Internship Report
Quantum computing and query complexity

Quoc Tung Le*

Computer Science Department, École normale supérieure

August 24, 2018

Abstract

Quantum computer is likely to be the next computer generation, which possesses a greater computing power comparing to the current classical one. Therefore, in this report, I will present my work on this field. To be more precise, my report will demonstrate the power of quantum computing by investigating 2 problems, namely the discrete version of Convex Optimization Problems for functions of one variable and the Ordered Search Problem [1] in 2 dimensions. Both problems will be analyzed in 2 different settings: classical and quantum in term of algorithms and lower bounds. This allows to show the superiority of quantum computing over classical one.

1 The introduction over the internship

The idea of quantum computing is to use the laws of physics to carry out the computation more efficiently. Many results in quantum computing are interesting and striking such as the Grover's algorithm [4] or Shor's algorithm [5]. Those studies exert a huge impact in many fields such as communication and cryptography.

In the internship, the main objective is to study the concept of quantum computing and to apply such knowledge to the convex optimization problems [6], the main interest of the internship. Nevertheless, the report also introduces some classical algorithms as well as the classical complexity lower bounds of the same problem. This gives us the comparison between 2 settings: quantum and classic and shows that quantum computing can obtain something which could not be achieved by classical one.

In the rest of the report, there are 2 main problems which will be discussed. The first problem is *Discrete minimum of a convex function in* 1 *dimension*. In this problem, my contribution is:

1. Devise an algorithm $O(M)$ for classical setting, which appear to surpass classical binary search in certain cases.
2. Prove a lower bound for the classical algorithm under certain condition.
3. Using a quantum algorithm for Ordered Search Problem to achieve a constant speed-up (which is proved not possible for any classical algorithm)

The second problem is the *Ordered Search Problem in* 2 *dimension*. My main contribution in this problem is:

1. Establish the lower bound for the problem in classical setting.
2. Devise an algorithm reaches the lower bound, and possibly extend the idea for d dimension.
3. A quantum algorithm that surpasses the classical lower bound.
4. A lower bound for quantum algorithms.

*quoctung.le@ens.fr

2 A brief overview on quantum computing notations and query model

2.1 Quantum bits

Similar to bit in classical setting, quantum bit (qubit) is the smallest unit of information in quantum setting. They also have two basic states which are "$|0\rangle$" and "$|1\rangle$" corresponding to two states 0 and 1 in classical setting. The notations "$|\rangle$" (and also "$\langle|$") are deployed to distinguish qubits from classical bits.

The main difference between qubits and classical bits is that qubits can possess both states (also known as *"superposition"*) in one qubit. In general, a qubit can be expressed in the following formulation:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

where α and β are complex numbers and satisfy $|\alpha|^2 + |\beta|^2 = 1$. Two most famous states are:

$$|+\rangle = \tfrac{1}{\sqrt{2}}\,|0\rangle + \tfrac{1}{\sqrt{2}}\,|1\rangle,\ |-\rangle = \tfrac{1}{\sqrt{2}}\,|0\rangle - \tfrac{1}{\sqrt{2}}\,|1\rangle$$

2.2 Multiple Qubits

For the discussion of multiple qubits, we will only look into the simplest form: the 2 qubits system since the extension of d qubits is rather natural. Similarly to the classical setting, they have 4 basic states, namely $(|00\rangle, |01\rangle, |10\rangle, |11\rangle)$. The general representation of a 2 qubits system is as follow:

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle$$

where $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$.

2.3 Mathematics with Qubits

A quantum state in a qubit system can be represented as a unit vector in $\mathbb{C}^2$ plane, spanned by the following 2 basis state:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}, |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

Therefore, the qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix}\alpha\\\beta\end{bmatrix}$.

A quantum state $|\psi\rangle$ is a (column) vector, also known as a **ket**, whereas a state $\langle\psi|$ is the (row) vector dual to $|\psi\rangle$, also known as a **bra**. In the $\mathbb{C}^2$ vector space, the **bra** is the *conjugate transpose* of the corresponding **ket**.

$$\langle\psi| = (|\psi\rangle)^\dagger = \begin{bmatrix}\alpha\\\beta\end{bmatrix}^\dagger = \begin{bmatrix}\alpha^\dagger & \beta^\dagger\end{bmatrix}$$

Given 2 qubits: $|\psi_0\rangle = \alpha_0\,|0\rangle + \beta_0\,|1\rangle$ and $|\psi_1\rangle = \alpha_1\,|0\rangle + \beta_1\,|1\rangle$

Definition 2.1. The *inner product* of $|\psi_0\rangle$ and $|\psi_1\rangle$ is notated as $\langle\psi_0|\,.\,|\psi_1\rangle$ or simply $\langle\psi_0|\psi_1\rangle$ and defined as:

$$\langle\psi_0|\,.\,|\psi_1\rangle = \begin{bmatrix}{\alpha_0}^\dagger & {\beta_0}^\dagger\end{bmatrix}\begin{bmatrix}\alpha_1\\\beta_1\end{bmatrix} = {\alpha_0}^\dagger\alpha_1 + {\beta_0}^\dagger\beta_1$$

Definition 2.2. The *outer product* of $|\psi_0\rangle$ and $|\psi_1\rangle$ is notated as $|\psi_0\rangle\,.\,\langle\psi_1|$ and defined as:

$$|\psi_0\rangle\,.\,\langle\psi_1| = \begin{bmatrix}\alpha_0\\\beta_0\end{bmatrix}\begin{bmatrix}{\alpha_1}^\dagger & {\beta_1}^\dagger\end{bmatrix} = \begin{bmatrix}\alpha_0{\alpha_1}^\dagger & \alpha_0{\beta_1}^\dagger\\\beta_0{\alpha_1}^\dagger & \beta_0{\beta_1}^\dagger\end{bmatrix}$$

Now, we get to handle the system with more than 2 qubits $|x\rangle = \alpha_0\,|0\rangle + \beta_0\,|1\rangle$ and $|y\rangle = \alpha_1\,|0\rangle + \beta_1\,|1\rangle$. How can we describe their joint state? The answer is to use tensor product.

$$
\begin{aligned}
|x\rangle \otimes |y\rangle &= (\alpha_0 |0\rangle + \beta_0 |1\rangle) \otimes (\alpha_1 |0\rangle + \beta_1 |1\rangle) \\
&= \alpha_0\alpha_1 |0\rangle \otimes |0\rangle + \beta_0\alpha_1 |1\rangle \otimes |0\rangle + \alpha_0\beta_1 |0\rangle \otimes |1\rangle + \beta_0\beta_1 |1\rangle \otimes |1\rangle \\
&= \alpha_0\alpha_1 |00\rangle + \beta_0\alpha_1 |10\rangle + \alpha_0\beta_1 |01\rangle + \beta_0\beta_1 |11\rangle
\end{aligned} \tag{1}
$$

It is worth knowing that there exists some systems of 2 qubits which cannot be described as the tensor product of 2 qubit. Those states demonstrate one property of quantum computing - *entanglement*. The most famous of such system is $|\psi\rangle = \frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle$.

2.4 Quantum computation

In this section, we will discuss on how quantum circuits work in practice. It is equivalent to understand how the state of the whole quantum circuit changes under the effects of circuit gates. In fact, in order for the circuit gate U to be physically realizable, we have two restrictions:

1. $U : \mathbb{C}^d \to \mathbb{C}^d$ has to map from a quantum state to a quantum state
2. U has to be linear (i.e $U(|x\rangle + |y\rangle) = U|x\rangle + U|y\rangle$).

To satisfy those 2 restrictions, $U \in \mathbb{C}^{d\times d}$ is **unitary**. In other words, U is a matrix satisfying:

$$U^\dagger U = I$$

Remark 2.1. Until now, we only discuss the unitary operator (quantum circuit) for 1 qubit only. What happen if we apply 2 circuits U and V (or more circuits generally) to $|x\rangle$ and $|y\rangle$ in 2-qubit system? What is the state of the whole system after applying such circuit? We have:

$$(U|x\rangle) \otimes (V|y\rangle) = (U \otimes V)(|x\rangle \otimes |y\rangle)$$

Thus, the answer is the effect is equivalent to a unitary operator $U \otimes V$.

2.5 The quantum query model

In query model, we are given a black-box function f and have to answer a question based on the its returns after a couple of accesses. The query complexity, naturally, is simply the number of queries one will pose to the black-box function to solve the problem. While it appears to be relatively simple, the query model offers a lot of advantages:

1. It simplifies the analysis of the algorithm.
2. In query model, we are able to prove nontrivial lower bounds which could not have achieved by classical complexity theory.
3. Usually, the time complexity is dominated by the query complexity. This results directly from the fact that the black box function f is usually easy to implement and the part outside the black-box function is also efficiently carried out.

Compare to the classical setting where the oracle simply takes a string of bits x as input and return the output y, the quantum setting allows the oracle to take a superposition as input and return another superposition as output (since the oracle itself is a quantum transformation). This could be utilized to gain the advantage by the quantum computing.

In the 2 following sections, there are 2 main problems which will be discussed. Both of them are analyzed in query model. In both problem, we are interested in the lower bounds and also the algorithm which could possibly reach the lower bounds. The complexity, without specification, will be understood as query complexity.

3 Discrete minimum of a convex function in 1 dimension

3.1 Definition and query model

The problem itself is the discrete version of the problem of minimizing a continuous one-variable convex M-Lipschitz function $f(x)$. In the continuous version, the usual question is to find the value of x such that $|f(x) - \min f(x)| \leq \epsilon, \epsilon > 0$ [8]. Another question which is also frequently asked is to find the value of x such that $|x - argmin f(x)| \leq \epsilon$ with $\epsilon > 0$. In our case, we are interested in the discrete version of the latter. We formalize the concept of one-variable convex function into the convex array.

Definition 3.1. An array A of n integers is convex M-Lispchitz iff the sequence $S[i] = A[i+1] - A[i], 1 \leq i < n$ is non-decreasing and $|S[i]| \leq M, M > 0, 1 \leq i < n$.

Below is the formal definition of the problem of convex array minimization.

DISCRETE MINIMUM OF A CONVEX FUNCTION IN 1 DIMENSION
Input: given a convex M-Lipschitz array A of n integers.
Query model:

1. Classical setting: Given x, the oracle O will return $A[x]$ and $S[x] = A[x+1] - A[x], 1 \leq x < n$. For $x = n$, we can assume that $S[n] = M$ to avoid technical difficulties.
2. Quantum setting: Given $|x\rangle |a_0\rangle |a_1\rangle$, $O |x\rangle |a_0\rangle |a_1\rangle = |x\rangle |a_0 \oplus S[x]\rangle |a_1 \oplus A[x]\rangle$.

Output: Finding i which minimizes the value $A[i]$.

From the definition of the array A, it is obvious that $|S[i]|$ is an non-decreasing bounded sequence of integers. Here, the sequence $S[i]$ which is non-decreasing and bounded is to mimic the behavior of the convex M-Lipschitz continuous function f.

In the first problem, the main result of classical setting is summarized in the following theorem:

Theorem 3.1. *In classical setting of the problem of finding the minimum of a convex M-Lipschitz array A of n integers.*

1. *The upper bound of query complexity is* $\min(4M, \log n)$.
2. *With M sufficiently large, the lower bound of query complexity is* $\log n - c$ *where $c > 0$ is a constant (that will be determined later).*

The following subsections will give the proof for the theorem 3.1.

3.2 An $O(M)$ and $\log n$ algorithm in classical settings

Before jumping to the $O(M)$ algorithm, we will discuss a relatively simpler $\log n$ algorithm which is based on binary search.

The algorithm is based on the simple observation: since the sequence S is increasingly sorted and we want to find the first index i of S which make $S[i]$ positive (also the minimum point), a binary search is sufficient to solve the problem. The number of queries which have to be used in the worst case is $\lceil \log n \rceil$.

The algorithm 1 is trivially true. However, the only information used from oracle is only $S[m]$ while $A[m]$ is left untouched. Therefore, one might consider the value of $A[m]$ to achieve a better performance. Indeed, the second algorithm utilized such information to improve the algorithm in a certain case (when M is a constant or relatively small comparing to n).

The algorithm is different with the previous $\log n$ algorithm by the way of choosing m. The following lemma will prove the correctness as well as analyze the complexity of the algorithm.

Lemma 3.2. *During the execution of the algorithm 2, these 2 statements will hold:*

1. *The point that minimizes the array is always between f and l (including f and l).*

Algorithm 1 An $O(\log n)$ binary search based algorithm

```
function MINIMIZE(n)
    f = 1, l = n
    while f < l do
        m = (f+l)/2
        Query the oracle at the position m
        if S[m] > 0 then
            l = m
        else
            f = m
    return l
```

Algorithm 2 An $O(M)$ algorithm

```
function MINIMIZE(n)
    f = 1, l = n
    Query the oracle at position f and l to get S[l], S[f], A[l], A[f]
    while S[f] ≠ S[l] do
        m_0 = ⌊(A[f] − f S[f] − A[l] + l S[l]) / (S[l] − S[f])⌋
        m_1 = ⌈(A[f] − f S[f] − A[l] + l S[l]) / (S[l] − S[f])⌉
        Query the oracle at the position m_0, m_1
        if S[m_i] = 0, i ∈ 0, 1 then return m_i
        if S[m_0]S[m_1] < 0 then return m_1
        if S[m_0] > 0 then
            l = m_0, update S[l], A[l] correspondingly
        else
            f = m_1, update S[f], A[f] correspondingly
    if S[l] > 0 then return f
    return l
```

2. *Denote* $T = \{S[i] | f \leq i \leq l\}$. *After each while loop, either the algorithm returns the true answer or* $|T|$ *decreases at least by* 1.

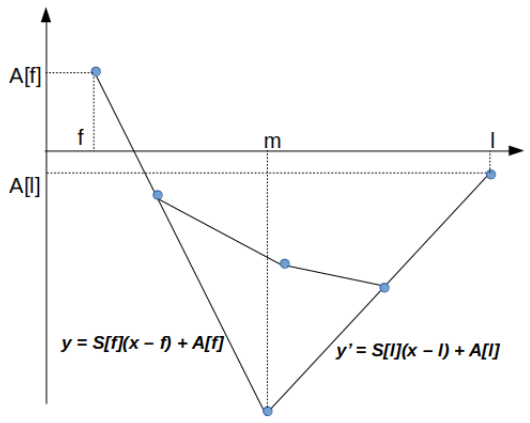


Figure 1: Illustration of m choice in algorithm 2. In fact, $m = \frac{A[f] - fS[f] - A[l] + lS[l]}{S[l] - S[f]}$ is the intersection of two lines y and y'. $m_0 = \lfloor m \rfloor, m_1 = \lceil m \rceil$

Proof. We prove the statements by inductions:

1. *Before entering While loops:* Since $f = 1, l = n$, it is trivial that the first statement is true
2. Suppose that the result is true until the end of the k^{th} iteration. We will prove the correctness in the $(k+1)^{th}$ iteration. There are several cases that need to be taken care of.

(a) If either $S[m_0]$ or $S[m_1]$ is equal to 0 then it is clear that m_0 or m_1 is the minimum point of the array due to the property of convex array. The algorithm will return correct answer.

(b) If $S[m_0]S[m_1] < 0$, then $S[m_0] < 0 < S[m_1]$. Again, by the property of convex array, $m_1 = m_0 + 1$ is the minimum point. The algorithm will return correct answer.

(c) If $S[m_0] < 0$ and $S[m_0]S[m_1] > 0$. First, we check that $m_1 \leq l$. Indeed,

$$\begin{aligned} \frac{A[f] - A[l] - fS[f] + lS[l]}{S[l] - S[f]} &= \frac{-(\sum_{i=f}^{l-1} S[i]) - fS[f] + lS[l]}{S[l] - S[f]} \\ &\leq \frac{-(l-f)S[f] - fS[f] + lS[l]}{S[l] - S[f]} = l \end{aligned} \tag{2}$$

With the fact that both m_1 and l are integers, it is clear that $m_1 \leq l$
Moreover, $S[m_1] > S[f]$. Indeed, if it is not the case, $S[m_1] = S[f]$, which leads to:

$$\begin{aligned} A[l] - A[f] = \sum_{i=f}^{l-1} S[i] &= \sum_{i=f}^{m_1} S[i] + \sum_{i=m_1+1}^{l-1} S[i] \\ &\leq (m_1 - f + 1)S[f] + (l - m_1 - 1)S[l] \\ &< (\frac{A[f] - A[l] - fS[f] + lS[l]}{S[l] - S[f]} - f)S[f] + (l - \frac{A[f] - A[l] - fS[f] + lS[l]}{S[l] - S[f]})S[l] \\ &= A[l] - A[f] \end{aligned} \tag{3}$$

Thus, $S[m_1] > S[f]$. The new interval will have $|T|$ which decreases by at least 1. In addition, since $S[m_0], S[m_1] < 0$ and the induction hypothesis, the minimum point is still between the new interval $[m_1, l]$.

(d) The case where $S[m_0] > 0$ and $S[m_0]S[m_1] > 0$ could be dealt by similar manner.

□

By the lemma 3.2, the algorithm 2 will eventually output the correct answer. Moreover, there cannot be more than $2M + 1$ iterations (since $|S[i]| \leq M$). The query complexity is thus $O(M)$.

3.3 Lower bounds in classical setting in certain conditions

In previous subsection, we already discussed 2 algorithms, with $\log n$ and $O(M)$ complexity respectively. As already remarked, the algorithm 1 of $\log n$ complexity only used a small piece of information from the oracle - the sign of $S[x]$. This might make us think that we can improve such algorithm. Nevertheless, in this subsection, we will prove the opposite: If M is large enough, then one cannot perform better than $\log n - c$ (c is a constant) queries. The technique used in this proof could easily adapt to the case where $S[i]$ is now real or rational number without considering the value of M.

The main idea is for any given deterministic algorithm V, we will prove that V cannot solve all instances of size $L = 2^N - 7$ with less than $N - 4$ queries. This could be achieved by fabricating an instance array A which have the following property ($*$): $\forall 0 \leq i \leq N - 4$, there exists a continuous segments s_i of length $l_i = 2^{N-i} - 7$ of A satisfies:

1. $s_i \subset s_{i-1}$
2. All the points which are queried after first i iterations of the algorithm V are not inside s_i (inclusively)
3. The minimum point of A is inside s_i (inclusively)

4. Denote f and l are two endpoints of s_i. Then $A[f] = A[l], S[f] = -S[l] < 0$.

If we have such an array A, then it is obvious that after first $N-4$ queries, the algorithm still fails to deliver the answer because the minimum points are in the segments s_{N-4} of size 11 which we still do not have any information. So what is left is to determined how to create such an A.

Lemma 3.3. *An array A satisfying $(*)$ exists.*

Proof. Proof can be found in the Appendix. □

3.4 Ordered Search Problem and a constant factor speedup in quantum settings

Up until now, we show that there are an $O(M)$ and $O(\log n)$ algorithm. More interestingly, algorithm 1 used exactly $\lceil \log n \rceil$ queries. We also have the lower bound of the problem to be $\log n - c$ in a certain condition. Therefore, in classical computing, there is almost no room for improvement. Nevertheless, in quantum setting, there are still some ways which allow to have a constant factor speedup regardless of the condition. This is a direct result of the work of E. Farhi et al. [1] of the Ordered Search Problem, which will be introduced shortly.

ORDERED SEARCH PROBLEM (OSP)
Input: given an array A of n elements.
Query model:

1. Classical setting: Given x, the oracle $O_j, 0 \leq n < n$ will return b such that:

$$b = \begin{cases} -1 & \text{if } x < j \\ 1 & \text{if } x \geq j \end{cases}$$

2. Quantum setting: Given $|x\rangle |a\rangle$, the oracle O_j, $O_j |x\rangle |a\rangle = |x\rangle |a \oplus f_j(x)\rangle$ where $f_j(x)$ satisfies:

$$f_j(x) = \begin{cases} 1 & \text{if } x < j \\ 0 & \text{if } x \geq j \end{cases}$$

Output: Finding j

In [1], the algorithm firstly doubles the array size and define a new equivalent oracle, which is: $F_j |x\rangle = F_j(x) |x\rangle$ where $F_j(x)$satisfies:

$$F_j(x) = \begin{cases} (-1)^{f_j(x)} & \text{if } x < n \\ -(-1)^{f_j(x)} & \text{if } n \leq x < 2n \end{cases} \tag{4}$$

This oracle allows us to exploit the symmetry of the problem and have a couple of nice properties which simplified the analysis. Nevertheless, we ignore the detail and focus to the general idea. With the new oracle, the problem switches to find a set of unitary operators $U_1, ..., U_k$ which could satisfy:

$$U_k F_j ... U_1 F_j (H^{\otimes(\log n+1)}) |0\rangle = \tfrac{1}{\sqrt{2}} |j\rangle \pm \tfrac{1}{\sqrt{2}} |j+n\rangle$$

where $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ is the Hadamard gate.

Then measuring the final state and taking the modulo n will return the answer.

In the article, the author successfully found U_1, U_2, U_3 to solve every instance array of size 52 by 3 queries. By using divide and conquer argument, one can solve the general problem in $3\lceil \log_{52} n \rceil \approx 0.526 \log n$. This result is further studied and improved to an $4\lceil \log_{605} n \rceil \approx 0.433 \log n$ algorithm in [2]. Those quantum algorithms enable us to have a quantum algorithm to minimize the convex M-Lipschitz array with better performance than classical lower bound.

Proposition 1. *The quantum query complexity of the discrete minimum of convex function in* 1 *dimension is at most* $0.433 \log n$

Proof. It is sufficient to propose a quantum algorithm which could solve the problem in $0.433 \log n$ queries. We will use the algorithm for OSP to achieve such complexity.

Indeed, if we only consider the bit b which determines the sign of $S[i]$. Suppose $S[i] < 0$ then $b = 1$, otherwise $b = 0$. On the other hand, the minimum of the convex array is identical to the first index j which makes the oracle return $b = 0$. Thus, by using the algorithms which are mentioned in [1] and [2], we can speed up the algorithm by constant factor $(0.433 \log n)$, something that is not possible in classical computing. □

4 Ordered Search Problem in 2 Dimension

4.1 Definition and query model

This problem is an attempt for the more general problem: minimizing a convex multiple variables function. This problem appears frequently in machine learning and takes the form:

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{m} f_i(x) + \lambda R(x) \tag{5}$$

where f_i, R are convex functions (f_i are usually the cost of using x in some data set and $R(x)$ is the regularization term). The classical setting is studied extensively in [9]. The study of such problem in quantum setting might enable a faster method for many machine learning optimization problems.

In this report, we deal with the general problem with some restrictions. Firstly, the function has only 2 variables. Secondly, we deal with the discrete version instead of the continuous one. It is worth noticing that minimizing a convex one-variable function is reducible to the Ordered Search Problem. Naturally, the same problem for 2-variable function could be also reducible to the Ordered Search Problem, but in 2 dimensions. Nevertheless, we will try to formalize a more general version with d dimension.

The problem of Ordered Search in d dimensions is formalized as follow:

ORDERED SEARCH PROBLEM (OSP) d-D
Input: given an d-D array A of n^d elements.
Query model:

1. Classical setting: Given $(x_1, x_2, ..., x_d)$, the oracle $O_{j_1, j_2, ..., j_d}, 0 < j_1, j_2, ..., j_d \leq n$ will return the pair (i, b) such that:
   (a) $i = \underset{i=0,1,...,d-1}{argmax} |x_i - j_i|$.
   (b)
   $$b = \begin{cases} -1 & \text{if } x_i < j_i \\ 1 & \text{if } x_i \geq j_i \end{cases}$$
2. Quantum setting: Given $|x_0\rangle |x_1\rangle ... |x_{d-1}\rangle |a_0\rangle |a_1\rangle$, the oracle $O_{j_0, j_1, ..., j_d}, 0 \leq j_0, j_1, ..., j_d < n$ will return the pair $|x_0\rangle |x_1\rangle ... |x_{d-1}\rangle |a_0 \oplus i\rangle |a_1 \oplus b\rangle$ such that.
   (a) $i = \underset{i=0,1,...,d-1}{argmax} |x_i - j_i|$.
   (b)
   $$b = \begin{cases} 1 & \text{if } x_i < j_i \\ 0 & \text{if } x_i \geq j_i \end{cases}$$

Output: Finding $(j_1, j_2, ..., j_d)$

Our interested problem is slightly different from the natural extension of OSP, which is quite easy since the simple binary search could reach the lower bound $\log n$. In this variant, the oracle returns $i = \underset{i=0,1,\ldots,d-1}{argmax} |x_i - j_i|$ and the corresponding relative position of the i^{th} element. It is motivated from the problem of finding the minimum of function $f(x) = \|x - j\|_\infty$ while having queries to the function value and gradient.

In the following section, we firstly establish the lower bounds of this problem for the general d dimension in classical case.

4.2 Lower bound in classical cases

In this part, we will prove the following lemma:

Lemma 4.1. *To solve the OSP d dimension, an algorithm needs to have at least $d \log_{2d} n$ queries in the worst case.*

Proof. Suppose V be an algorithm which would be able to the problem OSP of d dimensions. By adapting the concepts of query, V can be seen abstractly in term of decision tree. There are several conditions on this tree.

1. Every point in the searching space $n \times n \times .... \times n$ has to be represented as a leaf of the tree for the algorithm to work properly.
2. Each inner node has at most $2d$ children since the answer for one query has $2d$ different possibilities.

Denote h as the decision tree corresponding to the algorithm V. h is also the number of queries in the worst case. Since the number of leaves of the decision tree is at most $(2d)^h$, we have:

$$(2d)^h \geq n^d \implies h \geq d \log_{2d} n \tag{6}$$

□

4.3 An algorithm which could reach lower bound for OSP 2D

We will shortly introduce an algorithm which could reach the newly established lower bound. It is worth noticing that the normal binary search cannot reach the lower bound. Indeed, for $d = 2$, the lower bound is $2 \log_4 n = \log n$ while the complexity for normal binary search is $2 \log n$. In general d dimension problem, normal binary search has the complexity of $d \log n$.

Naturally, the queries allow us to reduce the searching space, thus get closer to the answer after each query. With this idea in mind, we will investigate how the searching space is reduced by posing one query.

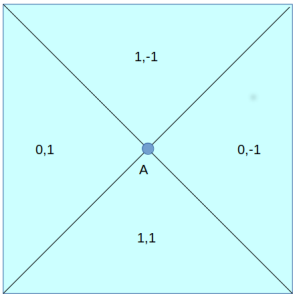


Figure 2: The division of the searching space caused by a query and pairs of (i, b) returned by the oracle

Illustrated by the Figure 2, the searching space is divided into 4 parts. Considering the initial searching space is a square of size $n \times n$, if we could consistently reduce the searching space by

4 times (corresponding to 4 partitions), we can eventually achieve a complexity of $log_4 n^2 = \log n$ queries, which is also the lower bound. That is also the main idea of the following algorithm. The following pseudo codes are only for one case where the first query return $(0, -1)$, but it could easily extend for other cases thank to the symmetry as well.

Algorithm 3 SQUARE

function SQUARE(x, y, t)
  if $t = 1$ **then**
    Query all 4 *points and return the answer*
  $x_1 = x - \frac{t}{2}$
  $y_1 = y - \frac{t}{2}$
  $i, b = \text{query}(x_1, y_1)$
  if $i = 0, b = -1$ **then**
    return Triangle(x, y, t)
  else
    The three remaining cases could be handled similarly by symmetry.

Algorithm 4 RHOMBUS

function RHOMBUS(x, y, t)
  if t $= 2$ **then**
    Query all 5 *points in the searching space and return the answer*
  $x_1 = x$
  $y_1 = y - \frac{t}{2}$
  $i, b = \text{query}(x_1, y_1)$
  if $i = 0, b = -1$ **then**
    return Rhombus$(x + \frac{t}{4}, y - \frac{t}{4}, \frac{t}{2})$
  if $i = 0, b = 1$ **then**
    return Rhombus$(x - \frac{t}{4}, y - \frac{t}{4}, \frac{t}{2})$
  if $i = 1, b = -1$ **then**
    return Rhombus$(x, y, \frac{t}{2})$
  if $i = 1, b = 1$ **then**
    return Rhombus$(x, y - \frac{t}{2}, \frac{t}{2})$

All of three procedures are illustrated in the Figure 3. Informally, the procedures *Square, Rhombus* and *Triangle* are used when the shape of the searching space is the square, rhombus and triangle respectively. (x, y) is the peak while t is the longest size of each shape. To solve the original problem, it is sufficient to call procedure Square(n, n, n) (Here, we use the first index of array as $(0, 0)$, for the sake of simple analysis later).

While the correctness is trivial, we concentrate to analyze the complexity. In the execution of the algorithm, *Square* is called only one time. After the first time, only *Triangle* and *Rhombus* could be possibly called. But every time *Triangle* and *Rhombus* are called, t decreases by 2. Since $t = n$ initially, the number of necessary queries is $\log n + c$ (c is a constant which used to handle the basic case). Therefore, the algorithm reaches the lower bound of the problem.

It is also worth noticing that to make the algorithm work properly, n is necessarily a power of 2 (to keep the shape of the searching space stable and queried points integral). Nevertheless, this technical difficulty could be easily overcome by considering the problem of size $N = 2^{\lceil \log n \rceil}$.

4.4 A general idea for algorithm of d dimension problem. An algorithm for 3-D problem

The previous subsection introduced an algorithm which could reach the lower bound (differed by c constants queries) for 2-D problem. In this subsection, we will try to generalize the idea to get a good algorithm for d-D problem.

Unfortunately, for 3-D (and d-D) problem, it is really hard to generalize the idea of algorithm

Algorithm 5 TRIANGLE

```
function TRIANGLE(x, y, t)
    if t = 2 then
        Query all 4 points in the searching space and return the answer
    x_1 = x
    y_1 = y − t/2
    i, b = query(x_1, y_1)
    if i = 0, b = 1 then
        return Rhombus(x − t/4, y − t/4, t/2)
    if i = 1, b = −1 then
        return Triangle(x, y, t/2)
    if i = 1, b = 1 then
        return Triangle(x, y − t/2, t/2)
```

for 2-D. One of the main difficulties is the complicacy of the partition of the query. Figure 4 illustrates the partition of the searching space under the effect of 1 query.

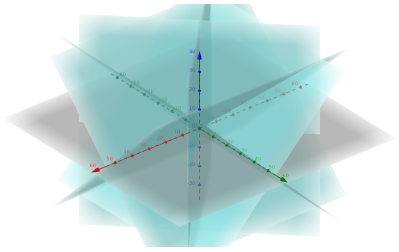


Figure 4: The division of the searching space caused by a query in 3 Dimension

Hence, to generalize the algorithm, we need to take a different point of views. Back to the algorithm 4, there are a remarkable note regarding with the *Rhombus* procedure. Different from *Square* and *Triangle*, once the algorithm enters *Rhombus* procedure, it will stay forever in *Rhombus* procedure. It means if we query at the center of a rhombus, we could reduce the searching space by 4 times while retain the shape. We could say that the rhombus is *stable* under the partition of the query.

Now, if the initial searching space's shape is a rhombus itself, the previous algorithm could be simplified considerably instead of having 3 procedures (only *Rhombus* is necessary). Therefore, one of the natural idea which could be deployed here is to turn the initial searching space from a square to a rhombus and only apply the *Rhombus* procedure. This could be accomplished by putting the square inside the rhombus as illustrated in Figure 5. Surprisingly, the algorithm resulted from this idea is identical to the previous shown algorithm.

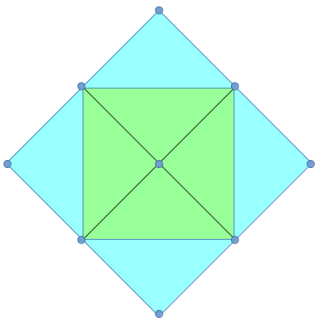
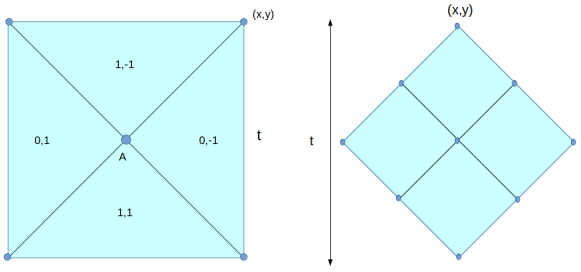
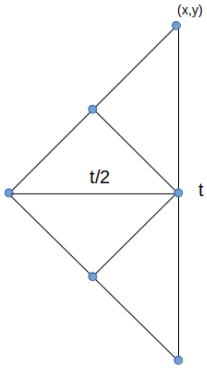


Figure 5: Square inscribed in a rhombus



(a) Square procedure

(b) Rhombus (diamond) procedure



(c) Triangle procedure

Figure 3: Illustration of the 3 query strategies and how searching space is separated

To tackle the general problem for d-D, the idea is described as follow: Inscribing the initial searching space by a *stable* shape (under the partition of the query) and keep querying at the *center*. In case $d = 2$, we accomplish this by using the rhombus. If we could find a proper polyhedron, we could also solve the problem for d-D within $d \log_{2d} n + c$ queries. In case $d = 3$, such shape is the octahedron (Figure 6).

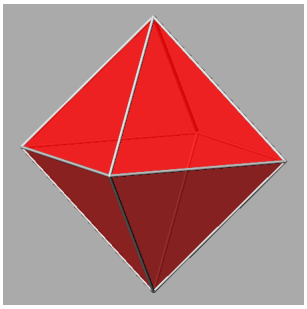
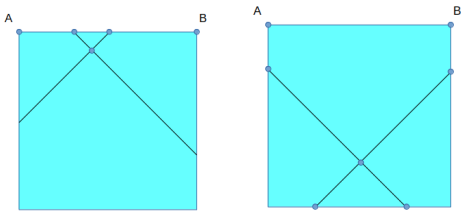


Figure 6: The octahedron

The last noticeable remark is the increase of the searching space on the algorithm. For $d = 2$, the searching space needs to be doubled (since the area of smallest rhombus which could contain a square $n \times n$ is $2n^2$). For $d = 3$, the number is 18.

4.5 A quantum algorithm in 2-D problem

This part of the report will be devoted to devise a quantum algorithm, which could surpass the lower bound in the classical setting (which is $\log n$ queries). In reality, we will utilize the algorithm in 1-D problem. One of the most natural idea is to use the algorithm of 1-D problem in each



(a) The segment of AB is partitioned into 3 parts (instead of 2)

(b) The segment of AB is not partitioned at all

Figure 7: Illustration of the difficulty of the idea of considering each dimension separately

dimension by only considering the boundary of the searching space. Unfortunately, this idea is difficult to work out because there is a lot of cases where the problem in boundaries violating the condition of OSP in 1 dimension (illustrated in Figure 7).

To avoid such difficulty, we will choose different axes which satisfies: Each of the axis is parallel with one of the partitioned line. It appears that two main diagonals could satisfy this property. Therefore, the main idea of the algorithm is: we used the algorithm for 1-D in two main diagonals of the original searching space. Given the answer on each diagonal be (x_0, y_0) and (x_1, y_1). Thus the answer will be one of the intersections of two pair of lines $y = x - x_0 + y_0 / y = x - x_0 + y_0 - \frac{1}{2}$ and $y = -x + x_1 + y_1 / y = -x + x_1 + y_1 + \frac{1}{2}$ (since the intersection of the partitioned line and main diagonal might not be integral) (illustrated by Figure 8). The problem in 2-D, therefore, is reduced to the problem in 1-D, in 2 main diagonals.

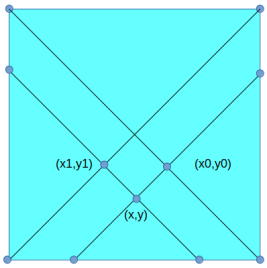


Figure 8: The main idea of the quantum algorithm

The behavior of the oracle to the entire space is illustrated in Figure 9. Due to the symmetry of 2 diagonals, it is sufficient to demonstrate how to find (x_1, y_1) in the diagonal $y = x$.

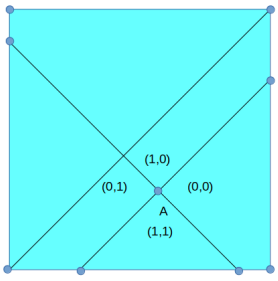


Figure 9: The value of (i, b) returned on all partitions

Firstly, the diagonal $y = x$ is separated into 2 parts: The first part is contained in the partition whose $b = 1$. The other part resides in the partition whose $b = 0$. Therefore,

$$O_{j_0,j_1} |x\rangle |x\rangle |a_1\rangle = |x\rangle |x\rangle |a_1 \oplus b\rangle \tag{7}$$

O_{j_0,j_1} behaves on the qubit a_1 exactly like the oracle O_j of the OSP 1-D. For the sake of simplicity, we will use O_j instead of O_{j_0,j_1} for the rest of this subsection.

Next, we defined a modification which could help the unitary matrices of 1-D algorithm to solve the 2-D OSP.

Definition 4.1. Given U be a unitary matrix that has the effect: $U |x\rangle = \sum_{i=0}^{n-1} \alpha_i |i\rangle$, then the after the modification, we have the following U' unitary matrix:

1. If $x \neq y$, then $U' |x\rangle |y\rangle = |x\rangle |y\rangle$
2. If $x = y$, then $U' |x\rangle |x\rangle = \sum_{i=0}^{n-1} \alpha_i |i\rangle |i\rangle$

The following lemma will explain how this modification works.

Lemma 4.2. *Let* $\phi_i = U_i O_j U_{i-1} ... O_j U_0 |0\rangle$ *and* $\phi'_i = U'_i O_j U'_{i-1} ... O_j U'_0 |0\rangle |0\rangle$ *where* U'_i *is transformed from* U_i *under the operator defined by definition 4.1. If* $\phi_i = \sum_{i=0}^{n-1} \alpha_i |i\rangle$ *then* $\phi'_i = \sum_{i=0}^{n-1} \alpha_i |i\rangle |i\rangle$.

Proof. The proof is inductive. For $i = 0$, $\phi_0 = \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} |i\rangle, \phi'_0 = \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} |i\rangle |i\rangle$, which makes the base case true.

Suppose that the lemma holds until $i = k$. We will prove the lemma is true for $k + 1$. By equation 7, we can further have: if $O_j \phi_k = \sum_{i=0}^{n-1} \alpha_i |i\rangle$ then $O_j \phi'_k = \sum_{i=0}^{n-1} \alpha_i |i\rangle |i\rangle$.

Moreover, $\phi_{k+1} = U_{k+1} O_j \phi_k, \phi'_{k+1} = U'_{k+1} O_j \phi'_k$.

Suppose that $U_{k+1} |i\rangle = \sum_{l=0}^{n-1} \beta_{i,l} |l\rangle$, then

$$\phi_{k+1} = U_{k+1}(O_j \phi_k) = U_{k+1}(\sum_{i=0}^{n-1} \alpha_i |i\rangle) = \sum_{i=0}^{n-1} \alpha_i (U_{k+1} |i\rangle) = \sum_{i=0}^{n-1} (\sum_{l=0}^{n-1} \alpha_l \beta_{l,i}) |i\rangle \tag{8}$$

$$\phi'_{k+1} = U'_{k+1}(O_j \phi'_k) = U'_{k+1}(\sum_{i=0}^{n-1} \alpha_i |i\rangle |i\rangle) = \sum_{i=0}^{n-1} \alpha_i (U'_{k+1} |i\rangle |i\rangle) = \sum_{i=0}^{n-1} (\sum_{l=0}^{n-1} \alpha_l \beta_{l,i}) |i\rangle |i\rangle \tag{9}$$

This proves the lemma. □

Therefore, if the 1-D algorithm solves the normal 1-D problem, so does the 1-D problem in the diagonal. Since the algorithm for 2 dimension has to used 2 times 1-D algorithm for 2 diagonal, the query complexity will be doubled. As introduced, since there is an $0.433 \log n$ algorithm for 1-D problem, we have an algorithm for 2-D one with complexity equal to $2 \times 0.433 \log n = 0.866 \log n$, better than classical lower bound $\log n$.

4.6 A lower bounds of quantum setting

So far, we could see that quantum computing surpasses the classical one to a certain extent. However, one might wonder how far quantum computing could achieve. Is there a way to solve, for example, 2-D OSP in a constant c queries? This last subsection will deliver the answer for that question.

The lower bound for 2-D OSP is a natural question, especially we already had some critical results for the query complexity lower bound of 1-D OSP. In fact, for 1-D OSP, the lower bound for quantum setting was firstly established to be $\frac{\log n}{2 \log \log n}$ [10]. The result was gotten better to be $\frac{1}{12} \log n - O(1)$ in [3]. The current best lower bound is $0.220 \log n$, proved in [7].

In this report, we will prove that even in quantum setting, the 2-D OSP required at least $c \log n$ (c is a constant which is determined later) queries to accomplish the task. The technique we used here is an extension of the proof of query complexity lower bound of 1-D OSP presented in [3]. My proof modified the main procedure to adapt it to 2D problem. The lower bound is resulted from the following theorem.

Theorem 4.3. *Let* $q \in R, q > 1, t \in \mathbb{N}, u \in \mathbb{N}$ *and*

$$q' = \frac{\sqrt{2}}{\sqrt{t}} + \frac{2}{q-1}$$

be such that $q(q')^u < 1$. *Then at least* $\frac{\log n}{u \log t} - O(1)$ *queries are necessary for the quantum searching.*

Proof. The detail of the proof can be found in the appendix. □

By the theorem, we can choose q, t, u to satisfy the condition of theorem. For instance, we can take $q = 18.3, t = 8, u = 8$. The lower bound is thus $\frac{1}{24} \log n + O(1)$.

5 Summary

In this report, we already demonstrated the power of quantum computing by investigating the classical algorithm, classical lower bounds and quantum algorithm for 2 problems: Minimizing Convex M-Lipschitz Array and OSP 2-D. In fact, There are also a lot of rooms for improvement. The classical lower bound for the first problem is obtained under some conditions, which could be improved. The quantum algorithm for the general OSP d-D is still left uncovered. These are the potential questions which could be worked out in the future.

References

[1] E. Farhi, J. Goldstone, S. Gutmann and M. Siper, e-print quant-ph/9901059

[2] Andrew M. Childs, Andrew J. Landahl and Pablo A. Parrilo, *Quantum algorithms for the ordered search problem via semidefinite programming*, PHYSICAL REVIEW A 75, 032335 (2007)

[3] Ambainis, A.: A better lower bound for quantum algorithms searching an ordered list. Proc. of 40th IEEE FOCS (1999) 352–357.

[4] Lov Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM Symposium on Theory of Computing*, pages 212–219, 1996.

[5] Peter Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM *journal on computing*, 26(5):1484–1509, 1997.

[6] Bubeck S., eprint arXiv:1405.4980

[7] P. Hoyer, J. Neerbek, Y. Shi, eprint quant-ph/0102078

[8] A. Nemirovski. Information-based complexity of convex programming. Lecture Notes, 1995.

[9] Bubeck S. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3-4):231–357, 2015.

[10] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, quant-ph/9812057

6 Appendix

6.1 Proof of lemma 3.3

Proof. We will construct A step by step. Initially, we take A_0 have the following form:

$$A_0[i] = |i - \frac{l_0 + 1}{2}|M$$

Apparently, $S_0[1] = -S_0[l_0] = -M < 0$, thus A_0 has $s_0 = [1, l_0]$ satisfying $(*)$.

Suppose that our A_k has all the satisfying segments $s_0, s_1, ..., s_k$. We will slightly modify A_k to create A_{k+1} having s_{k+1} while keeping other s_i. Simulate the algorithm V on A_k until the $(k+1)^{th}$ step. Denote $X = \{x_i | 1 \leq i \leq k+1\}$ is the set of queried points. There are two possibilities:

1. $x_{k+1} \in s_k$: For simple analysis, we can further suppose $s_k = [1, l_k], A_k[1] = A_k[l_k] = 0, S_k[1] = -S_k[l_k] = -M'$ and $x_{k+1} \geq \frac{l_k+1}{2}$. This will not pose any problem since the procedure carried out below can be modified correspondingly. We will define the sequence $S_{k+1}[i]$ (thus define A_{k+1}) as follow:

$$S_{k+1}[i] = \begin{cases} -M' & \text{if } i = 1, 2 \\ -\frac{M'}{2^{N-k-1}-3} & \text{if } 2 < i < 2^{N-k-2} - 1 \\ \frac{M'}{2^{N-k-1}-3} & \text{if } 2^{N-k-2} - 1 \leq i \leq l_k - 2 \\ M' & \text{if } i = l_k - 1 \\ S_k[i] & \text{otherwise} \end{cases}$$

Notice that comparing with A_k, only elements in s_k of A_{k+1} are modified by our procedure because:

$$\begin{aligned} A_{k+1}[l_k] &= A_{k+1}[1] + \sum_{i=1}^{l_k-1} S_{k+1}[i] \\ &= A_k[1] + \sum_{i=1}^{l_k-1} S_{k+1}[i] \\ &= A_k[1] - 2M' - \frac{M'}{2^{N-k-1}-3}(2^{N-k-2} - 4) + \frac{M'}{2^{N-k-1}-3}(l_k - 2^{N-k-2}) + M' \\ &= 0 - M' + \frac{M'}{2^{N-k-1}-3}(2^{N-k-1} - 3) \\ &= 0 \end{aligned} \tag{10}$$

Therefore, all the queries of algorithm V until i^{th} one remain unchanged and thus, makes $s_i, \forall i \leq k$ still valid. Moreover, A_{k+1} is still a convex array. Considering $s_{k+1} = [3, 2^{N-k-1} - 5]$, we have:

   (a) $s_{k+1} \subset s_k$
   (b) $x_{k+1} \geq \frac{l_k+1}{2} > 2^{N-k-1} - 5$. Therefore, the first $k+1$ queries cannot touch s_{k+1}
   (c) By the construction, the minimum point of A_{k+1} (which is $2^{N-k-2} - 1$) is inside s_{k+1}.
   (d) $A_{k+1}[3] = A_{k+1}[2^{N-k-1} - 5]$ and $S_{k+1}[3] = -S_{k+1}[2^{N-k-1} - 5] < 0$

2. $x_{k+1} \notin s_k$: We can use the similar argument of the previous case.

□

This concludes our construction of A and also the proof for the lower bounds of this problem in classical settings. For N large enough, $\log(2^N - 7) \geq N - 1$. Therefore, the lower bound is $\log n - 5$. The argument that we needs M to be large enough because at each step of the construction, M' will be decreased by $2^{N-k-1} - 3$ times. In the case where $S[i]$ is integer, M have to be necessarily large enough. Nevertheless, if we work on the case where $S[i]$ is real, $|M|$ is not necessarily large to make our proof valid.

6.2 Proof of theorem 4.3

To enable to readability, the proof below will cover every part of [3], including the redo of some proofs.

Before working on the detail, we define technical lemmas:

Definition 6.1. Given q and $'q$ be 2 probability distribution. The *variational distance* of q and q' is $\sum_x |p(x) - p'(x)|$.

Definition 6.2. An instance (j_0, j_1) is an instance of the $2D$ problem whose oracle satisfies:

$$i, b = \begin{cases} 1,1 & \text{if } |x - j_0| \leq |y - j_1|, y > j_1 \\ 1,0 & \text{if } |x - j_0| \leq |y - j_1|, y \leq j_1 \\ 0,1 & \text{if } |x - j_0| > |y - j_1|, x > j_0 \\ 0,0 & \textit{otherwise} \end{cases} \tag{11}$$

Where i, b has the same meaning in the problem definition.

This definition is just to get rid of the technical difficulties which might occur because of the uncertainty of i (in case if $|x - j_0| = |y - j_1|$). The oracle is still valid, just more consistent, which turns out to be a great help for our proof.

Lemma 6.1. *Let ψ and ϕ be 2 superpositions such that $\|\psi - \phi\| \leq \epsilon$. Then the total variational distance resulting from measurements of ψ and ϕ is at most 4ϵ.*

Now, for every algorithm A which works on the problem in the quantum setting, they will have the following form:

$$\psi = U_T O U_{T-1} O ... O U_0 \left|0\right\rangle$$

where U_i is the unitary matrix. The algorithm calculates the result by measuring the final superposition ψ.

Suppose that the quantum algorithm A have two different inputs: instance j_0, j_1 and j_0', j_1'. Suppose that A is able to give the result with the probability greater than $\frac{3}{4}$. Then if we apply A to 2 instances with ψ and ψ' be 2 superpositions then the total variation is greater than $2 \times (\frac{3}{4} - \frac{1}{4}) = 1$. By lemma 6.1, we have $\|\psi - \psi'\| \geq \frac{1}{4}$.

Now, which those definitions and lemma in hand, we will prove the theorem 4.3

Firstly, we describe a procedure called *subdivide*. This is the my main contribution comparing to the original proof. Considering an inscribed rhombus and a subspace of the original searching space bounded by a cross. Notice that the cross is made up by 2 pairs of parallel lines. The distance between each pair is m, and they intersect the borders of the rhombus forming the segment $[(l-1)m+1, lm]$ (illustrated by Figure 10). For simplicity, we will shorten the subspace bounded by a cross to the cross $[(l-1)m+1, lm]$. Then the subdivide procedure takes the cross $[(l-1)m+1, lm]$ and returns a cross $[(l'-1)m'+1, l'm']$.

subdivide(m,l,s)

1. Let $m' = m/t$. Split $[(l-1)m+1, lm]$ into t subintervals: $[(l-1)m+1, (l-1)m+m']$, $[(l-1)m+m'+1, (l-1)m+2m']$, ..., $[(l-1)m+(t-1)m'+1, lm]$. For each interval, we create the corresponding cross.

2. Simulate the first s steps (and unitary operator between queries) on instance $(lm+1, lm+1)$. Let

$$\left|\phi_i\right\rangle = U_{i-1} O ... O U_0 \left|0\right\rangle \tag{12}$$

   be the superposition before the i^{th} query step, $\left|\psi_i\right\rangle$ be its part corresponding to querying x_{j_0,j_1} for j_0, j_1 which is inside the cross $[(l-1)m+1, lm]$, $\left|\psi_{i,r}\right\rangle$ be the part of $\left|\psi_i\right\rangle$ corresponding for x_{j_0,j_1} for j_0, j_1 which is inside the cross $[(l-1)m+(r-1)m'+1, (l-1)m+rm']$ and $\left|\chi_i\right\rangle$ is the part corresponding to querying x_{j_0,j_1} for j_0, j_1 which is inside the square formed by 2 pairs of parallel lines.

3. For every $1 \leq r \leq t$, compute the sum:

$$S_r = \| \left|\psi_{s,r}\right\rangle \| + q \| \left|\psi_{s-1,r}\right\rangle \| + q^2 \| \left|\psi_{s-2,r}\right\rangle \| + ... + q^{s-1} \| \left|\psi_{1,r}\right\rangle \|$$

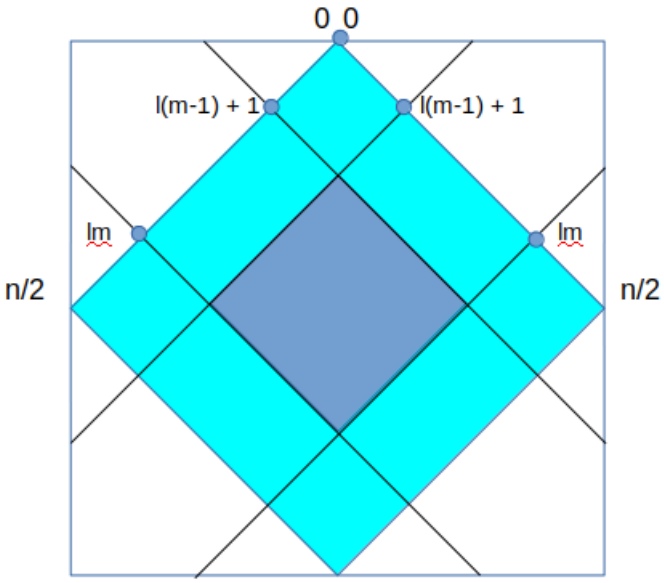


Figure 10: Illustration of the "cross"

4. Take the r that minimizes S_r and set $l' = (l-1)t + r$. Then the newly created cross is $[(l'-1)m'+1, l'm'] = [(l-1)m + (r-1)m' + 1, (l-1)m + rm']$.

Next we will analyze this procedure by the quantity:

$$S = \| |\psi_s\rangle \| + q\| |\psi_{s-1}\rangle \| + q^2\| |\psi_{s-2}\rangle \| + ... + q^{s-1}\| |\psi_1\rangle \|$$

By the similar manner, we defined ϕ', ψ', S' given the instance $(l'm'+1, l'm'+1)$ corresponding to ϕ, ψ, S .

Lemma 6.2. *With S and S' defined in the **subdivide** procedure, q' defined in theorem 4.3, we have:*

$$S' \leq q'S$$

This lemma can be obtained by using the following claims:

Claim 6.1.

$$\| |\phi_i\rangle - |\phi'_i\rangle \| \leq 2(\| |\psi_1\rangle \| + ... + \| |\psi_{i-1}\rangle \|) \tag{13}$$

Proof. By induction, if $i = 1$ then $|\phi_i\rangle = |\phi'_i\rangle$.

Next, we assume that $\| |\phi_{i-1}\rangle - |\phi'_{i-1}\rangle \| \leq 2(\| |\psi_1\rangle \| + ... + \| |\psi_{i-2}\rangle \|)$. Then,

$$\begin{aligned} \| |\phi_i\rangle - |\phi'_i\rangle \| &= \|U_i O_{lm} |\phi_{i-1}\rangle - U_i O_{l'm'} |\phi'_{i-1}\rangle \| \\ &= \|O_{lm} |\phi_{i-1}\rangle - O_{l'm'} |\phi'_{i-1}\rangle \| \\ &\leq \|O_{lm} |\phi_{i-1}\rangle - O_{l'm'} |\phi_{i-1}\rangle \| + \|O_{l'm'} |\phi_{i-1}\rangle - O_{l'm'} |\phi'_{i-1}\rangle \| \\ &\leq \|O_{lm} |\phi_{i-1}\rangle - O_{l'm'} |\phi_{i-1}\rangle \| + \| |\phi_{i-1}\rangle - |\phi'_{i-1}\rangle \| \end{aligned} \tag{14}$$

The second part is simply from the induction hypothesis. The first part could be analyzed as follow: All the elements outside the cross is indifferent with O_{lm} and $O_{l'm'}$. Thus, the maximum l_2 distance is actually $2\| |\psi_{i-1}\rangle \|$. Therefore

$$\begin{aligned} \| |\phi_i\rangle - |\phi'_i\rangle \| &\leq \|O_{lm} |\phi_{i-1}\rangle - O_{l'm'} |\phi_{i-1}\rangle \| + \| |\phi_{i-1}\rangle - |\phi'_{i-1}\rangle \| \\ &\leq 2\| |\psi_{i-1}\rangle \| + 2(\| |\psi_1\rangle \| + ... + \| |\psi_{i-2}\rangle \|) = 2(\| |\psi_1\rangle \| + ... + \| |\psi_{i-1}\rangle \|) \end{aligned} \tag{15}$$

□

Projecting both $|\phi_i\rangle$ and $|\phi'_i\rangle$ to the subspace corresponding the part inside the cross $[(l'-1)m'+1, l'm']$. We have the following claim:

Claim 6.2.

$$\| |\phi_{i,r}\rangle - |\phi'_i\rangle \| \leq 2(\| |\psi_1\rangle \| + ... + \| |\psi_{i-1}\rangle \|) \tag{16}$$

This means:

$$\begin{aligned} S' = \sum_{i=1}^{s} q^{s-i} \| |\psi'_i\rangle \| &\leq \sum_{i=1}^{s} q^{s-i}(\| |\phi_{i,r}\rangle \| + 2(\| |\psi_1\rangle \| + ... + \| |\psi_{i-1}\rangle \|)) \\ &= \sum_{i=1}^{s} q^{s-i} \| |\phi_{i,r}\rangle \| + 2\sum_{i=1}^{s}(q^{s-i} \sum_{j=1}^{i-1} \| |\phi_j\rangle \|) \\ &= S_r + 2\sum_{j=1}^{s}(\| |\phi_j\rangle \| \sum_{i=j+1}^{s} q^{s-i}) \\ &\leq S_r + 2\sum_{j=1}^{s}(\| |\phi_j\rangle \| \sum_{i=j+1}^{\infty} q^{s-i}) \\ &= S_r + 2\sum_{j=1}^{s}(\| |\phi_j\rangle \| \frac{q^{s-j}}{q-1}) \\ &= S_r + \frac{S}{q-1} \end{aligned} \tag{17}$$

Now we just need to bound S_r, which is, in fact, achievable by the following claim.

Claim 6.3.

$$S_r \leq \frac{\sqrt{2}S}{\sqrt{t}} \tag{18}$$

Proof. Indeed, we have $\| |\psi_i\rangle \|^2 + \| |\chi_i\rangle \|^2 = \| |\psi_{i,1}\rangle \|^2 + \| |\psi_{i,2}\rangle \|^2 + ... + \| |\psi_{i,t}\rangle \|^2$. By the fact that $\| |\psi_i\rangle \|^2 \geq \| |\chi_i\rangle \|^2$ (since $|\chi_i\rangle$ is contained in $|\psi_i\rangle$) and Cauchy-Schwartz inequality, we have:

$$2\| |\psi_i\rangle \|^2 \geq \frac{(\| |\psi_{i,1}\rangle \| + \| |\psi_{i,2}\rangle \| + ... + \| |\psi_{i,t}\rangle \|)^2}{t} \tag{19}$$

Equivalently,

$$\| |\psi_i\rangle \| \geq \frac{\| |\psi_{i,1}\rangle \| + \| |\psi_{i,2}\rangle \| + ... + \| |\psi_{i,t}\rangle \|}{\sqrt{2t}} \tag{20}$$

Since S is just a weighted sum of $\| |\psi_i\rangle \|$ and $S_1, S_2, ..., S_t$ are weighted sums of $\|\psi_{i,1}\|, ..., \|\psi_{i,t}\|$, respectively. Hence $S \geq \frac{S_1+...+S_t}{\sqrt{2t}} \geq \sqrt{\frac{t}{2}} S_r$. The claim 3 thus holds, which also proves the lemma 6.2.

□

Next, we will use the subdivide procedure to construct two inputs that are not distinguishable by A. The process is as follow: Let $v = \lceil \log\Big(\frac{1}{10}(1 - q(q')^u)(1 - \frac{1}{q})\Big) / \log q' \rceil$.

1. Let $m = n/2, l = 1, s = 1$
2. While $m > t^v$ repeat:
   (a) u times do subdivide(m, l, s) and set $l = l', m = m'$

(b) $s = s + 1$

3. $v - u$ times do subdivide(m, l, s) and set $l = l', m = m'$

   At the beginning, $m = n/2$. Each step decreases m by a factor of t^u while $m > t^v$. Consequently, we needs at least $\frac{\log n/2t^v}{\log t^u} = \frac{\log n}{u \log t} - O(1)$.

The final cross crafted by this procedure has the following property: If the instance has the result inside this cross, it is hard for A to distinguish one from another. This idea leads to Lemma 3.6.

Lemma 6.3. *With q, q', u defined in theorem 4.3, we have:*

1. *At the beginning of step* 2.1*, S is at most $\frac{1}{1-q(q')^u}$*
2. *At the end of step* 2.1*, S is at most $\frac{q}{1-q(q')^u}$*

Proof. By induction, with $s = 1$ then $S = \| |\psi_1\rangle \| < 1$ and $1 < \frac{1}{1-q(q')^u}$.

Suppose that the proof is true until some $s \leq 1$, then by lemma 3.5, new S is decreased by q' times. Therefore, $S' \leq \frac{q}{1-q(q')^u}$.

In the next step, new S will be:

$$\sum_{i=1} s + 1 q^{s+1-i} \| |\psi_i\rangle \| = \| |\psi_{s+1}\rangle \| + qS \leq 1 + \frac{q}{1 - q(q')^u} = \frac{1}{1 - q(q')^u} \tag{21}$$

This concludes the proof of lemma 3.6.

By definition of v, we have:

$$S \leq \frac{\frac{1}{10}(1 - q(q')^u)(1 - \frac{1}{q})}{1 - q(q')^u} = \frac{1}{10}(1 - \frac{1}{q}) \tag{22}$$

Therefore, $\| |\psi_i\rangle \| < \frac{1-\frac{1}{q}}{10q^{s-i}}$ (since $S = \sum_{i=1}^{s} q^{s-i} \| |\psi_i\rangle \| \geq q^{s-i} \| |\psi_i\rangle \|$. □

Finally, we consider A working on 2 instances: instance (lm, lm) and $(lm - 1, lm - 1)$. The final superposition of 2 inputs are

$$|\varphi\rangle = U_T O_{lm} U_{T-1} ... U_0 |0\rangle, |\varphi'\rangle = U_T O_{lm-1} U_{T-1} ... U_0 |0\rangle \tag{23}$$

We denote

$$|\varphi_i\rangle = U_T O_{lm-1} ... U_{i+1} O_{lm-1} U_i O_{lm} U_{i-1} ... O_0 |0\rangle \tag{24}$$

Thus, $\varphi = \varphi_s, \varphi = \varphi_0$.

Claim 6.4.

$$\| |\varphi_i\rangle - |\varphi_{i-1}\rangle \| \leq \frac{1 - \frac{1}{q}}{5q^{s-i}} \tag{25}$$

Proof. We denote:

$$|\phi_i\rangle = U_i O_{lm} U_{i-1} ... O_0 |0\rangle \tag{26}$$

The part that O_{lm-1} and $O_l m$ are different is always inside $|\psi_i\rangle$. Therefore,

$$\begin{aligned} \| |\varphi_{i+1}\rangle - |\varphi_i\rangle \| &= \| O_{lm} |\phi_i\rangle - O_{lm-1} |\phi_i\rangle \| \\ &= \| O_{lm} |\psi_i\rangle - O_{lm-1} |\psi_i\rangle \| \\ &\leq 2 \| |\psi_i\rangle \| \\ &= \frac{1}{5} \frac{1 - \frac{1}{q}}{q^{s-i}} \end{aligned} \tag{27}$$

□

This leads us to the final inequality:

$$\| \, |\varphi_0\rangle - |\varphi_s\rangle \, \| \leq \sum_{i=1}^{s} \| \, |\varphi_i\rangle - |\varphi_{i-1}\rangle \, \| \leq \sum_{i=0}^{\infty} \frac{1}{5} \frac{1 - \frac{1}{q}}{q^i} = \frac{1}{5} \tag{28}$$

This contradicts the fact delivered by lemma 6.1 and prove the theorem.