed poorly in finding solutions to (28) [7]. None of these methods utilize knowledge about the absence of local minima in their search, nor do they utilize our explicit formula for the cost function in (29) to find a direction which will provide a large decrease in cost. We found that zero-temperature annealing performed better than these, as it used the former fact, but it still performed poorly because the direction of movement is still random. The only method we found that performed well was gradient descent with line minimization [7].

Our implementation of this method starts with a set of functions $\mathbf{C}$ satisfying (20) and

(21) and the boundary conditions (24) and (25). It then calculates the gradient $\nabla cost(\mathbf{C})$ from (29), ensuring that the pairs of coefficients $a_{\ell,r}$ and $a_{\ell,N-r}$, and $b_{\ell,r}$ and $b_{\ell,N-r}$, are each treated as a single variable to respect the symmetries in (20) and (21).

Once we have found the gradient, we reduce the problem to a one-dimensional search along this vector to find a new $\mathbf{C}'$ for which $cost(\mathbf{C}') < cost(\mathbf{C})$. We perform this minimization by a process known as bracketing the minimum. Although this method is only useful for finding a local minimum along the line, we have already concluded that there are no local minima in the search space if a solution $\mathbf{C}^*$ exists. Thus, as long as this method decreases the cost function, it doesn't matter whether we move to a local or global minimum along the line of the gradient. The advantage of bracketing the minimum is that it is quick and will always return a local minimum. Our exact implementation of the line minimization algorithm is as follows:

1. Begin with three sets of functions which lie along the gradient $\nabla cost(\mathbf{C})$ from $\mathbf{C}$.

2. Choose a fourth set of functions between the outermost two.

3. Compute the cost of all four sets of functions.

4. Repeat with the three sets of functions which have the smallest costs.

Choosing our initial point to be $\mathbf{C}$ and two functions along the gradient which are sufficiently far away from $\mathbf{C}$, we will in this way obtain a local minimum along the gradient.

Finally, we discovered a heuristic which speeds up this method even further. The solutions for fixed k and nearby N seem to be very closely related. In fact, the coefficients $c_{l,r}$ in the solution $\mathbf{C}^*$ appear to converge to a fixed curve as N becomes large (See Figure 3). Using the solution curves of a given k and N to generate an initial guess for $\mathbf{C}$ for a larger value of N speeds up the search substantially over what would be obtained if a random set of functions were chosen.

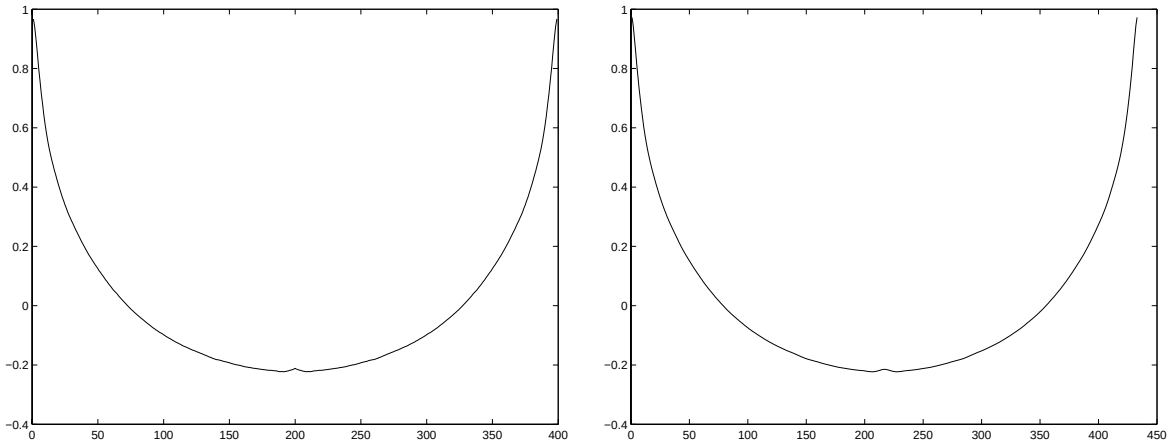

Figure 3: Plots of the coefficients $c_{1,r}$ of C_1 vs. r for $k = 4$ queries and $N = 400$ and $N = 434$ elements (left to right). Despite a difference in the number of coefficients, the curves for the two are nearly identical, with a minimum value of approximately -0.2 near the middle of the curve, and a maximum value of approximately 1 at either end.

3.3 Results

We have found solutions $\mathbf{C}^*$ with $k = 3$ queries for lists of up to size $N = 53$ elements, and with $k = 4$ queries for lists of up to size $N = 434$ elements. The latter of these can be converted into a recursive quantum algorithm that solves the ordered search problem in $4 \log_{434} N \approx 0.457 \log_2 N$ queries, providing a speedup over the quantum algorithms found in [3] and [4]. Neither of these are necessarily optimal solutions for their respective values of k, and better solutions are likely to follow as there is more time to run the program (for $k = 4$ queries and $N = 434$ elements, our search takes approximately 1.5 hours on a Pentium 4 1.80 GHz machine).

The graphs of the coefficients $c_{l,r}$ for the solution functions follow patterns which suggest that a closed form for these functions may exist (See Figures 4 and 5). Such a solution would be helpful in establishing optimal solutions for each k, and in doing so would likely find even better quantum algorithms for the problem. The graphs of the cost function $f_\ell(\theta)$ also suggest the problem has further structure, and could provide additional clues as to a closed

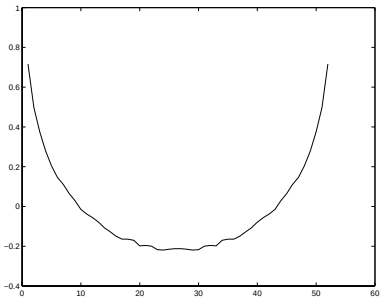

Figure 4: A plot of the coefficients $c_{l,r}$ of C_1 vs. r for $k = 3$ queries and $N = 53$ elements.

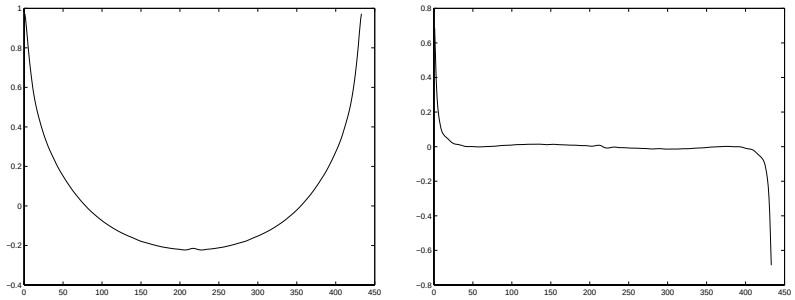

Figure 5: Plots of the coefficients $c_{l,r}$ of C_1 and C_2 vs. r (left to right) for $k = 4$ queries and $N = 434$ elements.

form for the coefficients for solutions (See Figures 6 and 7).

4 Conclusion

We have discovered a quantum algorithm that can search an ordered list of N items by making only $4 \log_{434} N \approx 0.457 \log_2 N$ queries, improving on the previous best algorithm, which requires $0.526 \log_2 N$ queries [4]. We have developed a cost function containing no local minima whose global minimum corresponds to a translationally invariant recursive quantum algorithm for this problem, and have presented the possibility of a universal curve for minima of this cost function. We believe further improvement is possible, although perhaps not quite as far as the best lower bound of $0.22 \log_2 N$ queries for this problem [3].

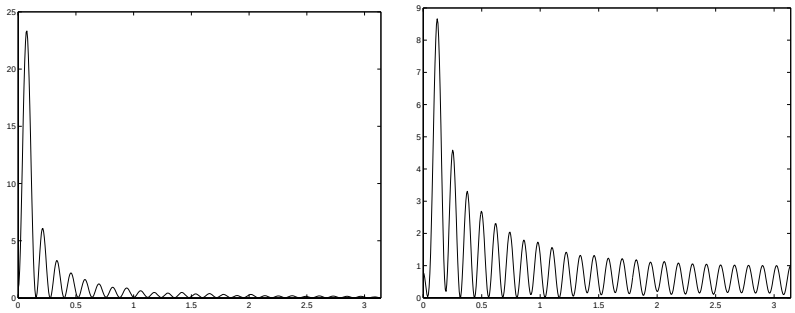

Figure 6: Plots of $f_1(\theta)$ and $f_2(\theta)$ vs. θ (left to right) for $k = 3$ queries and $N = 53$ elements.

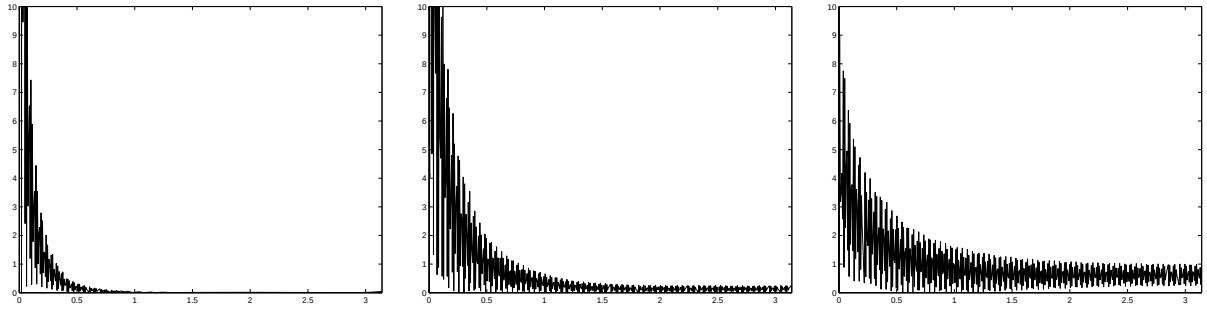

Figure 7: Plots of $f_1(\theta)$, $f_2(\theta)$, and $f_3(\theta)$ vs. θ (left to right) for $k = 4$ queries and $N = 434$ elements.

5 Acknowledgements

I would like to thank Dr. Andrew Landahl for his overwhelming help and support in my research, as well as Dr. Edward Farhi, Dr. Jeffrey Goldstone, and Dr. Sam Gutmann for useful discussions and suggestions. I would also like to thank those that have helped me to revise my paper, including Jeremy England and Ben Rahn. Finally, I would like to thank the Center for Excellence in Education for giving me the opportunity to do this research.

References

[1] Grover, Lov K. A fast quantum mechanical algorithm for database search. *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing* (1996). 212-219.

[2] Boyer, M., Brassard, G., Høyer, P., and Tapp, A. Tight Bounds on Quantum Searching. *Fortsch. Phys.* **46** (1998). 493-506.

[3] Høyer, P., Neerbek, J., and Shi, Y. Quantum Complexities of Ordered Searching, Sorting, and Element Distinctness. *Algorithmica*. **34** (2002). 429-448.

[4] Farhi, E., Goldstone, J., Gutmann, S., and Sipser, M. Invariant Quantum Algorithms for Insertion into an Ordered List. MIT CTP #2815 (1999).

[5] Preskill, John. *Lecture Notes for Physics 229: Quantum Information and Computation* (1998). http://www.theory.caltech.edu/people/preskill/ph229/

[6] Nielsen, M., and Chuang, I. *Quantum Computation and Quantum Information* (2000). Cambridge, UK. Cambridge University Press.

[7] Flannery, B., Press, W., Teukolsky, S., and Vetterling, W. *Numerical Recipes in C* (1988). Cambridge, UK. Cambridge University Press.