Quantum Complexity Theory

Nihar Shah

May 19, 2025

Quantum Computing and Quantum Information: Theory and Practice

Nihar Shah

Guide: Professor Phani Motamarri

भारतीय विज्ञान संस्थान

Department of Computational and Data Science
Indian Institute of Science

Abstract

The following content consists of the work done by me as a part of my Master's Thesis at Indian Institute of Science, Bengaluru. The work is done under the guidance of Prof. Phani Motamarri. This work includes notes made during the 1 year work from June 2024 to August 2025 and is based on the lectures, papers, books, and other resources that I have referred to during the course of my work. To a reader who is not at all familiar with Quantum Computing, this work will serve as a good starting point. I suggest starting with the Appendix to get a good grasp of the math used in the entire text. The work is divided into parts: The first part is the introduction to Quantum Computing, the second part is Quantum Linear Algebra, the third part is on Quantum Information Theory and at last the fourth part is on Quantum Error Correction. Along with the references, I have also included the code snippets that I have written during the course of my work. The references are included at the end of the document. I hope that this text helps the reader to get a good grasp on Quantum Computing. Enjoy reading!

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor, Professor Phani Motamarri, for the continuous support of my study and related, for his patience, and immense knowledge. His guidance helped me through the time of research and writing of this thesis.

Besides my advisor, I am also grateful to the faculty members and staff at the Department of Computational and Data Science of Indian Institute of Science for their assistance and support. Special thanks to MATRIX lab for providing a stimulating and collaborative research environment.

Last but not the least, I would like to thank my family: my parents, for supporting me spiritually throughout writing this thesis and my life in general.

Contents

Chapter 1

Computational Complexity Theory

1.1 Introduction

The modern incarnation of computer science was announced by the great mathematician Alan Turing in a remarkable 1936 paper. Turing developed in detail an abstract notion of what we would no call a programmable computer, a model for computation now known as *Turing machine*. Turing showed that there is a *Universal Turing Machine* that can be used to simulate any other *Turing Machine*. He claimed that Universal Turing Machine can be used to simulate any other Truing machine. The Universal Turing Machine completely captures what it means to perform a task by algorithmic means. That is, if an algorithm can be performed on any given piece of hardware (say, a modern computer), then there is an equivalent algorithm for a Universal Turing Machine which performs exactly the same task as the algorithm running on the personal computer.This assertion is known as **Church-Turing Hypothesis**, asserts the equivalence between the physical concept of what class of algorithms can be performed on some physical device with rigorous mathematical concept of a Turing Machine.

Not long after Turing's paper, John von Neumann developed a simple theoretical model for how to put together in a practical fashion all the components necessary for a computer to be fully capable as a Universal Turing Machine. Hardware development took off when John Bradeen, Walter Trattain, and Will Shockley developed the transistor in 1947. Computer hardware has grown n power at an amazing speed, so much so that their growth was codified by Gordon Moore in 1965 in what has come to be known as Moore's law which states that computer power will double for constant cost roughly once every two years.

Moore's law was approximately held true in the decades since the 1960s. Most

observers expect that this dream run will end in some time during the first two decades of the twenty-first century. Conventional approaches to the fabrication of computer technology are beginning to run up against fundamental difficulties of size. Quantum effects are beginning to interfere in the functioning of electronic devices as they are made smaller and smaller.

On possible solution to the problem posed by the eventual failure of Moore's law is to move to a different computing paradigm. One such paradigm is provided by the theory of quantum computation., which is based on the idea of using quantum mechanics to perform computations, instead of classical physics. It turns out that while an ordinary computer can be used to simulate a quantum computer it appears to be impossible to perform the simulation in an efficient fashion. Thus quantum computers offer an essential speed advantage over classical computers. This speed advantage is so significant that many researchers believe that no conceivable amount of progress in classical computation would be able to overcome the gap between the power of a classical computer and the power of a quantum computer.

1.1.1 Efficient vs Inefficient

What do we mean by 'efficient' versus 'inefficient' simulations of a quantum computer? The idea of efficient and inefficient algorithms was made mathematically precise by the field of computational complexity. Roughly speaking, an efficient algorithm is one which runs in time polynomial in the size of the problem solved. In contrast, an inefficient algorithm requires super-polynomial (typically exponential) time.

Turing machine model was a least as powerful as any other model of computation, in the sense that a problem which can be solved efficiently in some model of computation could also be solved efficiently in the Turing machine model, by using the Turing machine to simulate the other model of computation. This leads us to the Church-Turing Thesis

Any algorithmic process can be simulated efficiently using a Turing machine

The key strengthening in the strong Church-Turing Thesis is the word efficiently. If the strong Church-Turing thesis is correct, then it implies that no matter what type of machine we use to perform our algorithms , that machine can be simulated efficiently using a standard Turing machine. This implies that for the purposes of analyzing whether a given computational task can be accomplished efficiently, we may restrict ourselves to the analysis of the Turing model of computation.

1.1.2 The Challenges

One class of challenges to the strong Church-Truing thesis comes from the field of analog computation. In they years since Turing, many different teams of researchers have noticed that certain types of analog computers can efficiently solve problems believed to have no efficient solution on a Turing Machine. At first glance these analog computers appear to violate the strong form of the Church-Turing thesis. Unfortunately for analog computation, it turns out that when realistic assumptions about the presence of noise in analog computers are made, their power disappears in all known instances; they cannot efficiently solve problem which are not efficiently solvable on a Turing machine. This lesson - that the effects of realistic noise must be taken into account in evaluating the efficiency of a computational model - was one of the great early challenges of quantum computation and quantum information, a challenge successfully met by the development of a theory of quantum error-correcting codes and fault-tolerant quantum computation. Thus, unlike analog computation, quantum computation can in principle tolerate a finite amount of noise and still retain its computational advantages.

The first major challenge to the Strong Church-Turing thesis arose in the mid 1970s when Robert Solovay and Volker Strasses showed that it is possible to test whether an integer is prime of composite using a randomized algorithm. That is the Solvay-Strassen test for primality used randomness as an essential part of the algorithm. The algorithm did not determine whether a given integer was prime or composite with certainty. By repaeating the Solovay=Strassen test a few times it is possible to determine with near certainty whether a number is prime of composite. The solovay-Strassen test was of special significance at the time it was proposed as no deterministic test for primality was then known, nor is one known at the time of this writing. Thus, it seemed as though computer with access to random number generator would be able to efficiently perform computational tasks with no efficient solution on a conventional deterministic Turing machine. this discovery inspired a search for other randomized algorithms which has paid off handsomely, with the field blossoming into a thriving area of research.

Randomized algorithms pose a challenge to the strong Church-Turing Thesis, suggesting that here are efficiently solvable problems which, nevertheless, cannot be efficiently solved on a deterministic Turing machine. This challenge appears to be easily resolved by a simple modification of the strong Church-Turing thesis:

Any algorithmic process can be simulated efficiently using a probabilistic Turing machine

This ad hoc modification of the strong Church-Turing thesis should leave you

feeling rather queasy. Might it not turn out at some later date that yet another model of computation allows one to efficiently solve problems that are not efficiently solvable within Turing's model of computation? Is there any way we can find a single model of computation which is guaranteed to be able to efficiently simulate any other model of computation?

Motivated by this question, in 1985 David Deutsch asked whether the laws of physics could be use to derive an even stronger version of the Church-Turing thesis, Instead of adoption ad hoc hypotheses, Deutsch looked to physical theory to provide a foundation for the Church-Turing thesis that would be as secure as the status of that physical theory. In particular, Deutsch attempted to define a computational device that would be capable of effectively simulating an arbitrary physical system. These devices, quantum analogues of the machines defined forty-nine years earlier by Turning, led ultimately to the modern conception of a quantum computer.

At the time of writing it is not clear whether Deutsch's notion of a Universal Quantum Computer is sufficient to efficiently simulate an arbitrary Physical system. Proving or refuting this conjecture is one of the great problems of the field of quantum computation and quantum information. It is possible, for example, that some effect of quantum field theory or an even more esoteric effect based in string theory, quantum gravity or some other physical theory may take us beyond Deutsch's Universal Quantum computer, giving us a still more powerful model for computation. At this stage, we simply don't know.

What Deutsch's model of a quantum computer did enable was a challenge to the Strong form of the Church-Turing thesis. Deutsch asked whether it is possible for a quantum computer to efficiently solve computational problems which have no efficient solution on a classical computer, even a probabilistic Turing machine. He then constructed a simple example suggesting that, indeed, quantum computers might have computational powers exceeding those of classical computers.

This remarkable first step taken by Deutsch was improved in the subsequent decade by many people, culminating in Peter Shor's 1994 demonstration that two enormously important problems - the problem of finding the prime factors of an integer, and the so-called 'discrete logarithm' problem - could be solved efficiently on a quantum computer. This attracted widespread interest because these two problems were and still are widely believed to have no efficient solution on a classical computer. Shor's result are a powerful indication that quantum computers are more powerful than Turing machines, even probabilistic Turing machines. Further evidence for the power of quantum computers cam in 1995 when Lov Grover showed that another important problem - the problem of conducting a search through some unstructured search space - could also be sped up on a quantum computer. While Grover's al-

gorithm did not prove as spectacular speed up as Shor's algorithms, the widespread applicability has excited considerable interest in grover's algorithm.

At about the same time as Shor's and Grover's algorithm were discovered, many people were developing an idea Richard Feynman had suggest in 1982. Feynman had pointed out that there seemed to be essential difficulties in simulating quantum mechanical systems on classical computers, and suggested that building computers based on the principles of quantum mechanics would allow us to avoid those difficulties. In the 1990s several teams of researchers began fleshing this idea out, showing that it is indeed possible to use quantum computers to efficiently simulate systems that have no known efficient simulation on a classical computers. It is likely that one of the major applications of quantum computers in the future will be performing simulations of quantum mechanical systems too difficult to simulate on a classical computer, a problem with profound scientific and technological implications.

1.1.3 What other problems?

What other problems can quantum computers solve more quickly than classical computer? The short answer is that we don't know. Coming up with good quantum algorithms seems to be hard. It is because algorithm design for quantum computers is hard because designers face two difficult problems not aced in the constructing of algorithms for classical computer. First, our human intuition is rooted in the classical world. If we use that intuition as an aid to the construction of algorithms, then the algorithmic ideas we come up with will be classical ideas. To design good quantum algorithms one must turn off one's classical intuition for at least part of the design process, using truly quantum effects to achieve the desired algorithmic end. Second, to be truly interesting it is not enough to design an algorithm that is merely quantum mechanical. The algorithm must be better than any existing classical algorithm. Thus! it is possible that one may find an algorithm which makes use of truly quantum aspects of quantum mechanics, that is nevertheless not of widespread interest because classical algorithms with comparable performance characteristics exist. The combination of these two problems makes the construction of new quantum algorithms a challenging problem for the future.

Even more broadly, we can ask if there are any generalizations we can make about the power of quantum computers versus classical computers. What is it that makes quantum computers more powerful than classical computers. What is it that makes quantum computers more powerful than classical computers. Assuming that this is indeed the case? What class of problems can be solved efficiently on a quantum computer, and how does that class compare to the class of problems that can be

solved efficiently on a classical computer? One of the most exciting things about quantum computation and quantum information is how little is known about the answers to these questions! It is a great challenge for the future to understand these questions better.

1.2 The power of quantum computation

How powerful are quantum computers? What gives them their power? Nobody yet knows the answers to these questions, despite the suspicions fostered by examples such as factoring, which strongly suggests that quantum computers are more powerful than classical computers. It is still possible that quantum computers are no more powerful than classical computers, in the sense that any problem which can be efficiently solved on a quantum computer can also be efficiently solved on a classical computer. On the other hand, it may eventually be proved that quantum computers are much more powerful than classical computers.

Definition 1.2.1. (Algorithm): An algorithm is a precise recipe for performing some task. For example, adding two numbers

The fundamental question we are trying to address in the study of algorithms is: what resources are required to perform a given computational task. This question splits up into two parts:

1. What computational tasks are possible, preferably by giving explicit algorithms for solving specific problems. For example, algorithms that can quickly sort a list of numbers into ascending order.

2. To demonstrate limitations on what computational tasks may be accomplished. For example, lower bounds can be given for the number of operations that must be performed by any algorithm which sorts a list of numbers into ascending order.

Ideally, these two tasks - the finding of algorithms for solving computational problems, and proving limitations on our ability to solve computational problems, would dovetail perfectly. In practice, a significant gap often exists between the best techniques known for solving a computational problems, and the most stringent limitations known on the solution.

1.2.1 Why study classical computer science?

1. Classical computer science provides a cast body of concepts and techniques which may be reused to great effect in quantum computation and quantum information. Many of the triumphs of the quantum computation and quantum information have come by combining existing ideas from computer science with novel ideas from quantum mechanics. For example, some of the fast algorithms for quantum computers are based upon the Fourier transform, a powerful tool utilized by many classical algorithms. Once it was realized that quantum computers can perform a type of Fourier transform much more quickly than classical computer this enabled the development of many important quantum algorithms.

2. Computer scientists have expended great effort understanding what resources are required to perform a given computational tasks on a classical computer. These results an be used as the basis for a comparison with quantum computation and quantum information. For example, the problem of finding prime factors of a given number. On a classical computer this problem is believed to have no 'efficient' solution, here 'efficient' has a meaning to be explained later. An efficient solution to this problem is known for quantum computers. Thus, for the task of finding prime factors there appears to be a gap between what's possible on a classical computer and what is possible on a quantum computer. Thus a gap may exist for a wider class of computation problem than merely finding of prime factors. By studying this specific problem further, it may be possible to discern features of the problem which make it more tractable on a quantum computer than on a classical computer, and then act on these insights to find interesting quantum algorithms for the solution to other problems.

3. These is learning to think like a computer scientist. Computer scientists think in a rather different style than does a physicist. Thus for deep understanding of quantum computation and quantum information one must learn to think like a computer scientist at least some of the time, they must instinctively known what problems, what techniques and mot importantly what problems are of greatest interest.

We now take a brief look at what is known about the power of quantum computation.

1.2.2 Computational Complexity Theory

It is the subject of classifying the difficulty of various computational problems, both classical and quantum, and to understand the power of quantum computers we will first examine some general ideas from computational complexity. The most basic idea is that of a complexity class. A complexity classs can be thought of as a collection of computational problems, all of which share some common feature with respect to the computational resources needed to solve those problems.

Two of the most important complexity classes go by the names P and NP. Roughly speaking, P is the class of computational problems that can be solved quickly on a classical computer. NP is the class of problems which have solutions which can be quickly checked on a classical computer. To understand the distincition between P and NP, consider the problem of finding the prime factors of an integer, n. So far as is known there is no fast way of solving this problem on a classical computer, which suggests that the problem is not in P. On the other hand, if somebody tells you that some number p is a factor of n, then we can quickly tell that this is correct by dividing p into n, so factoring is a problem in NP.

It is clear that P is a subset of NP, since the ability to solve a problem implies the ability to check potential solutions. What is not clear is whether or not there are problems in NP that are not in P. Perhaps the most important unsolved problem in theoretical science is to determine whether these two classes are different:

$$P \neq NP$$

Most researchers believe that NP contains problems that are not in P. In particular there is an important subclass of the NP problems, the $NP-$complete problems, that are of especial importance for two reasons, First, there are thousands of problems, many highly important, that are known to be NP-complete. Second, any given NP-complete problem is in some sense 'at least as hard' as all other problems in NP. More precisely, an algorithm to solve a specific NP-complete problem can be adapted to solve any other problem in NP, with a small overhead. In particular, if $P \neq NP$, then it will follow that no NP-complete problem can be efficiently solved on a classical computer.

It is not known whether quantum computers can be used to quickly solve all the problems in NP, despite the fact that they can be used to solve problems - like factoring which are believed by many people to be in NP but not in P. (Note that factoring is not known to be NP-complete problem, otherwise we would already know how to efficiently solve all problems in NP using quantum computers). It would certainly be very exciting if it were possible to solve all the problems in NP

efficiently on quantum computer. There is a very interesting negative result known in this direction which rules out using a simple variant of quantum parallelism to solve all problems in NP. Specifically, one approach to the problem of solving problems in NP on a quantum computer is to try to use some form of quantum parallelism to search in parallel through all possible solutions to the problem. We will later show that no approach based upon such a search-based methodology can yield an efficient solution to all the problems in NP. While it is disappointing that this approach fails, it does not rule that some deeper structure exists in NP that will allow them all to be solved quickly using a quantum computer.

P and NP are just two of a plethora or complexity classes that have been defined. Another important complexity class is $PSPACE$. Roughly speaking, $PSPACE$ consists of those problems which can be solved using resources which are few in spatial size (that is, the computer is 'small'), but not necessarily in time ('long' computations are fine). $PSPACE$ is believed to be strictly greater than both P and NP although, again, this has never been proved. Finally, the complexity class BPP is the class of problems that can be solved using randomized algorithms in polynomial time, if a bounded probability of error (say $1/4$) is allowed in the solution to the problem. BPP is widely regarded as being, even more so than P, the class of problems which should be considered efficiently soluble on a classical computer. We have elected to concentrate here on P rather than BPP because P has been studied in more depth, however many similar ideas and conclusions arise in connection with BPP.

What of quantum complexity classes? We can define BQP to be the class of all computational problems which can be solved efficiently on a quantum computer, where a bounded probability of error is allowed. (strictly speaking this makes BQP more analogous to the classical complexity class BPP than to P, however we will ignore this subtlety for the purpose of the present discussion, and treat it as the analogue of P.) Exactly where BQP fits with respect to $P, NP, PSPACE$ is yet unknown. Therefore, BQP likes somewhere between P and $PSPACE$. An important implication is that if it is proved that quantum computers are strictly more powerful than classical computers, then it will follow that P is not equal to $PSPACE$. Proving this latter result has been attempted without success by many computer scientists, suggesting that it may be non-trivial to prove that quantum computers are more powerful than classical computers, despite much evidence in favor of this proposition.

It is clear that the theory of quantum computation poses interesting and significant challenges to the traditional notions of computation. What makes this an important challenges is that the theoretical model of quantum computation is be-

lieved to be experimentally realizable, because to the best of our knowledge - this theory is consistent with the way Nature works. If this were not so then quantum computation would be just another mathematical curiosity.

1.3 Models of Computation

The fundamental model for algorithms will be the *Turing machine*. This is an idealized computer, like a modern personal computer, but with a simpler set of basic instructions, and an idealized unbounded memory. The apparent simplicity of Turing machines is misleading; they are very powerful devices. They can be used to execute any algorithm whatsoever, even one running on an apparently much powerful computer.

What does it mean for an algorithm to perform some task. for example adding any two numbers, no matter how large those numbers are.

One learns Euclid's two thousand year old algorithm for finding greatest common divisor of two positive integers. In the late 1930s the fundamental notions of the modern theory of algorithms, and thus of computation, were introduced, by Alonzo Church, Alan Turing and others.This work arose in response to a profound challenge laid down by the great mathematician David Hilbert in the early part of the twentieth century. Hilbert asked whether or not there existed some algorithm which could be used, in principle, to solve all the problems of mathematics. Hilbert expect that the answer to this question, sometimes known as the *entscheidungsproblem* would be yes.

The answer to Hilbert's challenge turned out to be no: there is no algorithm to solve all mathematical problems. To prove this, Church and Turing had to solve the deep problem of capturing a mathematical definition what we mean when we use the intuitive concept of an algorithm. In doing s, they laid the foundations for the modern theory of algorithms, and consequently for the modern theory of computer science.

We now discuss the approach proposed by Turing, the Turing machines. Turing defined a class of machines known as Turing machines, in order to capture the notion of an algorithm to perform a computational task. We will then also discuss the circuit model of computation. Although these models of computation appear different ont he surface, it turns out that they are equivalent. Introducing different models of computation may yield different insights into the solutions of specific problems. Two or more ways of thinking about a concept are better than one.

1.3.1 Turing Machines

every two years. There are various taxonomies of computational problems according to their computational difficulty. The below shown are a few of these and how the power of quantum computer fits in within this classification.

Consider all binary functions over the set of all binary strings (of the form $f : \{0,1\}^* \to \{0,1\}$). The input is a binary string of some length n, where n can be anything. And the output is one bit. These are sometimes called decision problems since the answer is a binary decision, 0 or 1, yes or no, accept or reject. This is a common convention for reasons of simplicity. most problems that are not naturally decision problems can be reworked to be expressed as decision problems.

Definition 1.3.1. P (polynomial time) Solvable by $\mathcal{O}(n^c)$-size classical circuits (for some constant c). technically, we require uniform circuit families, one for each input size.

Definition 1.3.2. BPP (bounded-error probabilistic polynomial time Solvable by $\mathcal{O}(n^c)$-size probabilistic classical circuit whose worst-case error probability is $\leq \frac{1}{4}$.

Note that there is some arbitrariness in the error bound $\leq \frac{1}{4}$. Any polynomial-size circuit achieving this can be converted into another circuit whose error probability is $\leq \epsilon$ by repeating the process $\log(1/\epsilon)$ times and taking the majority value. using $\frac{1}{4}$ is simple, though any constant below $\frac{1}{2}$ would work.

In analogy too the classical class **BPP**, we will define **BQP** ("Bounded-error Quantum Polynomial time") as the class of languages that can efficiently be computed with success probability at least 2/3 by (a family of) quantum circuits whose size grows at most polynomially with the input length.

Definition 1.3.3. BQP (bounded-error quantum polynomial time) Solvable by $\mathcal{O}(n^c)$-size quantum circuits with worst-case error probability $\leq \frac{1}{4}$.

Definition 1.3.4. EXP (exponential time) Solvable by $\mathcal{O}(2^n)$-size classical circuits.

The following containment among these complexity classes are known:

$$\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{EXP}$$

Chapter 2

Loading classical data

The end-to-end quantum applications covered in this document have classical inputs and classical outputs, in the sense that the problem is specified by some set of classical data, and the solution to the problem should be a different set of classical data. In some cases, the input data is relatively small, and loading it into the algorithm does not contribute significantly to the cost of the algorithm. In other cases for example, "big data" problems within the areas of machine learning and finance - the dominant costs, both for classical and quantum algorithms, can be related to how the algorithms load and manipulate this large quantity of input data. Consequently, the availability of quantum speedups for these problems if often dependent on the ability to quickly and coherently access this data. The true cost of this access is the source of significant subtlety in many end-to-end quantum algorithms.

2.1 Quantum random access memory

Quantum random access memory (QRAM) is a construction that enables coherent access to classical data, such that multiple different elements in a classical database can be read in superposition. The ability to access large, unstructured classical data sets in this way is crucial to the speedup of certain quantum algorithms (for example, quantum machine learning based on quantum linear algebra). QRAM is commonly invoked in such cases as a way to circumvent data-input bottlenecks, i.e., situations where loading data could limit the end-to-end runtime of an algorithm. It remains an open question, however, whether a large-scale QRAM will ever be practical, casting doubt on quantum speedups that rely on QRAM. Note that, while here we focus on the more common use case of loading classical data with QRAM, certain QRAM architectures can be adapted to also load quantum data.

Consider a length N, unstructured classical data vector x, and denote the ith entry as x_i. Let the number of bits of x_i be denoted by d and let $D = 2^d$. Given an input quantum state $|\psi\rangle = \sum_{i=0}^{N-1} \sum_{j=0}^{D-1} \alpha_{ij} |i\rangle_A |j\rangle_B$, QRAM is defined as a unitary operation Q with the action,

$$Q|\psi\rangle = Q \sum_{i=0}^{N-1} \sum_{j=0}^{D-1} \alpha_{ij} |i\rangle_A |j\rangle_B = \sum_{i=0}^{N-1} \sum_{j=0}^{D-1} \alpha_{ij} |i\rangle_A |j \oplus x_i\rangle_B$$

Here, A is a $\log_2(N)$ -qubit register, and B is a d-qubit register. Note that the unitary Q can also be understood as an oracle (or a black box) providing access to x, as $Q(\sum_i \alpha_i |i\rangle |0\rangle) = \sum_i \alpha_i |i\rangle |x_i\rangle$.

Let T_Q denote the time it takes to implement the operation Q, where T_Q can be measured in circuit depth, total gate cost, T gate cost, etc.., depending on the context. Algorithms that rely on QRAM to claim exponential speedups over their classical counterparts frequently assume $T_Q = polylog(N)$.

2.1.1 Resource cost

The QRAM operation Q can be implemented as a quantum circuit that uses $\mathcal{O}(N)$ ancillary qubits and $\mathcal{O}(N)$ gates. Assuming gates acting on disjoint qubits can be parallelized, the depth of the circuit is only $T_Q = \mathcal{O}(\log(N))$. Explciit circuits can be found in e.g. The number of ancillary qubits can be reduced at teh price of increased circuit depth; circuit implementing Q can be constructed using $\mathcal{O}(N/M)$ ancillary qubits and depth $\mathcal{O}(M \log(N))$, where $M \in [1, N]$ see example

Bibliography