5-2022

The Primitive Root Problem: a Problem in BQP

Shixin Wu
Rose-Hulman Institute of Technology, WUS4@rose-hulman.edu

The Primitive Root Problem: a Problem in **BQP**

Shixin Wu
Department of Mathematics, Rose-Hulman Institute of Technology

May
2022

Abstract

Shor's algorithm proves that the discrete logarithm problem is in **BQP** [8]. Based on his algorithm, we prove that the primitive root problem, a problem that verifies if some integer g is a primitive root modulo p where p is the largest prime number smaller than 2^n for a given n, which is assumed to be harder than the discrete logarithm problem, is in **BQP** by using an oracle quantum Turing machine.

1 Introduction

Since the birth of the von Neumann model, electrons, able to represent 0 or 1, have been the predominant matter that a computer uses to compute. With the development of quantum physics, it has been discovered that a quantum particle can be in a superposed state, in that it represents "both 0 and 1". Although a quantum particle collapses to either 0 or 1 through measurement, it provides the possibility to process multiple pieces of information simultaneously without measurement. This inspires computer scientists to inquire: given the same problem, how does a quantum computer perform compared to a classical computer?

Paul Benioff proved it is possible to use quantum mechanics to implement a Turing machine in 1980 [3]. This means that given a problem solvable with certain efficiency with a classical Turing machine, there exists a quantum Turing machine that can achieve the same efficiency for the problem. As a result, his paper proves a deterministic complexity class is a subset of its quantum analog and the quantum Turing machine is at least as fast as a classical Turing machine. From his work, for example, we can derive that if a problem can be solved in polynomial time with a classical Turing machine with high probability of correctness over 2/3, it can be solved in the same time complexity with a quantum Turing machine. This motivated future researchers to look for problems that are faster to solve for quantum computers than classical computers.

David Deutsch and Richard Josza published the Deutsch-Josza problem in 1992 [7]. The problem proposes a function $f_L : \{0,1\}^n \to \{0,1\}$ where $L \subseteq \{0,1\}^n$ is the language that f decides on. It is promised that $L = \varnothing$ or $|L| = 2^{n-1}$. The task of the problem aims to determine the cardinality of L. A deterministic Turing machine has to query d_L for at most $2^{n-1} + 1$ times, while a quantum Turing machine only requires polynomial time to solve the problem. The quantum algorithm, the Deutsch-Josza algorithm, that solves this problem is among the first quantum algorithms that perform better than the classical algorithms.

Using a similar idea as Deutsch and Josza, Daniel Simon published Simon's problem in 1994 [9] that also proposes a periodic function $\phi : \{0,1\}^n \to \{0,1\}^n$ such that $\phi(x) = \phi(y)$ if and only if $x = y$ or $x = y \oplus s$ where $s \in \{0,1\}^n$. The problem is to determine the period s. A probabilistic Turing machine has to query f for at least $2^{\frac{n}{3}}$ times to succeed with probability over 2/3, while a quantum Turing machine while a quantum Turing machine only requires $n + 1$ calls to succeed with probability over 3/4. Like the Deutsch-Josza algorithm, Simon's algorithm proposes a quantum algorithm that performs better than the classical algorithms, but Simon's algorithm is able to find the period of a periodic function, which has inspired Shor's algorithm that solves the discrete logarithm problem efficiently.

Observe Simon's problem, if there is some $s, y \in \{0,1\}^n$ such that $\phi(y \oplus s) = \phi(y)$, we may say that ϕ is periodic and the goal of Simon's problem is essentially to find the period of ϕ. Based on this idea, Peter Shor published a quantum algorithm in 1994 to solve the discrete logarithm problem, a long-existing problem for which no classical polynomial solution has been found and which is widely used in encryption methods like Diffie-Hellman key exchange. This algorithm, famously referred to as Shor's algorithm, is capable of solving the discrete logarithm problem in polynomial time with a probability of success of over 80% [8]. The significance of Shor's algorithm is not only in computational complexity as it solves a problem long assumed not able to be solved in polynomial time, but also in cryptography. Since it is able to solve the discrete logarithm problem in polynomial time, a quantum computer can break the Diffie-Hellman key-exchange method much more efficiently than a classical computer. Now, with Shor's algorithm, any quantum computer with enough qubits is able to break any encryption system that use discrete logarithm in polynomial time.

Compare and contrast the three famous problems in quantum computing. While Deutsch-Josza and Simon's problem both require finding the property of functions, the latter finds a specific number while the former finds a binary property of the function. Shor's algorithm, since it solves an existing problem, no longer requires the creation of such a function.

In this paper, we prove that the determination of whether an integer is a primitive root of a prime number, an existing problem presumed to be harder than the discrete logarithm problem, is in the same complexity class as the discrete logarithm problem. Our proof is inspired by the combination of the algorithms discussed above in that we solve an existing problem with the help of querying a language decidable in polynomial time with bounded probability of correctness. In Diffie-Hellman key exchange, it is important to use an integer that is the the primitive root of the prime number in the protocol, because it can minimize the probability for the eavesdropper to guess correctly.

In Section 2, we give a formal definition of the problems and concepts used in this paper. Section 3

and Section 4 define the languages to be decided by and input as an oracle to the quantum oracle Turing machine, a Turing machine whose states and content on the tape may be superposed and that queries an oracle about the membership of some strings. The formal proof is given in Section 5. In Section 7, we discuss the implications of the result of this paper.

2 Preliminaries

2.1 Definitions and Theorems

In this section, we give the definitions of the concepts that we use for this paper. For the completeness of this paper, we define the discrete logarithm problem and the primitive root problem.

Definition 2.1. A *discrete logarithm problem* is the problem to find a positive integer $s \in [1, p-1]$, given a prime number p and integers $y, g \in [1, p-1]$ such that $y = g^s \bmod p$.

Definition 2.2. A *primitive root problem* is the problem of given a positive integer n, determine whether g is a *primitive root* of the prime number p for which p is the largest prime number under 2^n. g is said to be a *primitive root* of p if for any integer $s \in [1, p-1]$, there exists an integer y such that $y = g^s \bmod p$.

In this paper, we assume that the primitive root problem is hard, because we do not know how to efficiently factor $p-1$ for a prime number p [10]. Furthermore, we assume that the primitive root problem is at least as hard as the discrete logarithm problem, because the discrete logarithm problem requires finding s to satisfy $y = g^s \bmod p$ for a given y, while the primitive root problem requires for any y, there always exists some $s \in [1, p-1]$ such that $y = g^s \bmod p$. If this assumption holds, this paper proves that the primitive root problem is in the same complexity class as the discrete logarithm problem. We will discuss more regarding this assumption in Section 7.

In order to quantitatively discuss the hardness of the problems, it is necessary to define complexity classes. As it is natural to define quantum analogues of classical complexity classes [6], quantum complexity classes may be defined with quantum Turing machines as classical complexity classes are defined with classical Turing machines. We now define a deterministic Turing machine and analogously define a quantum Turing machine.

Definition 2.3. A *deterministic Turing machine* (DTM) is defined by a triplet (Σ, Q, δ) where Σ is a finite alphabet with an identified blank symbol #; Q is a finite set of states with an identified start state q_1 and a subset of final states; and δ, the deterministic transition function, is defined by:

$$\delta : Q \times \Sigma \to \Sigma \times Q \times \{L, R\}$$

where $\{L, R\}$ indicates the set of tape head movements (L: left; R: right) on an infinitely long two-way tape [5]. The tape head of a DTM writes a character in Σ before moving to the left or right on each step. Its initial position is on the first character of the input.

Definition 2.4. A *quantum Turing machine* (QTM) has exactly the same definition as a DTM except that its transition function is defined as

$$\delta : Q \times \Sigma \to \widetilde{\mathbf{C}}^{\Sigma \times Q \times \{L,R\}}$$

where $\widetilde{\mathbf{C}}$ is the set of $\alpha \in \mathbf{C}$ such that there is a deterministic algorithm that computes the real and imaginary parts of α to within 2^{-n} in time polynomial in n [5]. In other words, every element in $\widetilde{\mathbf{C}}$ is approximated in polynomial amount of time with error no greater than 2^{-n}. If $p, q \in Q, \sigma, \tau \in \Sigma, d \in \{L, R\}$, we use the notation $\delta(p, \sigma)[\tau, q, d]$ to mean that the output of $\delta(p, \sigma)$ in $\widetilde{\mathbf{C}}^{\Sigma \times Q \times \{L,R\}}$ takes (τ, q, d) as the input and outputs an element in $\widetilde{\mathbf{C}}$.

With the DTM and QTM defined, we may formally define an important complexity class *bounded-error quantum probabilistic polynomial time* (**BQP**). If an algorithm is found to solve a problem in **BQP**-time, but none is found that solves the problem in polynomial time with a classical Turing machine, this problem is evidence of the superiority in computation efficiency of quantum computers. The Deutsch-Josza problem, Simon's problem, and the discrete logarithm problem mentioned before all fall into this category of problems.

Definition 2.5 ([11])**. BQP** is the set of computational problems that can be solved in polynomial time by some QTM with probability of correctness $\geq \frac{2}{3}$.

The Deutsch-Josza problem, Simon's problem and the discrete logarithm problem are all in **BQP**, because there are efficient algorithms that solve these problems in polynomial time with bounded correctness. The result of this paper shows that the primitive root problem, a problem at least as hard as the discrete logarithm problem, is also in **BQP**.

In addition to a single string, we may also specify a language as input to a Turing machine. The Turing machine may query if a string is in the language and get the result in one computational step. We give the definition of a Turing machine with an additional input of a language.

Definition 2.6 (Arora 2009, [2])**.** An *oracle TM* is a Turing machine M that has a special read/write tape we call M's *oracle tape* and three special states q_{query}, q_{yes}, q_no. To execute M, we specify, in addition to the input, a language $O \in \{0,1\}^*$ that is used as the oracle for M. Whenever during execution M enters the state q_{query}, the machine moves into the state q_{yes} if $x \in O$ and q_{no} if $x \notin O$, where x is the contents of the special oracle tape. Note that, regardless of the choice of O, a membership query to O counts only as a single computational step. We denote this oracle TM M^O.

If an oracle Turing machine M^O is capable of solving a computational problem in the complexity class C, we say that this computational problem is in C^O.

Similar to the way a quantum Turing machine is used to define a complexity class **BQP**, an oracle Turing machine can be used to define another complexity class $\mathbf{BQP^{BQP}}$, a concept key to this paper.

Definition 2.7. $\mathbf{BQP}^O$ is the set of computational problems that can be solved in polynomial time with probability of correctness $\geq 2/3$ by some oracle Turing machine with input language O. If O is a **BQP**-complete language, since having oracle access to a complete language for **BQP** allows us to solve every problem in **BQP**, we call this complexity class $\mathbf{BQP^{BQP}}$.

This paper shows that the primitive root problem is in $\mathbf{BQP^{BQP}}$ and therefore in **BQP** according to a theorem proved by Bennett et al.

Theorem 2.1 ([4])**.** $\mathbf{BQP} = \mathbf{BQP^{BQP}}$.

We present a sketch of our proof of this theorem. First, it is easy to see $\mathbf{BQP} \subseteq \mathbf{BQP^{BQP}}$. Suppose there is some QTM M that decides L in **BQP**-time, then we modify M with an additional input language O where O can be any language. M^O can decide L within the same running time for any language O, because it does not need to query O at all to decide L in the first place. If O is a **BQP**-complete language, it proves that $L \in \mathbf{BQP^{BQP}}$.

Now we prove $\mathbf{BQP^{BQP}} \subseteq \mathbf{BQP}$. Suppose there is some oracle QTM Q^N that decides L in **BQP**-time where N is a **BQP**-complete language. Let T be the QTM that decides N. We construct a QTM Q' that decides L in **BQP**-time by combining Q^N and T. First, let Q' be the disjoint union of Q^N and T. Then, remove the language input to Q^N and its oracle tape. To link the two disjoint parts, suppose the start state and the two final states of T are p_0, p_{acc} and p_{rej}, let $p_0 = q_{query}$, $p_{acc} = q_{yes}$ and $p_{rej} = q_{no}$. Before going into q_{query}, suppose the content to query N starts from slot i, then Q' moves its tape head to slot i when going into p_0.

To see the running time of Q', we divide Q' into 3 stages. Stage A' starts at the initial state of Q' and ends before Q' is at the state p_0; stage B' starts from p_0 and ends at p_{acc} or p_{rej}; stage C' starts after p_{rej}. We can divide Q^N similarly to stages A, B and C by q_{query}, q_{yes} and q_{no}. Notice that $A' = A$ and $C' = C$. These stages all run in **BQP**-time, because otherwise, Q^N cannot run in **BQP**-time. Furthermore, since T runs in **BQP**-time, stage B' runs in **BQP**-time. Hence, because A', B' and C' all run in **BQP**-time, Q' runs in **BQP**-time.

2.2 Notations

In this paper, we constantly switch between a positive integer and a binary encoded string of the integer. If x is a positive integer, $x_{(2)}$ is the string that encodes x and vice versa. We use $\circ$ to represent the concatenation of strings. If not otherwise specified, x^n means the string of x's of length n if $x = \{0,1\}$.

3 Language Decided by the Oracle Quantum Turing Machine

Recall the definition of the primitive root problem. The goal of this section is to define a language L with a specific g such that if an oracle QTM can decide L, then it can decide whether g is the primitive root of p where p is the largest prime number smaller than 2^n given any arbitrary positive integer n. Artin's conjecture states that if the positive integer g is not a perfect square, there are infinitely many primes p such that g is a primitive root modulo p [10]. Therefore, assuming the correctness of Artin's conjecture, a Turing machine cannot "cheat" to compute in polynomial time by storing all the primes p that g is a primitive root modulo p. In this paper, we specifically pick $g = 2$.

The language L is defined to be

$$L = \{1^n | \exists y, \forall s, y \neq 2^s \bmod p\},$$

where $1 \leq y < p < 2^n$ for an integer y and p is the largest prime number less than 2^n. This language can be interpreted as $1^n \in L$ if and only if 2 is not a primitive root of p.

For an oracle QTM that decides L, instead of directly inputting to the oracle TM n encoded in binary, we encode n in unary so that the input size is strictly n and the following discussions of running time can be safely based on n. The choice of 1^n is in L if and only if 2 is not the primitive root of p instead of vice versa may be counter-intuitive, but using the existence of y is key to the following constructions.

4 Language Input as an Oracle

Recall the definition of the oracle Turing machine that a language is an additional input to it. The goal of this section is to define a language $D \in \mathbf{BQP}$, which can be reduced to a **BQP**-complete language O. This is possible because there exists at least one **BQP**-complete language. For example, the local Hamiltonian eigenvalue sampling problem, the phase estimation sampling problem, and the local unitary matrix average eigenvalue problem are all proven to be in **BQP**-complete by Pawel Wocjan and Shengyu Zhang [12].

We first define a language $D = \bigcup_{n \in \mathbb{Z}^+} D_n$ in **BQP** where

$$D_n = \{y_{(2)} | \exists s \in \mathbb{Z}^+, y = 2^s \bmod p_n, |y_{(2)}| = n\} \cup \{y_{(2)} | y = 0 \text{ or } y \geq p, |y_{(2)}| = n\},$$

for which p_n is the largest prime number smaller than 2^n and $y_{(2)}$ is the binary encoding of y. Thus, D is the language of all binary encoded strings of y of any length, say n, where $y = 2^s \bmod p_n$ or $y = 0$ or $y \geq p$. Note that multiple binary representations of an integer may be in D. For example, if $n = 3$, $2 = 2^1 \bmod 7$ and if $n = 4$, $2 = 2^1 \bmod 13$. Observe that since $010 \in D_3$ and $0010 \in D_4$, $010, 0010 \in D$.

To see that $D \in \mathbf{BQP}$, we propose an algorithm that decides D in **BQP**-time. Specifically, this algorithm queries two languages P and S in **BQP** and other steps of this algorithm also runs in **BQP**-time. Let P be the set of all binary-encoded prime numbers with the shortest length possible. For example, 10 and 0010 encode both 3 and only 10 is in P. The AKS primality test proves that examining whether a binary encoded integer of length n is prime has time complexity $O(n^6)$ [1], so $P \in \mathbf{BQP}$.

Let

$$S = \{(y_{(2)}, p_{(2)}) | \exists s \in \mathbb{Z}^+, y = 2^s \bmod p\},$$

where $y_{(2)}$ and $p_{(2)}$ are the shortest binary encoding possible for y and p. For example, because $1 = 2^4 \bmod 3$, $(1, 100, 10) \in S$. S is in **BQP**, because given y, p and $g = 2$, Shor's algorithm will find a possible s, which is correct with probability more than 80%. The operations of taking powers and taking mods are required to calculate $2^s \bmod p$ to examine the correctness of s, which are both in polynomial time. If s is correct, accept $(y_{(2)}, p_{(2)})$; otherwise, reject. Since Shor's algorithm and the final examination step both run in **BQP**-time, $S \in \mathbf{BQP}$.

Now, with both P and S defined, we give the algorithm that decides D below.

This algorithm first reads the length n of the input $y_{(2)}$. For every odd number i smaller than 2^n, in descending order, the algorithm queries P, the language of prime numbers, whether i is a prime number. If i is not, try $i - 1$. If i is, the algorithm queries S with $i_{(2)}$. If $i_{(2)} \in S$, then accept $y_{(2)}$; otherwise, reject. If there is no prime number found within polynomial number of largest integers smaller than 2^n, reject $y_{(2)}$.

For the algorithm to decide D in **BQP**-time, the while loop can only repeat a polynomial number of times and must correctly decide $y_{(2)}$ with probability over 2/3. Since Shor's algorithm succeeds in deciding

Algorithm 1 Algorithm that decides D (input: $y_{(2)}$)

$i \leftarrow 1$, $n \leftarrow |y_{(2)}|$
Let $y'_{(2)}$ be the shortest binary encoding of $y_{(2)}$
while $i \leq kn$ for some $k \in \mathbb{Z}^+$ **do**
    Query P with $(2^n - i)_{(2)}$
    if $2^n - i \in P$ **then**
        Query S with $(y'_{(2)}, (2^n - i)_{(2)})$
        if $(y'_{(2)}, (2^n - i)_{(2)}) \in S$ **then**
            Accept $y_{(2)}$
        else
            Reject $y_{(2)}$
        end if
    else
        $i \leftarrow i - 2$
    end if
end while
Reject $y_{(2)}$

correctly with probability over 80%, the codes in the while loop should be executed so many times that the probability of successfully finding a prime number is over 5/6, assuming that all the rejections followed by not finding a prime number are mistakes. We are aware that under some circumstances, failing to find a prime number does not indicate a mistake when the input should be rejected anyway. It is technically true that k can be smaller than the value given in our following discussions, but as we will later find out that k is a constant, the difference caused by such assumption makes little difference in the running time.

The prime number theorem says that the probability of randomly picking a number smaller than 2^n such that the number is prime is $\ln(2)/n$. By the assumption that the primality of an odd number i will not affect the primality of $i-2$, the probability of none of the kn consecutive odd numbers smaller than 2^n being prime is given by $(1 - \ln(2)/n)^{kn}$. Therefore, the probability of success, that is, the probability for this not to happen, is given by

$$1 - \left(1 - \frac{\ln(2)}{n}\right)^{kn} \geq \frac{5}{6}.$$

We then obtain the following equation by solving for k:

$$\frac{\ln(6)}{n \ln\left(1 - \frac{\ln(2)}{n}\right)} \leq k. \tag{1}$$

Next, we approximate the equation by Taylor series with

$$\ln\left(1 - \frac{\ln(2)}{n}\right) \approx -\frac{\ln(2)}{n} - \frac{\ln^2(2)}{2n^2}.$$

Then, Equation 1 is written as:

$$\frac{\ln(6)}{\ln(2) + \frac{\ln^2(2)}{2n}} \leq k.$$

Therefore, the smallest value of k is $\ln(6)/\ln(2) = \log_2 6 = 2.58$. If the while loop repeats $2.58n$ times, the algorithm decides D in polynomial time with probability of correctness over 2/3.

Let O be a **BQP**-complete language and $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function such that $y_{(2)} \in D$ if and only if $f(y_{(2)}) \in O$. If an oracle TM queries D on some string x to decide some language K in **BQP**-time, $K \in \mathbf{BQP}^D$. The oracle TM may as well query O with $f(x)$ and still decide K and then $K \in \mathbf{BQP}^O$. Since O is a **BQP**-complete language, $K \in \mathbf{BQP}^O = \mathbf{BQP}^{\mathbf{BQP}} = \mathbf{BQP}$. We use this idea in the next section where we will define an oracle QTM that decides L in **BQP**-time.

For our purpose, we need to further define a **BQP**-complete language $O' = \bigcup_{m \in \mathbb{Z}^+}$ where $O_m = \{0^i \circ x : |x| + i = m, x \in O\}$. The language O' is a union of strings in O of different lengths.

5 Oracle Quantum Turing Machine

5.1 Construction

Before the construction of the oracle QTM, we must first define a reversible function ζ based on f, the reducing function mentioned in Section 4. Suppose m_n is the length of the longest string $f(z)$ such that $z \in D_n$ if and only if $f(z) \in O$ where D and O are the languages in Section 4. We claim that m_n exists and is polynomial of n, because if $f(x)$ had length superpolynomial in n, it would be impossible for the reducing function f to reduce D in O in polynomial time. For any string x where $|f(x)| = j \leq m_n$, let $f'(x) = 0^i \circ f(x)$ where $i + j = m_n$. Saying $x \in D$ if and only if $f(x) \in O$ is equivalent to saying $x \in D$ if and only if $f'(x) \in O'$ mentioned in Section 4. Given a quantum state $|p \circ q \circ c\rangle$ where $|p| = n$, $|q| = m_n$ and $|c| = 1$, define ζ to be:

$$\zeta(|p \circ q \circ c\rangle) = |p \circ q \oplus f'(p) \circ c\rangle\,,$$

where the last qubit is reserved for the result of the oracle queries. Since f can be applied in polynomial time and m_n is polynomial in n, ζ can be applied in polynomial time as well.

Now we construct our oracle QTM by describing its operation step by step. Immediately after **Step** x, we always state the states of the TM tape $|\sigma_x\rangle$ and of the oracle tape $|\tau_x\rangle$. Before the execution of the oracle QTM (**Step 0**), the state of the TM tape $|\sigma_0\rangle$ and the state of the oracle tape $|\tau_0\rangle$ are:

$$|\sigma_0\rangle = |1^n \circ 0^{m_n+1}\rangle \qquad |\tau_0\rangle = |0^{m_n}\rangle\,.$$

The oracle QTM needs to decide the first n qubits of $|\sigma_0\rangle$, which represent $|1^n\rangle$.

Step 1: on the input of $|\sigma_0\rangle$, the oracle QTM first counts the number of 1's in $|\sigma_0\rangle$ to gain the value of n. Then it applies $(HX)^{\otimes n} \otimes I^{\otimes(m_n+1)}$ to the TM tape and leaves the oracle tape as is. After this step,

$$|\sigma_1\rangle = (HX)^{\otimes n} \otimes I^{\otimes(m_n+1)} |\sigma_0\rangle = \left(\frac{1}{\sqrt{2}}\right)^n \sum_{y=0}^{2^n-1} |y_{(2)} \circ 0^{m_n+1}\rangle\,, \qquad |\tau_1\rangle = |\tau_0\rangle\,,$$

where $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ is the *Pauli-X* gate, $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ is the *Hadamard* gate and $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

Step 2: the oracle QTM applies ζ to $|\sigma_1\rangle$. Upon finishing this operation:

$$|\sigma_2\rangle = \zeta(|\sigma_1\rangle) = \left(\frac{1}{\sqrt{2}}\right)^n \sum_{y=0}^{2^n-1} |y_{(2)} \circ f'(y_{(2)}) \circ 0\rangle\,, \qquad |\tau_2\rangle = |\tau_1\rangle\,.$$

Step 3: the oracle QTM swaps the $(n+1)$th to the $(n+m_n)$th qubit on the TM tape with all the qubits on the oracle tape. Thus,

$$|\sigma_3\rangle = \left(\frac{1}{\sqrt{2}}\right)^n \sum_{y=0}^{2^n-1} |y_{(2)} \circ 0^{m_n} \circ 0\rangle\,, \qquad |\tau_3\rangle = \left(\frac{1}{\sqrt{2}}\right)^n |f'(y_{(2)})\rangle\,.$$

Step 4: the oracle QTM moves to q_{query} and queries the string on its tape to O', the language defined in Section 4. First, consider one of the computational basis states of $|\tau_3\rangle$. For some integer $y \in [1, p-1]$, if the result of the query of $f'(y_{(2)})$ to O' is positive, the oracle QTM moves to the state q_{yes} and changes the last bit on the TM tape to 1. Otherwise, if the result of the query of $f'(y_{(2)})$ to O' is negative, the oracle QTM moves to the state q_{no} and changes the last bit on the TM tape to 0. In the quantum situation, the last qubit of $|\sigma_4\rangle$ is possibly in the state that it can collapse to 0 or 1 upon measurement. Thus,

$$|\sigma_4\rangle = \left(\frac{1}{\sqrt{2}}\right)^n \sum_{y=0}^{2^n-1} |y_{(2)} \circ 0^{m_n} \circ \chi_y\rangle\,, \qquad |\tau_4\rangle = |\tau_3\rangle\,,$$

where $\chi_y \in \{0, 1\}$. The value of χ_y is dependent on its corresponding y and the result of the query.

Step 5: measure the last qubit of $|\sigma_4\rangle$. If the measurement reveals 1, the oracle QTM is certain that 1^n is a member of the language L; otherwise, the oracle QTM is able to claim that its member is not a member of the language L with bounded possibility of correctness. We prove this in the next section. The TM tape collapses to one of its computational basis states, while the oracle tape remains the same.

$$|\sigma_5\rangle = y_{(2)} \circ 0^{m_n} \circ \chi_y \qquad |\tau_5\rangle = |\tau_4\rangle\,.$$

After clearing out the contents on both tapes, we repeat Step 1 to Step 5 four times to improve the probability of being correct, mentioned in Step 5, to over 2/3.

5.2 Example

We elaborate on this process with an example. We know that $11 \in L$ and $111 \notin L$, because 2 is a primitive root of 3 but not 7. Suppose the string to decide is 111, then at **Step 0**:

$$|\sigma_0\rangle = |111 \circ 0^{m_3+1}\rangle \qquad |\tau_0\rangle = |0^{m_3}\rangle\,.$$

Step 1: after counting the number of 1's in the input, the oracle QTM applies $(HX)^{\otimes n} \otimes I^{\otimes(m_3+1)}$ to $|\sigma_0\rangle$:

$$\begin{aligned}|\sigma_1\rangle =& (HX)^{\otimes 3} \otimes I^{\otimes(m_3+1)} |\sigma_0\rangle \\ =& \left(\frac{1}{\sqrt{2}}\right)^3 (|0_{(2)} \circ 0^{m_3+1}\rangle + |1_{(2)} \circ 0^{m_3+1}\rangle + |2_{(2)} \circ 0^{m_3+1}\rangle + |3_{(2)} \circ 0^{m_3+1}\rangle + \\ & |4_{(2)} \circ 0^{m_3+1}\rangle + |5_{(2)} \circ 0^{m_3+1}\rangle + |6_{(2)} \circ 0^{m_3+1}\rangle + |7_{(2)} \circ 0^{m_3+1}\rangle), \qquad |\tau_1\rangle = |\tau_0\rangle\,.\end{aligned}$$

Step 2: the oracle QTM applies ζ to $|\sigma_1\rangle$. Upon finishing this operation:

$$\begin{aligned}|\sigma_2\rangle =& \zeta(|\sigma_1\rangle) \\ =& \left(\frac{1}{\sqrt{2}}\right)^3 (|0_{(2)} \circ f'(0_{(2)}) \circ 0\rangle + |1_{(2)} \circ f'(1_{(2)}) \circ 0\rangle + |2_{(2)} \circ f'(2_{(2)}) \circ 0\rangle + |3_{(2)} \circ f'(3_{(2)}) \circ 0\rangle + \\ & |4_{(2)} \circ f'(4_{(2)}) \circ 0\rangle + |5_{(2)} \circ f'(5_{(2)}) \circ 0\rangle + |6_{(2)} \circ f'(6_{(2)}) \circ 0\rangle + |7_{(2)} \circ f'(7_{(2)}) \circ 0\rangle) \qquad |\tau_2\rangle = |\tau_1\rangle\,.\end{aligned}$$

Step 3: the oracle QTM swaps the 4th to the $(3 + m_3)$th qubit on the TM tape with all of the qubits on the oracle tape. Thus,

$$\begin{aligned}|\sigma_3\rangle =& \left(\frac{1}{\sqrt{2}}\right)^3 (|0_{(2)} \circ 0^{m_3+1}\rangle + |1_{(2)} \circ 0^{m_3+1}\rangle + |2_{(2)} \circ 0^{m_3+1}\rangle + |3_{(2)} \circ 0^{m_3+1}\rangle + \\ & |4_{(2)} \circ 0^{m_3+1}\rangle + |5_{(2)} \circ 0^{m_3+1}\rangle + |6_{(2)} \circ 0^{m_3+1}\rangle + |7_{(2)} \circ 0^{m_3+1}\rangle) \\ |\tau_3\rangle =& \left(\frac{1}{\sqrt{2}}\right)^3 (|f'(0_{(2)})\rangle + |f'(1_{(2)})\rangle + |f'(2_{(2)})\rangle + |f'(3_{(2)})\rangle + |f'(4_{(2)})\rangle + |f'(5_{(2)})\rangle + |f'(6_{(2)})\rangle + |f'(7_{(2)})\rangle)\end{aligned}$$

To see the content on the tape of the oracle QTM after the next step, consider $2^s \bmod 7$ for every positive integer s.

$$2^s \bmod 7 = \begin{cases} 2 & (s=1) \\ 4 & (s=2) \\ 1 & (s=3) \end{cases} \qquad 2^s \bmod 7 = \begin{cases} 2 & (s=4) \\ 4 & (s=5) \\ 1 & (s=6) \end{cases} \qquad \ldots$$

Intuitively, we can observe that $2^s \bmod 7 \in \{2, 4, 1\}$ for every positive integer s. In fact, this is true and the reason will be given later in this section. Thus, if $y \in \{2, 4, 1\}$, there is some positive integer s such that $y = 2^s \bmod 7$, $y_{(2)} \in D$ by the definition of D in Section 4 and $f'(y_{(2)}) \in O'$. Conversely, if $y \in \{3, 5, 6\}$, there is not any positive integer s such that $y = 2^s \bmod 7$, $y_{(2)} \notin D$ and $f'(y_{(2)}) \notin O'$. Lastly, if $y = 0$ or

$y \geq 7$, $y_{(2)} \in D$ and $f'(y_{(2)}) \in O'$. Recall that in Step 4, the oracle QTM does nothing to the last qubit of $|\sigma_3\rangle$ if $f'(y_{(2)}) \in O$ and changes it to $|1\rangle$ otherwise. Thus, after **Step 4**, the TM tape and the oracle tape are in the states:

$$|\sigma_4\rangle = \left(\frac{1}{\sqrt{2}}\right)^3 (|0_{(2)} \circ 0^{m_3} \circ 0\rangle + |1_{(2)} \circ 0^{m_3} \circ 0\rangle + |2_{(2)} \circ 0^{m_3} \circ 0\rangle + |3_{(2)} \circ 0^{m_3} \circ 1\rangle + |4_{(2)} \circ 0^{m_3} \circ 0\rangle + |5_{(2)} \circ 0^{m_3} \circ 1\rangle + |6_{(2)} \circ 0^{m_3} \circ 1\rangle + |7_{(2)} \circ 0^{m_3} \circ 0\rangle) \qquad |\tau_4\rangle = |\tau_3\rangle .$$

Similarly, if 11 inputs to the oracle QTM, the tape after the query is at the state

$$|\psi_4\rangle = \left(\frac{1}{\sqrt{2}}\right)^3 (|0_{(2)} \circ 0^{m_2} \circ 0\rangle + |1_{(2)} \circ 0^{m_2} \circ 0\rangle + |2_{(2)} \circ 0^{m_2} \circ 0\rangle + |3_{(2)} \circ 0^{m_2} \circ 0\rangle .$$

Observe the difference of the last qubit of $|\sigma_4\rangle$ and $|\psi_4\rangle$. The result of the measurement of the last bit of $|\psi_4\rangle$ is always 0 and that of $|\sigma_4\rangle$ is either 0 or 1. Hence, if 1 is measured for some urary encoding of n, it is certain that the urary encoding of n is not in L. Otherwise, we are not certain of the result, but we claim that the urary encoding of n is not in L with a bounded probability of making a mistake, which we will prove in the next section.

Step 5: measure the last qubit of $|\sigma_4\rangle$ and decide according to the criteria stated above. Assume $|\sigma_4\rangle$ collapses to the state where $y = 3$, then

$$|\sigma_5\rangle = 3_{(2)} \circ 0^{m_3} \circ 1 \qquad |\tau_5\rangle = |\tau_4\rangle .$$

Finally, repeat Step 1 to Step 5 four times.

6 Bounded Probability of Making a Mistake

To see why the probability of making a mistake is bounded, first consider the case where 1 is measured and the oracle QTM accepts the input. If 1 is measured, it means for some integer $y \in [1, 2^n - 1]$, $f'(y_{(2)}) \notin O'$ and $y_{(2)} \notin D$. Thus, for $y = 2^s \bmod p$ where p is the largest prime number smaller than $2^{|y_{(2)}|} - 1$, Shor's finds an incorrect solution for s. Equivalently, there is some y such that for any integer s, $y \neq 2^s \bmod p$, which qualifies the condition for the input 1^n to be in L. In the example given above, only the result of the measuring the last qubit of $|\sigma_4\rangle$ has the probability of being 1 and that of $|\psi_4\rangle$ does not. Thus, if 1 is measured for the last qubit of $|\sigma_4\rangle$, it is with certainty that $111 \in D$.

Now consider the case where 0 is measured and the oracle QTM rejects the input. The last qubit on the tape of the oracle QTM may be in a state that cannot collapse to 1, which means the input is not in D. For example, the last qubit of $|\psi_4\rangle$ is in this state and since $11 \notin D$, the decision of rejection is correct. Alternatively, the last qubit on the tape may be in a state that can either collapse to 0 or 1, which means the input is in D. For example, the last qubit of $|\sigma_4\rangle$ is in this state and since $111 \in D$, the oracle QTM makes a mistake by rejecting 111! From here, we prove that the probability of making a mistake is below 1/3. Notice that because a mistake can only be made when the oracle QTM rejects 1^n, all the discussions below are considering the rejection of 1^n.

Consider running the algorithm only once. Let S be the sequence generated by $2^s \bmod p$ for some prime $p \geq 3$ and $s = 1, 2, \ldots, p-1$ and let $U = \{u : u = 0 \text{ or } u \geq p\}$ and $R = \{r : r \in S\}$ and $T = \{1, 2, \ldots, 2^n - 1\} \setminus R \setminus U$ where n is the smallest value such that $2^n > p$. For every $t \in T$, there does not exist $s \in \mathbb{Z}^+$ such that $t = 2^s \bmod p$, so $t_{(2)} \notin D$ and $f'(t_{(2)}) \notin O'$, which causes the last qubit of the oracle tape to be $|0\rangle$. Similarly, for every $r \in R \cup U$, by the definition of D, $r_{(2)} \in D$ and $f'(r_{(2)}) \in O'$, which causes the last qubit of the oracle tape to be $|1\rangle$. Therefore, the state of the TM tape after Step 4 is

$$|\sigma_4\rangle = \left(\frac{1}{\sqrt{2}}\right)^n \left(\sum_{y \in R \cup U} |y_{(2)} \circ 0^{m_n} \circ 0\rangle + \sum_{y \in T} |y_{(2)} \circ 0^{m_n} \circ 1\rangle\right) .$$

Recall the situation that a mistake is made. If 0 is measured and $T \neq \varnothing$, the oracle QTM will reject the input 1^n while, in fact, it should accept. Therefore, the probability of making a mistake is equal to the probability that the measurement of the $(n+1)$th slot of the tape results in 0 when $T \neq \varnothing$, which is $(|R| + |U|)/2^n$. It is obvious that $|U| = 2^n - p + 1$, so the next step is to find $|R|$.

First, we prove that S is periodic and the period length k is the smallest positive integer such that $1 = 2^k \bmod p$ for some prime number p. By periodic, we mean for any positive integer i, $2^i \bmod p = 2^{i+k} \bmod p$. We start with 2^{i+k}:

$$\begin{aligned} 2^{i+k} &= (2^i \bmod p \cdot 2^k \bmod p) \bmod p \\ &= (2^i \bmod p) \bmod p \\ &= 2^i \bmod p. \end{aligned}$$

Therefore, S is periodic and the period length is k. Since R is the set of different numbers of S, $|R|$ is the period length of S and therefore $|R| = k$. Given that k divides $p-1$, there are a limited number of possibilities for the period length in S.

The highest possible value of k is where $k = p-1$. In this case, since every value in a period is different, there are $p-1$ different numbers in S. Thus, there is no integer $y \in [1, p-1]$ such that $y \neq 2^s \bmod p$ for any s. Equivalently, 2 is a primitive root of p, so the oracle QTM does not make a mistake when it rejects 1^n.

The second-largest possible value of k is when $k = (p-1)/2$. This is because since the prime number p is odd, $p-1$ must be even, 2 and $(p-1)/2$ are both factors of k. In this case, $|R| = (p-1)/2$. Therefore, the probability of making a mistake is

$$\frac{\frac{p-1}{2} + 2^n - p + 1}{2^n}.$$

Although it is impossible to further discuss the next largest possible values, as it is uncertain what other factors $p-1$ may have, there is no need to. Recall that we are finding an upper bound of making a mistake. As k becomes smaller, $|R|$ and hence the probability of making a mistake becomes smaller. Since $p \leq 2^n - 1$, the upper bound of the probability of making a mistake when running the algorithm once is

$$\frac{2^n - 2^{n-2}}{2^n} = \frac{3}{4}.$$

Running the algorithm at least 4 times allows the upper bound of making a mistake to be $\left(\frac{3}{4}\right)^4 = \frac{81}{256} < \frac{1}{3}$. We have proved that the primitive root problem is in $\mathbf{BQP}^{\mathbf{BQP}}$ and hence in $\mathbf{BQP}$ by Theorem 2.1.

7 Conclusion

In this paper, we have proved that the primitive root problem that examines, given n, whether some positive integer g is a primitive root of p where p is the largest prime number smaller than 2^n, is in $\mathbf{BQP}$.

This is proved with an oracle QTM. We first define a language D in $\mathbf{BQP}$ which can be reduced to some $\mathbf{BQP}$-complete language O with some reduction function f. O is used as the oracle input to the oracle QTM. On some input string of length n to the oracle QTM, it makes a state $|\psi\rangle$ that is the superposition of all the values in $[1, p-1]$ where p is the largest prime number smaller than 2^n and queries O' with $|f'(\psi)\rangle$. The result of the query is therefore also in a superposed state which may collapse to 0 or 1 after measurement. If the result of the measurement is 1, the oracle QTM accepts the input and otherwise, it rejects. All of the operations above runs in polynomial time. Repeat this process 4 times. We then prove that the probability for the oracle QTM to make the correct decision is over 2/3, so the primitive root problem is in $\mathbf{BQP}^{\mathbf{BQP}}$. By Theorem 2.1, the primitive root problem is in $\mathbf{BQP} = \mathbf{BQP}^{\mathbf{BQP}}$.

This result is significant for several reasons. First of all, it provides an efficient quantum solution to a problem to which no efficient classical solution is yet found. It is another piece of evidence of the potential power of quantum computers besides the Deustch-Josza algorithm, Simon's algorithm and Shor's algorithm. Second,unlike these algorithms, our algorithm uses the result of an efficient quantum algorithm, Shor's algorithm, to solve a problem assumed to be harder than the discrete logarithm problem. In specific, they are both in $\mathbf{BQP}$. Possible future work may include expanding existing algorithms to solve problems harder than the problems solved by these existing algorithms.

Additionally, we should notice the simplicity of the algorithm. The algorithm queries the oracle with all possible values and measures the result directly. Due to the fact that the sequence created by $2^s \bmod p$ for integer $s \in [1, p-1]$ is periodic and its period length divides $p-1$, a direct measurement to the result of the query yields a bounded possibility of making mistakes when making decisions accordingly. In the future, we would like to explore other mathematical properties that allow a bounded probability of making a mistake when making decisions based on a direct measurement of a quantum state.

References

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[2] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[3] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22:563–591, 05 1980.

[4] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997.

[5] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[6] André Berthiaume and Gilles Brassard. Oracle quantum computing. *Journal of Modern Optics*, 41(12):2521–2535, 1994.

[7] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

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

[9] Daniel R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997.

[10] William Stein. *Elementary Number Theory: Primes, Congruences, and Secrets: A Computational Approach*. Undergraduate Texts in Mathematics. Springer New York, 2008.

[11] Collin Williams. *Explorations in Quantum Computing*. Texts in Computer Science. Springer London, 2010.

[12] Pawel Wocjan and Shengyu Zhang. Several natural BQP-complete problems. *arXiv preprint quant-ph/0606179 (2006)*, Jul. 2006.