Quantum complexities of ordered searching, sorting, and element distinctness*

Peter Høyer[‡] Jan Neerbek[§] Yaoyun Shi[¶]

September 17, 2001

Abstract

We consider the quantum complexities of the following three problems: searching an ordered list, sorting an un-ordered list, and deciding whether the numbers in a list are all distinct. Letting N be the number of elements in the input list, we prove a lower bound of $\frac{1}{\pi}(\ln(N) - 1)$ accesses to the list elements for ordered searching, a lower bound of $\Omega(N \log N)$ binary comparisons for sorting, and a lower bound of $\Omega(\sqrt{N} \log N)$ binary comparisons for element distinctness. The previously best known lower bounds are $\frac{1}{12} \log_2(N) - O(1)$ due to Ambainis, $\Omega(N)$, and $\Omega(\sqrt{N})$, respectively. Our proofs are based on a weighted all-pairs inner product argument.

In addition to our lower bound results, we give an exact quantum algorithm for ordered searching using roughly $0.631 \log_2(N)$ oracle accesses. Our algorithm uses a quantum routine for traversing through a binary search tree faster than classically, and it is of a nature very different from a faster exact algorithm due to Farhi, Goldstone, Gutmann, and Sipser.

Keywords: Quantum computation, Searching, Sorting, Element distinctness, Lower bound.

1 Introduction

The speedups of quantum algorithms over classical algorithms have been a main reason for the current interest in quantum computing. One central question regarding the power of quantum computing is: How much speedup is possible? Although dramatic speedups seem

* Research supported by Canada's NSERC, the EU fifth framework program QAIP, IST-1999-11234, the National Science Foundation under grant CCR-9820855, and the Pacific Institute for the Mathematical Sciences.

[‡] Department of Computer Science, University of Calgary, Alberta, Canada T2N 1N4. email: hoyer@cpsc.ucalgary.ca. Research conducted in part while at BRICS, University of Aarhus, Denmark.

[§] Department of Computer Science, University of Aarhus, DK–8000 Århus C, Denmark. email: neerbek@daimi.au.dk.

[¶] Department of Computer Science, Princeton University, Princeton, NJ 08544, USA. email: shiyy@cs.princeton.edu.

possible, as in the case of Shor's [23] algorithms for factoring and finding discrete logarithms, super-polynomial speedups are so far proven only in restricted models such as the black box model.

In the black box model, the input is given as a black box, so that the only way the algorithm can obtain information about the input is via queries, and the complexity measure is the number of queries. Many problems that allow provable quantum speedups can be formulated in this model, an example being the unordered search problem considered by Grover [18]. Several tight lower bounds are now known for this model, most of them being based on techniques introduced in [6, 4, 2].

We study the quantum complexities of the following three problems.

Ordered searching Given a list of numbers $x = (x_0, x_1, \ldots, x_{N-1})$ in non-decreasing order and some number y, find the minimal i such that $y \leq x_i$. We assume that $y \leq x_{N-1}$ so that the problem is always well-defined.

Sorting Given a list of numbers $x = (x_0, x_1, \ldots, x_{N-1})$, output a permutation σ on the set $\{0, \ldots, N-1\}$ so that the list $(x_{\sigma(0)}, x_{\sigma(1)}, \ldots, x_{\sigma(N-1)})$ is in non-decreasing order.

Element distinctness Given a list of numbers $x = (x_0, x_1, \ldots, x_{N-1})$, are they all distinct?

These problems are closely related and are among the most fundamental and most studied problems in algorithmics. They can also be formulated naturally in the black box model. For the ordered searching problem, we consider queries of the type "$x_i =?$", to which the oracle returns the value x_i, and for the sorting and element distinctness problems, we consider queries of the type "Is $x_i < x_{i'}$?", which are simply binary comparisons. Let $H_i = \sum_{k=1}^{i} \frac{1}{k}$ denote the ith harmonic number. We prove a lower bound for each of these three problems.

Theorem 1 *Any quantum algorithm for ordered searching that errs with probability at most $\epsilon \geq 0$ requires at least*

$$\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right)\frac{1}{\pi}(H_N - 1) \tag{1}$$

queries to the oracle. In particular, any exact quantum algorithm requires more than $\frac{1}{\pi}(\ln(N) - 1) \approx 0.220 \log_2 N$ queries.

Theorem 2 *Any comparison-based quantum algorithm for sorting that errs with probability at most $\epsilon \geq 0$ requires at least*

$$\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right)\frac{N}{2\pi}(H_N - 1) \tag{2}$$

comparisons. In particular, any exact quantum algorithm requires more than $\frac{N}{2\pi}(\ln(N) - 1) \approx 0.110 N \log_2 N$ comparisons.

Theorem 3 *Any comparison-based quantum algorithm for element distinctness that errs with probability at most $\epsilon \geq 0$ requires at least*

$$\left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right)\frac{\sqrt{N}}{2\pi}(H_N - 1) \tag{3}$$

comparisons.

The previously best known quantum lower bound for ordered searching is $\frac{1}{12}\log_2(N) - O(1)$, due to Ambainis [1]. For comparison-based sorting and element distinctness, the previously best known quantum lower bounds are $\Omega(N)$ and $\Omega(\sqrt{N})$, respectively, both of which can be proven in many ways.

We prove our lower bounds by utilizing what we refer to as a weighted all-pairs inner product argument, or a probabilistic adversary argument. This proof technique is based on the work of Bennett, Bernstein, Brassard, and Vazirani [6] and Ambainis [2].

Farhi, Goldstone, Gutmann, and Sipser give in [15] an exact quantum algorithm for ordered searching using roughly $0.526\log_2(N)$ queries. We provide an alternative quantum algorithm that is also exact and uses $\log_3(N) + O(1) \approx 0.631\log_2(N)$ queries. Our construction is radically different from the construction proposed in [15], and these are the only constructions known leading to quantum algorithms using at most $c\log_2(N)$ queries for some constant c strictly less than 1.

Whereas most quantum algorithms are based on Fourier transforms and amplitude amplification [9], our algorithm is based on binary search trees. We initiate several applications of the binary search algorithm in quantum parallel and let them find the element we are searching for in teamwork. By cooperating, these applications can traverse the binary search tree faster than classically, hereby reducing the complexity from $\log_2(N)$ to roughly $\log_3(N)$.

There are at least three reasons why the quantum complexities of the three problems are of interest. Firstly because of their significance in algorithmics in general. Secondly because these problems possess some symmetries and periodicities of a different nature than other studied problems in quantum algorithmics. Determining symmetries and periodicities seems to be a primary ability of quantum computers and it is not at all clear how far-reaching this skill is. Thirdly because searching and sorting represent non-Boolean non-symmetric functions. A (partial) function is said to be symmetric if it is invariant under permutation of its input. Only few non-trivial quantum bounds for non-Boolean and non-symmetric functions are known.

The rest of the paper is organized as follows. We first discuss the model in Section 2, prove a general lower bound in Section 3, and then apply it to ordered searching, sorting, and element distinctness in Sections 4, 5, and 6, respectively. We give our quantum algorithm in Section 7 and conclude in Section 8.

2 Quantum black box computing

We use the so-called black box model in which the input is given as an oracle and our measure of complexity is the number of queries to the oracle [7, 4]. Fix some positive integer $N > 0$. The input $x = (x_0, \ldots, x_{N-1}) \in \{0,1\}^N$ is given as an oracle, and the only way we can access the bits of the oracle is via queries. A query implements the operator

$$\mathsf{O}_x : \quad |z;i\rangle \longmapsto \begin{cases} (-1)^{x_i}|z;i\rangle & \text{if } 0 \le i < N \\ |z;i\rangle & \text{if } i \ge N. \end{cases} \tag{4}$$

Here i and z are non-negative integers. By a query to oracle x we mean an application of the unitary operator O_x. We sometimes refer to O_x as the oracle. A quantum algorithm A

that uses T queries to an oracle O is a unitary operator of the form

$$\mathsf{A} = (\mathsf{UO})^T\mathsf{U}. \tag{5}$$

We always apply algorithm A on the initial state $|0\rangle$, and after applying A, we always measure the final state in the computational basis. Thus, a quantum algorithm for oracle quantum computing is defined by specifying a unitary operator U and a number of iterations T. Our model for oracle quantum computing is slightly different from, but equivalent to, the "standard" model used for example in [4]. We favor utilizing this model, since hereby oracle O_x is a diagonal matrix with respect to the computational basis.

Consider the computation of some function $f : \{0,1\}^N \to \{0,1\}^m$. After applying quantum algorithm A on $|0\rangle$, we measure the m rightmost qubits of $\mathsf{A}|0\rangle$ and output the outcome w. The success probability p_x of A on input $x \in \{0,1\}^N$ is defined as the probability that $w = f(x)$. For complete functions $f : \{0,1\}^N \to \{0,1\}^m$, we define the success probability of A as the minimum of p_x over all $x \in \{0,1\}^N$. For partial functions $f : S \to \{0,1\}^m$, where $S \subseteq \{0,1\}^N$, we take the minimum over S only.

There is a very basic, yet key conceptual idea which is used in this paper and which we would like to emphasize, and that is that we are concerned about *distinguishing* oracles rather than *determining* the value of the function f. This idea is not new, nor revolutionary; we do however feel that it helps in developing and conveying bounds on algorithms. For a given problem we want to identify the *pairs of oracles* that are hard to *distinguish*. To capture the hardness of distinguishing each pair of oracles, we therefore introduce a *weight function*

$$\omega : \quad \{0,1\}^N \times \{0,1\}^N \to \mathbb{R}_+ \tag{6}$$

that takes non-negative real values. The harder an oracle x is to distinguish from an oracle y, the more weight we put on the pair (x, y). The total weight W distributed is the sum of $\omega(x, y)$ over all pairs $(x, y) \in \{0,1\}^N \times \{0,1\}^N$. We do not want to put any restrictions on ω in general, though for many applications we probably want ω to be symmetric, normalized and take the value 0 along the diagonal.

The weight function allows us to easily capture any complete as well as partial function. Let $f : S \to \{0,1\}^m$ be the function of interest, where $S \subseteq \{0,1\}^N$. We say that ω *is a weight function for* f if whenever $f(x) = f(y)$ then $\omega(x, y) = 0$, and if for every pair $(x, y) \notin S \times S$ we have $\omega(x, y) = 0$. Hereby, we may ignore f and just consider the scenario in which we are given weight function ω.

3 General lower bound

The first general technique for proving lower bounds for quantum computing was introduced by Bennett, Bernstein, Brassard and Vazirani in their influential paper [6]. Their beautiful technique is nicely described in Vazirani's exposition [24]. Our technique is a natural generalization of theirs as well as of Ambainis' powerful entanglement lower bound approach recently proposed in [2].

Here is the basic idea: Consider a quantum algorithm $\mathsf{A} = (\mathsf{UO})^T\mathsf{U}$ that we use to distinguish between two oracles $x, y \in \{0,1\}^N$. Our initial state is $|0\rangle$. After j iterations,

our state is $|\psi_x^j\rangle = (\mathsf{U}\mathsf{O}_x)^j\mathsf{U}|0\rangle$ if we are given oracle x, and it is $|\psi_y^j\rangle = (\mathsf{U}\mathsf{O}_y)^j\mathsf{U}|0\rangle$ if we are given oracle y. Two quantum states are distinguishable with high probability if and only if they are almost orthogonal. If the states $|\psi_x^j\rangle$ and $|\psi_y^j\rangle$ have large overlap, then they cannot be distinguished with high probability, and hence more queries are required. If a query can separate two states $|\psi_x^j\rangle$ and $|\psi_y^j\rangle$ by only a small additional amount, then many queries are required.

We have to choose how to measure the overlap of states among the plentiful studied measures. We pick here the probably simplest possibility: inner products. Two states can be distinguished with certainty if and only if their inner product is zero. Furthermore, two states can be distinguished with high probability if and only if their inner product is of small absolute value.

Lemma 4 *Suppose that we are given one of two states $|\Psi_x\rangle, |\Psi_y\rangle$. There exists some measurement that will correctly determine which of the two states we are given with error probability at most ϵ if and only if $|\langle\Psi_x|\Psi_y\rangle| \leq 2\sqrt{\epsilon(1-\epsilon)}$.*

We are not only interested in distinguishing two particular oracles, but many oracles, and thus we use an "all-pairs inner product" measure. But as we discussed in the previous section, some oracles are harder to distinguish than others, and this leads us to our final choice: we use an *all-pairs inner product measure weighted by ω*. We now formalize this approach.

Let $\mathsf{A} = (\mathsf{U}\mathsf{O})^T\mathsf{U}$ be any quantum algorithm. For every oracle $x \in \{0,1\}^N$ and every integer $j \geq 0$, let

$$|\psi_x^j\rangle = (\mathsf{U}\mathsf{O}_x)^j\mathsf{U}\,|0\rangle \tag{7}$$

denote the state of the computer after applying j iterations using oracle O_x. For every integer $j \geq 0$, let

$$W_j = \sum_{x,y\in\{0,1\}^N} \omega(x,y)\ \langle\psi_x^j|\psi_y^j\rangle. \tag{8}$$

denote the weighted all-pairs inner product after j iterations. Initially, the total weight is $W = W_0$. After T iterations, the total weight is $W_T = \sum_{x,y\in\{0,1\}^N} \omega(x,y)\langle\psi_x^T|\psi_y^T\rangle$. If algorithm A is capable of distinguishing with certainty between all pairs of oracles $(x,y) \in \{0,1\}^N \times \{0,1\}^N$ of nonzero weight, then $W_T = 0$. Conversely, if $W_T > 0$ then there exists some pair of oracles (x,y) with $\omega(x,y) > 0$ between which algorithm A does not distinguish perfectly.

In summary, initially all inner products are 1 and the initial weight is therefore W, whereas at the end of the computation all inner products are hopefully small and the final weight W_T is therefore small. If the total weight can decrease by at most Δ by each query, we require at least W/Δ queries to perfectly distinguish between all pairs of oracles of nonzero weight.

Theorem 5 *Let $f : S \to \{0,1\}^m$ be a given function where $S \subseteq \{0,1\}^N$, and let ω be a weight function for f. Let $\mathsf{A} = (\mathsf{U}\mathsf{O})^T\mathsf{U}$ be any quantum algorithm that computes f with error at most $\epsilon \geq 0$ using T queries. Then*

$$T \geq \left(1 - 2\sqrt{\epsilon(1-\epsilon)}\right)\frac{W}{\Delta} \tag{9}$$

where $W = \sum_{x,y\in\{0,1\}^N} \omega(x,y)$ *denotes the initial weight, and* Δ *is an upper bound on* $|W_j - W_{j+1}|$ *for all* $0 \le j < T$.

Proof By definition, $W_0 = W$, and by Lemma 4, $|W_T| \le 2\sqrt{\epsilon(1-\epsilon)}W$. Write $W_0 - W_T = \sum_{j=0}^{T-1}(W_j - W_{j+1})$ as a telescoping sum. Then $|W_0 - W_T| \le \sum_{j=0}^{T-1} |W_j - W_{j+1}| \le T\Delta$, and the theorem follows. □

Our formulation of Theorem 5 has been heavily inspired by the general formulations used by Ambainis in [2]. In [6], Bennett, Bernstein, Brassard, and Vazirani are interested in distinguishing one unique oracle x' from all other oracles. That is, for every pair of oracles $(x,y) \in \{0,1\}^N \times \{0,1\}^N$ of interest, we have $x = x'$. Ambainis [2] removes this restriction, and he also allows a non-uniform interest in different oracles by weighting each oracle individually. We are also interested in distinguishing general pairs of oracles, but we discriminate our interest in each pair by weighting each *pair of oracles* via weight function ω. This discrimination is essential when applying it to ordered searching, sorting, and element distinctness.

Let $L = [\alpha_{k,l}]_{1\le k,l<\infty}$ be the Hilbert matrix with $\alpha_{k,l} = 1/(k+l-1)$. That is,

$$L = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & & \\ \frac{1}{3} & \frac{1}{4} & & & \\ \frac{1}{4} & & & & \\ \vdots & & & & \end{bmatrix}.$$

Let $\|\cdot\|_2$ be the spectral norm, i.e., for any complex matrix $M \in \mathbb{C}^{m\times m}$,

$$\|M\|_2 := \max_{v\in\mathbb{C}^m, \|v\|_2=1} \|Mv\|_2.$$

Our lower bound proofs make use of the following property of the Hilbert matrix.

Lemma 6 $\|L\|_2 = \pi$.

Choi [12] has an elegant proof of this lemma.

4 Lower bound for ordered searching

Searching ordered lists is a non-Boolean promise problem: the list is promised to be sorted, and the answer is an index, not a bit. The input is a sorted Boolean list of size N and the problem is to find the index of the leftmost 1. We assume that not all values of the list are 0 and hence the problem is always well-defined. Formally, the set S of the N possible inputs consists of all $x \in \{0,1\}^N$ for which $x_{N-1} = 1$ and $x_{i-1} \le x_i$ for all $1 \le i < N$. The search function $f : S \to \{0,1\}^m$ is defined by $f(x) = \min\{0 \le i < N \mid x_i = 1\}$, where we identify the result $f(x)$ with its binary encoding as a bit-string of length $m = \lceil \log_2(N) \rceil$.

The first lower bound of $\Omega(\sqrt{\log(N)}/\log\log(N))$ was proved by Buhrman and de Wolf [11] by an ingenious reduction from the OR problem. Farhi, Goldstone, Gutmann,

and Sipser [14] improved this to $\log_2(N)/2\log_2\log_2(N)$, and Ambainis [1] then proved the previously best known lower bound of $\frac{1}{12}\log_2(N) - O(1)$. In [14, 1], they use, as we do here, an inner product argument along the lines of [6].

The first and essential step in our lower bound is to pick a good weight function ω for f. We choose

$$\omega(x,y) = \begin{cases} \frac{1}{f(y)-f(x)} & \text{if } (x,y) \in S \times S \text{ and } f(x) < f(y) \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

That is, we use the inverse of the Hamming distance of x and y. Intuitively, a weight function that (only) depends on the Hamming distance ought to be a good choice since it can put most weight on pairs of oracles that are almost identical.

The initial weighted all-pairs inner product is

$$W_0 = \sum_{x,y\in\{0,1\}^N} \omega(x,y) = \sum_{i=1}^{N-1} H_i = NH_N - N, \tag{11}$$

where $H_i = \sum_{k=1}^{i} \frac{1}{k}$ denotes the ith harmonic number. Note that $\ln(N) < H_N < \ln(N) + 1$ for all $N > 1$. Since any query can decrease the weighted all-pairs inner product by at most πN, Theorem 1 follows by applying Theorem 5.

Lemma 7 *For weight function ω defined by Equation 10, we have that*

$$|W_j - W_{j+1}| \leq \pi N$$

for all $0 \leq j < T$.

Proof For any oracle $x \in \{0,1\}^N$, we think of x as an infinite bit-string where $x_i = 0$ for all $i \geq N$. Operator O_x defined by Equation 4 is then given by

$$\mathsf{O}_x = \sum_{z\geq 0} \sum_{i\geq 0} (-1)^{x_i} |z;i\rangle\langle z;i|.$$

Let I denote the identity operator. For every $i \geq 0$, let $\mathsf{P}_i = \sum_{z\geq 0} |z;i\rangle\langle z;i|$ denote the projection operator onto the subspace querying the ith oracle bit.

Let $0 \leq j < T$. By definition

$$\begin{aligned} W_j - W_{j+1} &= \sum_{x,y\in\{0,1\}^N} \omega(x,y)\ \langle\psi_x^j|\psi_y^j\rangle - \sum_{x,y\in\{0,1\}^N} \omega(x,y)\ \langle\psi_x^{j+1}|\psi_y^{j+1}\rangle \\ &= \sum_{x,y\in\{0,1\}^N} \omega(x,y)\ \langle\psi_x^j|\mathsf{I} - \mathsf{O}_x^{-1}\mathsf{O}_y|\psi_y^j\rangle \\ &= 2 \sum_{x,y\in\{0,1\}^N} \sum_{i:x_i\neq y_i} \omega(x,y)\ \langle\psi_x^j|\mathsf{P}_i|\psi_y^j\rangle. \end{aligned}$$

For every $0 \leq a < N$ and $i \geq 0$, let $\beta_{a,i} = \|\mathsf{P}_i|\psi_x^j\rangle\|$ denote the ℓ_2 norm of the projection of $|\psi_x^j\rangle$ onto the subspace querying the ith oracle bit, where $x \in \{0,1\}^N$ is such that $f(x) = a$ ('a' for 'answer'). Then

$$W_j - W_{j+1} \leq 2 \sum_{0\leq a<b<N} \sum_{a\leq i<b} \frac{1}{b-a} \beta_{a,i}\beta_{b,i}.$$

Rewrite the above equation in terms of distances $d = b - a$,

$$W_j - W_{j+1} \leq 2 \sum_{d=1}^{N-1} \sum_{i=0}^{d-1} \frac{1}{d} \left(\sum_{a=0}^{N-d-1} \beta_{a,a+i} \beta_{a+d,a+i} \right).$$

For every $0 \leq i < N-1$, let

$$\gamma_i = \left(\sum_{a=0}^{N-1} \beta_{a,a+i}^2 \right)^{1/2} \quad \text{and} \quad \delta_i = \left(\sum_{a=0}^{N-1} \beta_{a,a-i-1}^2 \right)^{1/2}$$

denote the total mass that queries the oracle at i index-positions above and below the leftmost 1. By the Cauchy–Schwarz inequality,

$$|W_j - W_{j+1}| \leq 2 \sum_{d=1}^{N-1} \sum_{i=0}^{d-1} \frac{1}{d} \gamma_i \delta_{d-i-1}.$$

The right hand side is the written-out product of 3 matrices. Let $\gamma = [\gamma_0, \ldots, \gamma_{N-2}]$ and $\delta = [\delta_0, \ldots, \delta_{N-2}]^t$, where t denotes transposition, and let K denote the $(N-1) \times (N-1)$ matrix with entry (k, l) defined by

$$(K)_{(k,l)} = \begin{cases} \frac{1}{k+l+1} & \text{if } k + l < N - 1 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

for all $0 \leq k, l < N-1$. Then

$$|W_j - W_{j+1}| \leq 2\gamma K \delta \leq 2 \|\gamma\|_2 \cdot \|\!|K|\!\|_2 \cdot \|\delta\|_2.$$

Since $\|\gamma\|_2^2 + \|\delta\|_2^2 \leq \sum_{a=0}^{N-1} \sum_{i \geq 0} \beta_{a,i}^2 \leq N$, we have that $\|\gamma\|_2 \|\delta\|_2 \leq \frac{1}{2} N$. Since K can be obtained from a submatrix of the Hilbert matrix L by setting some entries equal to 0, and since all entries in K and L are nonnegative, $\|\!|K|\!\|_2 \leq \|\!|L|\!\|_2 = \pi$ by Lemma 6. Hence $|W_j - W_{j+1}| \leq \pi N$. □

5 Lower bound for sorting

We show that any quantum algorithm requires at least $\frac{N}{2\pi}(\ln(N) - 1)$ *comparisons* for sorting. To prove this, it suffices to assume that the N numbers to be sorted, $x = (x_0, \ldots, x_{N-1})$, correspond to some permutation σ on $\{0, \ldots, N-1\}$, that is, $x_i = \sigma(i)$ for every $0 \leq i < N$. We assume that the input is the $N \times N$ comparison matrix M_σ for σ defined by

$$(M_\sigma)_{ii'} = \begin{cases} 1 & \text{if } \sigma(i) < \sigma(i') \\ 0 & \text{otherwise.} \end{cases}$$

That is, the input to the quantum algorithm is a comparison matrix given as an oracle. The output is σ. To simplify notation, we sometimes identify the input M_σ with the underlying permutation σ.

The most crucial step in proving the lower bound is again to pick an appropriate weight function ω. Intuitively, it is hard to distinguish two inputs that are identical almost everywhere. Consider an input x. Let k and d be non-negative integers satisfying that $0 \leq k + d \leq n - 1$, and let x_u be the element of rank $k + d$ in x. (The rank of x_u is defined to be the number of elements in x of value strictly smaller than x_u. The rank of a smallest element is 0.) Then we may form a new input y by replacing x_u by a new element of rank k. The element in x that has rank i then has rank $i+1$ in y, for all $k \leq i \leq k+d-1$. The difference d of the ranks of the elements at index u in x and y, respectively, is a refined measure of the difficulty in distinguishing between inputs x and y. As for searching, we use the inverse of d in our weight function, which we now formally define.

We require the following definition. For every permutation σ on $\{0, \ldots, N-1\}$, and every integers $0 \leq k \leq N-2$ and $1 \leq d < N-k$, define a new permutation,

$$\sigma^{(k,d)} = (k, k+1, \ldots, k+d) \circ \sigma. \tag{13}$$

For any permutation σ, denote the inverse permutation of σ by σ^{-1}. If $\tau = \sigma^{(k,d)}$, then

$$\sigma^{-1}(i) = \begin{cases} \tau^{-1}(k) & \text{if } i = k+d \\ \tau^{-1}(i+1) & \text{if } k \leq i \leq k+d-1 \\ \tau^{-1}(i) & \text{otherwise.} \end{cases}$$

This implies that the comparison matrices M_σ and M_τ differ only on the following pairs of entries,

$$\left\{\sigma^{-1}(k+d), \sigma^{-1}(i)\right\} = \left\{\tau^{-1}(k), \tau^{-1}(i+1)\right\} \tag{14}$$

for $k \leq i \leq k+d-1$. Thus, the comparison matrices M_σ and M_τ differ on exactly $2d$ entries.

Informally, if M_σ corresponds to some list x, then M_τ corresponds to the list y obtained by replacing the element of rank $k+d$ in x by a new element of rank k, as described above. The only way the algorithm can distinguish σ from τ is by comparing the element of rank $k+d$ in x with one of the d elements of rank i for some $k \leq i \leq k+d-1$.

We choose the following weight function,

$$\omega(\sigma, \tau) = \begin{cases} \frac{1}{d} & \text{if } \tau = \sigma^{(k,d)} \text{ for some } k \text{ and } d \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

Then the initial weighted all-pairs inner product is

$$W_0 = \sum_{\sigma,\tau} \omega(\sigma, \tau) = \sum_{\sigma} \sum_{k=0}^{N-2} \sum_{d=1}^{N-1-k} \frac{1}{d} = N!(NH_N - N). \tag{16}$$

Since any query can decrease the weighted all-pairs inner product by at most $2\pi N!$, Theorem 2 follows by applying Theorem 5.

Lemma 8 *For weight function ω defined by Equation 15, we have that*

$$|W_j - W_{j+1}| \leq 2\pi N!$$

for all $0 \leq j < T$.

The proof of Lemma 8 is similar to that of Lemma 7.

Proof Query operator O_σ is given by

$$\mathsf{O}_\sigma = \sum_{z\geq 0}\sum_{i,i'\geq 0}(-1)^m|z;i,i'\rangle\langle z;i,i'|,$$

where $m = (M_\sigma)_{ii'}$ denotes the outcome of comparing the ith element with the (i')th element for $0 \leq i, i' < N$, and where $m = 0$ whenever $i \geq N$ or $i' \geq N$. For every pair $\{i, i'\}$ of indices with $0 \leq i, i' < N$, let

$$\mathsf{P}_{ii'} = \sum_{z\geq 0}|z;i,i'\rangle\langle z;i,i'| + \sum_{z\geq 0}|z;i',i\rangle\langle z;i',i|$$

denote the projection operator onto the subspace comparing the ith and (i')th elements.

Let $0 \leq j < T$. By definition

$$\begin{aligned} W_j - W_{j+1} &= \sum_{\sigma,\tau}\omega(\sigma,\tau)\,\langle\psi_\sigma^j|\psi_\tau^j\rangle - \sum_{\sigma,\tau}\omega(\sigma,\tau)\,\langle\psi_\sigma^{j+1}|\psi_\tau^{j+1}\rangle \\ &= \sum_{\sigma,\tau}\omega(\sigma,\tau)\,\langle\psi_\sigma^j|\mathsf{I} - \mathsf{O}_\sigma^{-1}\mathsf{O}_\tau|\psi_\tau^j\rangle \\ &= 2\sum_{\sigma}\sum_{k=0}^{N-2}\sum_{d=1}^{N-1-k}\sum_{i=0}^{d-1}\frac{1}{d}\,\langle\psi_\sigma^j|\mathsf{P}_{\sigma^{-1}(k+d),\sigma^{-1}(k+i)}|\psi_{\sigma(k,d)}^j\rangle. \end{aligned}$$

Rewrite the above equation in terms of distances,

$$W_j - W_{j+1} = 2\sum_{d=1}^{N-1}\sum_{i=0}^{d-1}\frac{1}{d}\left(\sum_{\sigma}\sum_{k=0}^{N-d-1}\,\langle\psi_\sigma^j|\mathsf{P}_{\sigma^{-1}(k+d),\sigma^{-1}(k+i)}|\psi_{\sigma(k,d)}^j\rangle\right). \tag{17}$$

The absolute value of the inner sum is by the Cauchy–Schwarz inequality upper bounded by

$$\sqrt{\sum_{\sigma}\sum_{k=0}^{N-d-1}\left\|\mathsf{P}_{\sigma^{-1}(k+d),\sigma^{-1}(k+i)}|\psi_\sigma^j\rangle\right\|^2}\sqrt{\sum_{\sigma}\sum_{k=0}^{N-d-1}\left\|\mathsf{P}_{\sigma^{-1}(k+d),\sigma^{-1}(k+i)}|\psi_{\sigma(k,d)}^j\rangle\right\|^2}, \tag{18}$$

which is equal to

$$\sqrt{\sum_{\sigma}\sum_{k=0}^{N-d-1}\left\|\mathsf{P}_{\sigma^{-1}(k+d),\sigma^{-1}(k+i)}|\psi_\sigma^j\rangle\right\|^2}\sqrt{\sum_{\tau}\sum_{k=0}^{N-d-1}\left\|\mathsf{P}_{\tau^{-1}(k),\tau^{-1}(k+i+1)}|\psi_\tau^j\rangle\right\|^2}. \tag{19}$$

For every $1 \leq i < N$, let

$$\begin{aligned} \gamma_i &= \sqrt{\sum_{\sigma}\sum_{l=0}^{N-1}\left\|\mathsf{P}_{\sigma^{-1}(l),\sigma^{-1}(l+i)}|\psi_\sigma^j\rangle\right\|^2} \\ \delta_i &= \sqrt{\sum_{\tau}\sum_{l=0}^{N-1}\left\|\mathsf{P}_{\tau^{-1}(l),\tau^{-1}(l-i)}|\psi_\tau^j\rangle\right\|^2}, \end{aligned}$$

where we let l range from 0 to $N-1$ and simply set the thus caused undefined projection operators to be zero operators.[1] Then

$$|W_j - W_{j+1}| \leq 2 \sum_{d=1}^{N-1} \sum_{i=1}^{d} \frac{1}{d} \gamma_i \delta_{d-i+1}.$$

Let $\gamma = [\gamma_1, \ldots, \gamma_{N-1}]$ and $\delta = [\delta_1, \ldots, \delta_{N-1}]^t$. Then, in analogy with the proof of Lemma 7,

$$|W_j - W_{j+1}| \leq 2\pi \|\gamma\|_2 \cdot \|\delta\|_2.$$

Since $\|\gamma\|_2^2 \leq N!$ and $\|\delta\|_2^2 \leq N!$, we conclude that $|W_j - W_{j+1}| \leq 2\pi N!$. □

6 Lower bound for element distinctness

Our lower bound for element distinctness is almost identical to the lower bound for sorting. As in Section 5, when we talk about permutations, the underlying set is $\{0, 1, \ldots, N-1\}$.

Here is a basic idea for creating two inputs that are hard to distinguish. Consider an input x that contains no collisions. Then we can create another input with a unique collision by replacing any of the elements in x with a *copy* of any other element in x. Though this describes a perfectly good procedure for creating hard input pairs, we will, however, describe a different and more complicated procedure for obtaining the same net result. The more complicated procedure consists of two stages. Consider an input x that contains no collisions. First we create a new input y that also contains no collisions, and then we create an input y' with a unique collision. Pick integers k and d so that $0 \leq k \leq N-2$ and $1 \leq d < N-k$. Form input y from x by replacing the element of rank $k+d$ in x with a *new* element of rank k. The element in x that has rank i then has rank $i+1$ in y, for all $k \leq i \leq k+d-1$. Now form input y' from y by replacing the element of rank k in y with a *copy* of the element of rank $k+1$ in y. Then y' contains a unique collision. Of course this procedure is essentially just a complicated version of the simple procedure described above, but it is helpful of thinking of the creation of an input with a unique collision in these terms in what follows.

For every permutation σ, and every integers $0 \leq k \leq N-2$ and $1 \leq d < N-k$, let the permutation $\tau = \sigma^{(k,d)}$ be defined as in Equation 13 in Section 5. For every permutation τ and every integer $0 \leq k \leq N-2$, let τ_k denote the input defined by

$$\tau_k(i) \;=\; \begin{cases} k+1 & \text{if } \tau(i) = k \\ \tau(i) & \text{otherwise.} \end{cases} \tag{20}$$

We refer to input τ_k as an *annotated permutation*. Write $\sigma_k^{(k,d)}$ as shorthand for $(\sigma^{(k,d)})_k$. For every permutation σ and every annotated permutation τ_k, define the value of the weight function on (σ, τ_k) in analogy with Equation 15,

$$\omega(\sigma, \tau_k) \;=\; \begin{cases} \frac{1}{d} & \text{if } \tau = \sigma^{(k,d)} \text{ for some } d \\ 0 & \text{otherwise.} \end{cases} \tag{21}$$

[1] Though $\gamma_i = \delta_i$ for all $1 \leq i < N$, we prefer to use both variables to increase readability by mimicking the proof of Lemma 7.

The comparison matrices M_σ and M_{τ_k} differ on all but one of the $2d$ entries given in Equation 14. They differ on the $2d-2$ entries given by

$$\left\{\sigma^{-1}(k+d), \sigma^{-1}(i)\right\} = \left\{\tau^{-1}(k), \tau^{-1}(i+1)\right\}$$

for $k < i \leq k+d-1$, and on the entry $\left(\sigma^{-1}(k), \sigma^{-1}(k+d)\right)$.

The total weight after j iterations is given by

$$W_j \;=\; \sum_{\sigma} \sum_{k=0}^{N-2} \sum_{d=1}^{N-k-1} \frac{1}{d} \langle \psi_\sigma^j | \psi_{\sigma_k^{(k,d)}}^j \rangle.$$

The initial weight is the same as in the case of sorting, $W_0 = N!(NH_N - N)$. Since any query can decrease the weighted all-pairs inner product by at most $2\pi N!\sqrt{N}$, Theorem 3 follows by applying Theorem 5.

Lemma 9 *For any integer j with $0 \leq j < T$, we have that $|W_j - W_{j+1}| \leq 2\pi N!\sqrt{N}$.*

Proof Almost identical to the proof of Lemma 8. In analogy with Equation 17, we have

$$|W_j - W_{j+1}| \leq 2 \sum_{d=1}^{N-1} \sum_{i=0}^{d-1} \frac{1}{d} \left(\sum_{\sigma} \sum_{k=0}^{N-d-1} \left| \langle \psi_\sigma^j | \mathsf{P}_{\sigma^{-1}(k+d), \sigma^{-1}(k+i)} | \psi_{\sigma_k^{(k,d)}}^j \rangle \right| \right). \tag{22}$$

Because the comparison matrices M_σ and M_{τ_k} differ only on $2d-1$ entries and we sum over all $2d$ entries as in the proof for sorting, we use a less-than-or-equal sign in Equation 22. This is a mere technicality and it is valid since we take absolute values on either side.

Similar to Equations 18 and 19, we apply the Cauchy–Schwarz inequality to upper bound the inner sum by

$$\sqrt{\sum_{\sigma} \sum_{k=0}^{N-d-1} \left\| \mathsf{P}_{\sigma^{-1}(k+d), \sigma^{-1}(k+i)} | \psi_\sigma^j \rangle \right\|^2} \sqrt{\sum_{\tau} \sum_{k=0}^{N-d-1} \left\| \mathsf{P}_{\tau^{-1}(k), \tau^{-1}(k+i+1)} | \psi_{\tau_k}^j \rangle \right\|^2}.$$

For every $1 \leq i < N$, let

$$\gamma_i = \sqrt{\sum_{\sigma} \sum_{l=0}^{N-1} \left\| \mathsf{P}_{\sigma^{-1}(l), \sigma^{-1}(l+i)} | \psi_\sigma^j \rangle \right\|^2}$$

$$\delta_i = \sqrt{\sum_{\tau} \sum_{l=0}^{N-1} \left\| \mathsf{P}_{\tau^{-1}(l), \tau^{-1}(l-i)} | \psi_{\tau_l}^j \rangle \right\|^2},$$

where we let l range from 0 to $N-1$, and define the vectors γ and δ as in the proof of Lemma 8. We then have that

$$|W_j - W_{j+1}| \;\leq\; 2 \sum_{d=1}^{N-1} \sum_{i=1}^{d} \frac{1}{d} \gamma_i \delta_{d-i+1} \;=\; 2\gamma K \delta,$$

where the matrix K is given by Equation 12. Besides having $\|\gamma\|^2 \leq N!$ as in the proof of Lemma 8, we also have that

$$\|\delta\|^2 = \sum_{i=1}^{N-1} \sum_{\tau} \sum_{l=0}^{N-1} \left\| \mathsf{P}_{\tau^{-1}(l),\tau^{-1}(l-i)} |\psi_{\tau_l}^j\rangle \right\|^2 \leq N!N,$$

and thus $|W_j - W_{j+1}| \leq 2\pi\sqrt{N!}\sqrt{N!N} = 2\pi N!\sqrt{N}$. □

7 A $\log_3(N)$ algorithm for ordered searching

We begin this section by considering binary search trees on which our quantum algorithm is based. A binary tree $\mathcal{T}$ is *complete* if the difference in depths of any two leaves is at most 1. A complete binary tree $\mathcal{T}$ is *right-complete* if the leaves with the largest depth lie in the rightmost positions. A binary tree is *perfect* if all leaves have the same depth. For the purpose of explaining our algorithm, we shall extensively refer to binary trees populated by pebbles. The following definitions are illustrated in Figure 1.

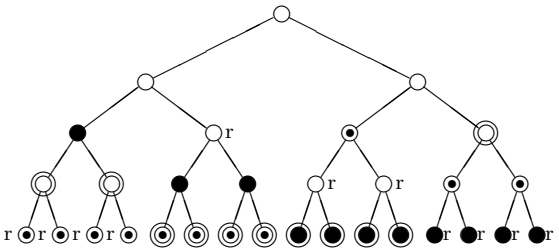

Figure 1: A covering of the perfect binary tree $\mathcal{T}$ with $N = 32$ leaves by $N' = 11$ pebbles. The leaves are not shown. We use 11 pebbles of each of 4 colors, yielding a total of 44 pebbles. The 11 black pebbles are placed on black-colored vertices, red pebbles on vertices marked with an 'r', white pebbles on vertices marked with an outer circle, and pink pebbles on vertices marked with an inner filled circle. Note that the placements of the pebbles of each color is obtained by shifting the placements of the black pebbles horizontally. The covering is *fair* since every color is used exactly 11 times. The covering is *tight* since on every vertex, either there is 1 pebble and none on its parent, or there are exactly as many pebbles as on its proper ancestors in total. The four vertices at depth 2 constitute the *boundary vertices*, since they are populated by pebbles while their parents are not.

Let $\mathcal{T}$ be a binary tree with $N \geq 2$ leaves. We put colored pebbles on the (internal) vertices of $\mathcal{T}$ subject to the 2 conditions:

(A) on every path from the root of $\mathcal{T}$ to a leaf, there is exactly 1 pebble of each color, and

(B) the number of pebbles p_v on any vertex $v \in \mathcal{T}$ is at least as large as the total number of pebbles on its proper ancestors.

We say that $\mathcal{T}$ is *covered by N' pebbles* if we can satisfy the 2 above conditions using at most N' pebbles of each color. We want to minimize the maximum number N' of pebbles used of any color.

We say a covering is *fair* if it uses the same number of pebbles of every color. We do not put any restrictions on the number of colors used. Note that if a tree can be fairly covered by N' pebbles using s colors, then it can also be fairly covered by N' pebbles using any multiple of s colors, simply by doubling the number of pebbles on each vertex. We say of a vertex $v \in \mathcal{T}$ for which $p_v > 0$ that it is *populated by pebbles*, or shortly, that it is populated. If a vertex $v \in \mathcal{T}$ is populated, but its parent is not or its parent is undefined, then we say that v is a *boundary vertex* and that it is located *on the boundary*. Vertices not on the boundary are said to be *non-boundary*.

We say a covering is *tight* if for every vertex $v \in \mathcal{T}$, either p_v equals the total number of pebbles on its proper ancestors, or v is on the boundary. Note that a covering is tight if whenever we walk down the tree from the root and record the number of pebbles on the vertices passed, we recognize the sequence $(0, \ldots, 0, 1, 1, 2, 4, 8, 16, \ldots)$ or an integral multiple of it. A tight covering satisfies that, for all leaves ℓ and all vertices v in $\mathcal{T}$,

$$p_v = \begin{cases} 2^{s+1}/2^{d(v,\ell)} & \text{if } v \text{ is boundary} \\ 2^s/2^{d(v,\ell)} & \text{if } v \text{ is populated and non-boundary,} \end{cases} \tag{23}$$

where 2^s is the number of colors used and $d(v, \ell)$ denotes the absolute value of the difference in depths of vertex v and leaf ℓ.

Consider a right-complete binary tree with N leaves. This tree can clearly be covered by $\lfloor N/2 + 1 \rfloor$ pebbles, for example, by putting one (say, black) pebble on each of the vertices adjacent to a leaf. We can, however, do better than this. Indeed, we can cover some binary tree with N leaves by only $N' = \lfloor N/3 + \log_2(N) \rfloor$ pebbles. A covering of the perfect binary tree with $N = 32$ leaves by $N' = 11$ pebbles is given in Figure 1. A straightforward generalization of the covering in this figure yields Lemma 10 and Theorem 11.

Lemma 10 *A perfect binary tree $\mathcal{T}$ with $N = 2 \cdot 4^n$ leaves can be fairly and tightly covered by $N' = \frac{1}{3}(N + 1)$ pebbles using 2^n colors.*

Proof See Figure 1 for an illustration of the case $N = 32 = 2 \cdot 4^2$. We use 2^n colors, $\{c_0, \ldots, c_{2^n-1}\}$. Label the 2^n subtrees of $\mathcal{T}$ rooted at vertices at depth n by $\{\mathcal{T}_0, \ldots, \mathcal{T}_{2^n-1}\}$. For each color c_i, put a pebble colored c_i on the root of $\mathcal{T}_i$. For each color c_i and for each $1 \le d \le n$, put a pebble colored c_i on each of the $2^d \cdot 2^{d-1}$ vertices at depth d in the 2^{d-1} subtrees $\mathcal{T}_{[i+2^{d-1}]}, \ldots, \mathcal{T}_{[i+2^d-1]}$, where $\mathcal{T}_{[k]}$ denotes the subtree $\mathcal{T}_l$ with $l = k \bmod 2^n$.

By construction, Condition (A) holds. Let $v \in \mathcal{T}$ be a vertex at depth d, and let p_v denote the number of pebbles on v. If $d < n$ then $p_v = 0$, if $d = n$ then $p_v = 1$, and if $n + 1 \le d \le 2n$ then $p_v = 2^{d-n-1}$. Thus, Condition (B) holds. We have used exactly $1 + \sum_{d=1}^{n} 2^d \cdot 2^{d-1} = \frac{1}{3}(N + 1)$ pebbles of each color. In conclusion, we have given a fair and tight covering of $\mathcal{T}$ by $N' = \frac{1}{3}(N + 1)$ pebbles. □

Theorem 11 *For every even $N \ge 2$, there exists a binary tree with N leaves that can be fairly and tightly covered by $N' = \lfloor \frac{1}{3}N + \log_2(N) \rfloor$ pebbles using 2^s colors, where $s = \lfloor \log_4(N/2) \rfloor$.*

Proof Write $N = \sum_{i=0}^{s} a_i 2 \cdot 4^i$, where $s = \lfloor \log_4(N/2) \rfloor$ and $0 \le a_i \le 3$ for $0 \le i \le s$. Let $\mathcal{T}'$ be any binary tree with $a = \sum_{i=0}^{s} a_i$ leaves. For each $0 \le i \le s$, substitute a_i of the original leaves of $\mathcal{T}'$ by perfect binary trees with $2 \cdot 4^i$ leaves. The resulting tree $\mathcal{T}$ has N leaves. Each of the substituted subtrees of $\mathcal{T}$ can be fairly and tightly covered using exactly 2^s colors, and thus so can $\mathcal{T}$.

We have shown that $\mathcal{T}$ can be covered by $N' = \sum_{i=0}^{s} a_i \frac{1}{3}(2 \cdot 4^i + 1)$ pebbles. Rewriting, we have $N' \le \frac{1}{3}N + (s+1) \le \frac{1}{3}N + (\log_4(N/2) + 1) \le \frac{1}{3}N + \log_2(N)$. The theorem follows by noting that we can only use an integral number of pebbles. □

We also require the following lemma when bounding the complexity of our algorithm given below.

Lemma 12 *Let integer-valued function $\tilde{F}$ be recursively defined by*

$$\tilde{F}(N) = \begin{cases} \tilde{F}\big(\lfloor \frac{1}{3}N + \log_2(N) + 1 \rfloor\big) + 1 & \text{if } N > 8 \\ 1 & \text{if } N \le 8. \end{cases}$$

Then $\tilde{F}(N) = \log_3(N) + O(1)$.

7.1 The algorithm

We now give our $\log_3(N) + O(1)$ quantum algorithm for ordered searching. The input $x = (x_0, \ldots, x_{N-1}) \in \{0,1\}^N$ is given as an oracle, and it satisfies that $x_{N-1} = 1$ and $x_{i-1} \le x_i$ for all $1 \le i < N$. The problem is to determine the leftmost 1 in x, that is, to compute $f(x) = \min\{0 \le i < N \mid x_i = 1\}$.

Without loss of generality, we assume that N is even. Let $\mathcal{T}$ be a binary tree with N leaves for which Theorem 11 holds. Let $s = \lfloor \log_4(N/2) \rfloor$ and $N' = \lfloor \frac{1}{3}N + \log_2(N) \rfloor$ be as in Theorem 11. We label the N leaves of $\mathcal{T}$ by $\{0, \ldots, N-1\}$ from left to right. Let $\ell_{f(x)}$ denote the leaf labeled by $f(x)$, and let $\mathcal{P}$ denote the path from the root of $\mathcal{T}$ to the parent of $\ell_{f(x)}$. We think of $\mathcal{P}$ as the path the classical search algorithm would traverse if searching for $f(x)$ in tree $\mathcal{T}$.

Let $\mathcal{C} = \{c_0, \ldots, c_{2^s-1}\}$ be the set of 2^s colors used in Theorem 11. For each color $c \in \mathcal{C}$, let V_c denote the set of vertices in $\mathcal{T}$ populated by a pebble of color c. By Condition (A), there are at most N' such vertices, that is, $|V_c| \le N'$. Let v_c denote the unique vertex in V_c that is on the path $\mathcal{P}$. We think of vertex v_c as the root of the subtree that "contains" the leaf $\ell_{f(x)}$. Though trivial, please note that $v_c \in \mathcal{P}$ for every color $c \in \mathcal{C}$, and that $\sum_{v \in \mathcal{P}} p_v = 2^s$.

In the following description, we use two registers. The first register is used to store information about our current position in the search tree. The set of basis vectors for the first register is the union of three sets. We use a basis vector for each vertex and leaf in $\mathcal{T}$, $\{|u\rangle \mid u \text{ is a vertex or leaf in } \mathcal{T}\}$, two additional basis vectors for each vertex on the boundary, $\{|v; b\rangle \mid v \text{ is boundary and } b \text{ is a bit}\}$, and also some dummy value $\{|0\rangle\}$. The second register holds information about a color. We use a basis vector for each color and some dummy value, $\{|c\rangle \mid c \in \mathcal{C}\} \cup \{|0\rangle\}$.

Our algorithm utilizes 3 unitary operators, U_1, O'_x, and U_2. The first operator, U_1, is defined by

$$\mathsf{U}_1 : \quad |v\rangle|0\rangle \;\longmapsto\; |v\rangle\left(\frac{1}{\sqrt{p_v}}\sum_c |c\rangle\right) \qquad (v \in \mathcal{T}), \tag{24}$$

where the summation is over all colors $c \in \mathcal{C}$ that are represented by a pebble on vertex v. We refer to U_1 as the *coloring operator* and its inverse as the *un-coloring operator*.

The query operator O'_x is defined by

$$\mathsf{O}'_x : \quad |v\rangle \;\longmapsto\; \begin{cases} |v\,;x_i\rangle & \text{if } v \text{ is boundary} \\ (-1)^{x_i}|v\rangle & \text{if } v \text{ is populated and non-boundary,} \end{cases} \tag{25}$$

where i denotes the label of the rightmost leaf in the left subtree of vertex v. Query operator O'_x is clearly unitary (or rather, can be extended to a unitary operator since it is only defined on a proper subspace). Operator O'_x is slightly different from, but equivalent to, the query operator defined in Section 2. It mimics the classical search algorithm by querying the bit x_i that corresponds to the rightmost leaf in the left subtree of v.

We also use a unitary operator U_2 that maps each vertex to a superposition over the leaves in its subtree. For every vertex and leaf u in $\mathcal{T}$, let $\mathcal{L}(u)$ denote the set of leaves in the subtree rooted at u, and let

$$|\Phi_u\rangle = \sum_{\ell \in \mathcal{L}(u)} \frac{1}{\sqrt{2^{d(u,\ell)}}}|\ell\rangle, \tag{26}$$

where $d(u,\ell)$ denotes the absolute value of the difference in depths of u and leaf ℓ. The unitary operator U_2 is (partially) defined as follows. For all boundary vertices $v \in \mathcal{T}$,

$$|v\,;0\rangle \;\longmapsto\; |\Phi_{\mathrm{right}(v)}\rangle \tag{27.1}$$

$$|v\,;1\rangle \;\longmapsto\; |\Phi_{\mathrm{left}(v)}\rangle, \tag{27.2}$$

and for all populated non-boundary vertices $v \in \mathcal{T}$,

$$|v\rangle \;\longmapsto\; \frac{1}{\sqrt{2}}\big(|\Phi_{\mathrm{right}(v)}\rangle - |\Phi_{\mathrm{left}(v)}\rangle\big). \tag{27.3}$$

Here $\mathrm{left}(v)$ denotes the left child of v, and $\mathrm{right}(v)$ the right child.

By Equations 25 and 27, for all populated vertices v on path $\mathcal{P}$, we have that

$$\langle \ell_{f(x)}|\mathsf{U}_2\mathsf{O}'_x|v\rangle \;=\; \begin{cases} 2^{-\frac{1}{2}[d(v,\ell_{f(x)})-1]} & \text{if } v \text{ is boundary} \\ 2^{-\frac{1}{2}[d(v,\ell_{f(x)})]} & \text{if } v \text{ is non-boundary,} \end{cases}$$

which, by Equation 23, implies that

$$\langle \ell_{f(x)}|\mathsf{U}_2\mathsf{O}'_x|v\rangle \;=\; \sqrt{p_v/2^s} \tag{28}$$

for all vertices v on path $\mathcal{P}$.

Our quantum algorithm starts in the initial state $|0\rangle|0\rangle$ and produces the final state $|\ell_{f(x)}\rangle|0\rangle$. Let $F(N)$ denote the number of queries used by the algorithm on an oracle x of size N.

1. We first set up a superposition over all 2^s colors,

$$\frac{1}{\sqrt{2^s}} \sum_{c \in \mathcal{C}} |0\rangle |c\rangle.$$

2. We then apply our exact quantum search algorithm recursively. For each color $c \in \mathcal{C}$ in quantum parallel, we search recursively among the vertices in V_c, hereby determining the root $v_c \in V_c$ of the subtree containing the leaf $\ell_{f(x)}$. Since $|V_c| \leq N'$, this requires at most $F(N'+1)$ queries to oracle x and produces the superposition

$$\frac{1}{\sqrt{2^s}} \sum_{c \in \mathcal{C}} |v_c\rangle |c\rangle.$$

   Since every vertex v_c in the above sum is on the path $\mathcal{P}$, we can rewrite the sum as

$$\frac{1}{\sqrt{2^s}} \sum_{v \in \mathcal{P}} |v\rangle \sum_{c \in \mathcal{C}: v_c = v} |c\rangle.$$

3. We then apply the un-coloring operator U_1^{-1}, producing the superposition $\frac{1}{\sqrt{2^s}} \sum_{v \in \mathcal{P}} \sqrt{p_v}\, |v\rangle |0\rangle$. Ignoring the second register which always holds a zero, this is

$$\frac{1}{\sqrt{2^s}} \sum_{v \in \mathcal{P}} \sqrt{p_v}\, |v\rangle.$$

   That is, we have (recursively) obtained a superposition over the vertices on the path $\mathcal{P}$ from the root of $\mathcal{T}$ to the parent of the leaf $\ell_{f(x)}$ labeled by $f(x)$.

4. We then apply the operator $\mathsf{U}_2\mathsf{O}'_x$, producing the final state

$$\mathsf{U}_2\mathsf{O}'_x \frac{1}{\sqrt{2^s}} \sum_{v \in \mathcal{P}} \sqrt{p_v}\, |v\rangle \;=\; \frac{1}{\sqrt{2^s}} \sum_{v \in \mathcal{P}} \sqrt{p_v}\, \mathsf{U}_2\mathsf{O}'_x\, |v\rangle.$$

   The amplitude of the state $|\ell_{f(x)}\rangle$ we are searching for, is

$$\frac{1}{\sqrt{2^s}} \sum_{v \in \mathcal{P}} \sqrt{p_v}\, \langle \ell_{f(x)} | \mathsf{U}_2\mathsf{O}'_x | v \rangle,$$

   which by Equation 28 is equal to

$$\frac{1}{\sqrt{2^s}} \sum_{v \in \mathcal{P}} \sqrt{p_v} \sqrt{\frac{p_v}{2^s}} \;=\; \frac{1}{2^s} \sum_{v \in \mathcal{P}} p_v \;=\; 1.$$

   Hence, the final state obtained after applying operator $\mathsf{U}_2\mathsf{O}'_x$ is precisely $|\ell_{f(x)}\rangle$.

The total number of queries to the oracle x is at most $F(N'+1)+1$, and thus, by Lemma 12, the algorithm uses at most $\log_3(N) + O(1)$ queries. Theorem 13 follows.

Theorem 13 *The above described quantum algorithm for searching an ordered list of N elements is exact and uses at most $\log_3(N) + O(1)$ queries.*

A few remarks on the operator U_2, as defined in Equation 27, are worthy mentioning. Firstly, it is possible to define generalizations of operator U_2 that can be applied to any rooted tree. Such a generalization might be of use in other search problems. Secondly, the operator U_2 is related to the Haar wavelet transform [13]. Applying operator U_2 is equivalent to applying the inverse of the Haar transform on each of the perfect subtrees rooted at the boundary vertices. Operator U_2 as applied in the fourth and final step of our algorithm can thus be implemented by applying the inverse of the quantum Haar transform. Since the Haar transform can be efficiently implemented [16, 20], so can U_2. The quantum version of the Haar transform was first considered and defined in [20], motivated by the successes of the quantum Fourier transforms. Possible relationships between the quantum Haar transform and ordered searching has previously been considered by Röhrig [22] and others.

8 Concluding remarks and open problems

The inner product of two quantum states is a measure for their distinguishability. For instance, two states can be distinguished with certainty if and only if their inner product is 0. In this paper, we have proposed a weighted all-pairs inner product argument as a tool for proving lower bounds in the quantum black box model. We have used this argument to give better and simpler lower bounds for quantum ordered searching, sorting, and element distinctness. It seems to us that the possibility of using non-uniform weights is particular suitable when proving lower bounds for non-symmetric (possibly partial) functions.

We have chosen here to use inner products which is only one of the many studied measures for distinguishability of states. A striking example of the limitations of using this measure is given by Jozsa and Schlienz in [21]. In [25], Zalka uses a non-linear measure to prove the optimality of Grover's algorithm [18]. Similarly, it might well be that utilizing some other (possibly non-linear) measure of distinguishability could be used to prove new, and improve old, lower bounds.

Even if only considering inner products, it is possible to improve our lower bounds (stated in Section 1) in terms of the error tolerance ϵ by generalizing Lemma 4 to more than two states. Suppose that we are given one of three states, $\{|\Phi_1\rangle, |\Phi_2\rangle, |\Phi_3\rangle\}$, and that there exists some measurement that correctly determines which of the three states we are given with error probability at most ϵ. Then the sum of the absolute values of the pairwise inner products, $|\langle\Phi_1|\Phi_2\rangle| + |\langle\Phi_1|\Phi_3\rangle| + |\langle\Phi_2|\Phi_3\rangle|$, can be at most three times $\sqrt{2}\sqrt{1-\epsilon}\sqrt{\epsilon} + \frac{\epsilon}{2}$. More generally, if given one of N states, the sum of the absolute values of the inner products is at most $\binom{N}{2}$ times $\frac{2}{\sqrt{N-1}}\sqrt{1-\epsilon}\sqrt{\epsilon} + \epsilon\frac{N-2}{N-1}$, which is in $\epsilon + o(1)$ for fixed ϵ. After the acceptance of publication of this paper, Ambainis [3] has informed us that a bound of $\epsilon + o(1)$ also holds for sums of weighted inner products. An anonymous referee has pointed out to us that in the lower bound of $\frac{1}{12}\log_2(N) + O(1)$ for ordered searching by Ambainis [2], the dependence of the error tolerance ϵ is only in the additive $O(1)$ term. This implies that if ϵ is at least 0.622 then $\frac{1}{12}\log_2(N) > (1-\epsilon)0.220\log_2(N)$, in which case the lower bound of $\frac{1}{12}\log_2(N) + O(1)$ is stronger.

The result of Grigoriev, Karpinski, Meyer auf der Heide, and Smolensky [17] implies that if only comparisons are allowed, the randomized decision tree complexity of Element Distinctness has the same $\Omega(N \log N)$ lower bound as sorting. Interestingly, their quantum complexities differ dramatically: the quantum algorithm for Element Distinctness by Buhrman *et al.* [10] uses only $O(N^{3/4} \log N)$ comparisons.

Space-time tradeoffs for sorting and related problems have been studied for the classical case. A *Time* $\cdot$ *Space* lower bound of $\Omega(N^2)$ is proved for comparison-based sorting by Borodin *et al.* [8], and for the R-way branching program by Beame [5]. Formulations and results on the quantum time-space tradeoffs for sorting and other problems such as Element Distinctness would be interesting.

Our algorithm for searching an ordered list with complexity $\log_3(N) + O(1)$ is based on the classical binary search algorithm. The quantum algorithm initiates several independent walks/searches at the root of the binary search tree. These searches traverse down the tree faster than classically by cooperating, and they eventually all reach the leaf we are searching for in roughly $\log_3(N)$ steps. We believe it is an interesting question whether similar ideas can be used to speed up other classical algorithms. We also think it would be interesting to consider other applications of operators like U_2 acting on rooted trees and graphs.

Wavelet transforms is a very rich and powerful area. We have here used only the most basic wavelet, the Haar transform. This transform might also be applicable in other problems where the input function is promised to be piecewise constant. For more smooth input functions, other wavelets, like Daubechies' D^4 wavelet transform [13], would probably do better. We believe it is a very interesting question to study the applicabilities of other bases than the Fourier bases for quantum computation.

Acknowledgments

We are grateful to Andris Ambainis, Harry Buhrman, Mark Ettinger, Gudmund S. Frandsen, Dieter van Melkebeek, Hein Röhrig, Daniel Wang, Ronald de Wolf, Andy Yao, and especially Sanjeev Arora, for their precious comments and suggestions. We thank the anonymous referees for helpful comments.

References

[1] A. Ambainis, A better lower bound for quantum algorithms searching an ordered list, *Proc. of 40th Ann. IEEE Symp. on Foundations of Computer Science*, 1999, pp. 352–357.

[2] A. Ambainis, Quantum lower bounds by quantum arguments, *Proc. of 32nd Ann. ACM Symp. on Theory of Computing*, 2000, pp. 636–643.

[3] A. Ambainis, Personal communication, July 2001.

[4] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf, Quantum lower bounds by polynomials, *Proc. of 39th Ann. IEEE Symp. on Foundations of Computer Science*, 1998, pp. 352–361.

[5] P. Beame, A general sequential time-space tradeoff for finding unique elements, *SIAM J. Comput.*, **20**(1991) 270–277.

[6] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, Strengths and weaknesses of quantum computation, *SIAM J. Comput.*, **26**(1997) 1510–1523.

[7] A. Berthiaume, Quantum computation, in *Complexity Theory Retrospective II* (L. Hemaspaandra and A. L. Selman, eds.), chap. 2, Springer-Verlag, 1997, pp. 23–50.

[8] A. Borodin, M. J. Fischer, D. G. Kirkpatrick, N A. Lynch, and M. Tompa, A time-space tradeoff for sorting on nonoblivious machines, *J. Comput. Sys. Sci.*, **22**(1981) 351–364.

[9] G. Brassard, P. Høyer, M. Mosca, and A. Tapp, Quantum amplitude amplification and estimation, in *Quantum Computation and Quantum Information: A Millennium Volume* (Samuel J. Lomonaco, Jr., ed.), AMS Contemp. Math. Series, 2001. To appear.

[10] H. Buhrman, C. Dürr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, and R. de Wolf, Quantum algorithms for finding claws, collisions and triangles. *Proc. of 16th IEEE Conf. on Computational Complexity*, 2001, pp. 131–137.

[11] H. Buhrman and R. de Wolf, A lower bound for quantum search of an ordered list, *Inform. Proc. Lett.*, **70**(1999) 205–209.

[12] M.-D. Choi, Tricks or treats with the Hilbert matrix, *Amer. Math. Monthly*, **90**(1983) 301–312.

[13] I. Daubechies, Orthonormal bases of compactly supported wavelets, *Comm. Pure Appl. Math.*, **XLI**(1988) 909–996.

[14] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, A limit on the speed of quantum computation for insertion into an ordered list. quant-ph/9812057, December 1998.

[15] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, Invariant quantum algorithms for insertion into an ordered list. quant-ph/9901059, January 1999.

[16] A. Fijany and C. P. Williams, Quantum wavelet transforms: Fast algorithms and complete circuits. quant-ph/9809004, September 1998.

[17] D. Grigoriev, M. Karpinski, F. Meyer auf der Heide, and R. Smolensky, A lower bound for randomized algebraic decision trees, *Comput. Complexity*, **6**(1996/1997) 357–375.

[18] L. K. Grover, Quantum mechanics helps in searching for a needle in a haystack, *Phys. Rev. Lett.*, **79**(1997) 325–328.

[19] A. Haar, Zur theorie der orthogonalen funktionensysteme (Erste mitteilung), *Math. Ann.*, **LXIX**(1910) 331–371.

[20] P. Høyer, Efficient quantum algorithms. quant-ph/9702028, February 1997.

[21] R. Jozsa and J. Schlienz, Distinguishability of states and von Neumann entropy, *Phys. Rev. A*, **62**(2000) 012301.

[22] H. Röhrig, Personal communication, 2000–2001.

[23] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM J. Comput.*, **26**(1997) 1484–1509.

[24] U. Vazirani, On the power of quantum computation, *Philos. Trans. Roy. Soc. London Ser. A*, **356**(1998) 1759–1768.

[25] Ch. Zalka, Grover's quantum searching algorithm is optimal, *Phys. Rev. A*, **60**(1999) 2746–2751.

Many of the above references can be found at the Los Alamos National Laboratory e-print archive (http://arXiv.org/archive/quant-ph).