A Compare between Shor's Quantum Factoring Algorithm and General Number Field Sieve

Shah Muhammad Hamdi, Syed Tauhid Zuhori, Firoz Mahmud, Biprodip Pal
Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology
Rajshahi, Bangladesh
e-mail: tohamdi@gmail.com, tauhid.ruet@yahoo.com, fmahmud.ruet@gmail.com, biprodip.cse@gmail.com

Abstract— **Factoring large integers has been one of the most difficult problems in the history of mathematics and computer science. There was no efficient solution of this problem until Shor's algorithm emerged. Shor's algorithm is a polynomial time factoring algorithm which works on a quantum computer. Quantum computing is a new paradigm of computing that uses quantum mechanical phenomena in solving problems. The computers we are using right now are called classical computer. The most efficient classical factoring algorithm is General Number Field Sieve (GNFS). GNFS also cannot factor integers in polynomial time. In this paper, we compared these two algorithms in factoring integer in a standalone system.**

Keywords— Factorization; Quantum Computing; Shor's Algorithm; GNFS

I. Introduction

There is a conjecture that "*Integer factoring is computationally much harder than integer multiplication. In other words, while there are obviously many polynomial time algorithms for integer multiplication, there are no polynomial time algorithms for integer factoring. i.e., integer factoring computationally requires super-polynomial time* [1]" This statement was true; but only for classical computation. Peter Shor, in 1994, challenged this conjecture by devising a polynomial time algorithm that run on a quantum computer. On the other hand, GNFS is the best known factoring algorithm so far. In this paper, we have shown the factoring of a composite number of 100 decimal digits using GNFS. Then we compared the result against Shor's algorithm. We also covered all the steps in detail of Shor's algorithm. But, as Shor's algorithm is a quantum algorithm, we present some basic concepts of quantum computer and quantum computing. We also covered some historical backgrounds of Shor's algorithm and GNFS.

II. Background of Quantum Computing and Shor's Algorithm

Quantum computer is a computer that follows the laws of quantum mechanics. On the other hand, classical computers are the computers we are using right now. Classical computers follow the laws of classical physics. Some historical landmarks of quantum computing are presented below.

In 1980, Paul Benioff offered a classical Turing machine which used quantum mechanics in its workings, thus showing that theoretically a quantum computer was at least as powerful as a classical computer [2].

In 1982, physicist Richard Feynman observed that that certain quantum mechanical effects cannot be simulated efficiently on a classical computer [3]. This observation led to speculation that perhaps computation in general could be done more efficiently if it made use of these quantum effects.

The unusual power of quantum computation was not really anticipated until 1985 when David Deutsch published a crucial theoretical paper in which he described a universal quantum computer [4]. 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. Probabilistic Turing machine is a Turing machine which is capable of making a random choice.

David Deutsch and Richard Jozsa showed in a paper in 1992 that there was an algorithm that could be run in poly-log time on a quantum computer, but required linear time on a deterministic Turing machine [5]. This may have been the first example of a quantum computer being shown to be exponentially faster than a deterministic Turing machine. Unfortunately for the quantum computer, the problem could also be solved in poly-log time in a probabilistic Turing machine.

In 1992, Andre Berthiaume proved that P is subset of QP [6], where P is the complexity class of those classical algorithms whose worst case running time is some polynomial function in the size of the input and QP corresponds to problems which can be solved in polynomial time in worst case by a quantum computer. So with regards to tractability a quantum computer is in some sense more powerful than any classical computer.

By the early nineties it was known that a quantum computer could be faster than any classical computer for certain problems [7]. Nonetheless these observations were largely driven by academic curiosity. There was not much motive for people to spend lots of money or time trying to build a quantum computer.

This changed in 1994 when Peter Shor, a scientist working for Bell Labs, gave a positive answer to Deutsch's question asked in 1985, by demonstrating that two very important problems – the problem of finding the prime factors of large integer and the discrete logarithm problem can be solved efficiently in quantum computer [8][9]. These two problems have no polynomial time solutions in classical computers. Shor's results are a powerful indication that quantum computers are more powerful than Turing machines, even probabilistic Turing machines. His discovery drew great attention to the field of quantum computing.

Shor's algorithm for factoring large integers is one of the most prominent algorithms in the field of quantum computing. The capacity of Shor's algorithm would be able to break widely used cryptographic codes such as RSA public key system. Experimental realization of Shor's algorithm has been a very important goal in the research of quantum computing and quantum information processing. However, due to substantial resource requirement, to date, there have been only a few small scale demonstrations. These experimental realizations used different methodologies like Nuclear Magnetic Resonance (NMR) [18], quantum entanglement [19], photonic qubit [20], qubit recycling [21] and Josephon phase [22]. The research is not only confined in experimental realizations. The research is also exploring for better optimization of the algorithm by reducing number of qubit and quantum gates [16] [17].

III. Background of Classical Factoring Efforts and GNFS

A lot of efforts were put on to solve prime factorization problem efficiently in last few centuries by numerous mathematicians and scientists. Before Peter Shor introduced his prime factorization algorithm using quantum computation, all the factoring algorithms, which were designed to run on classical computers could not solve this problem in less than super-polynomial time with respect to the number of digits in the input number. In this section, we are going to discuss some significant efforts of classical computing in solving prime factorization problem.

In the middle part of this century, computational issues seemed to be out of fashion. In most books the problem of factoring big numbers was largely ignored, since it was considered trivial. A few researchers ignored the fashions of the time and continued to try to find fast ways to factor. To these few it was a basic and fundamental problem, one that should not be shunted to the side [10].

But time changes. In the last few decades we have seen the advent of accessible and fast computing power, and we have seen the rise of cryptographic systems that base their security on our supposed inability to factor quickly. For this reason, researchers began to more concentrate on this problem, recognizing it not only as a benchmark for the security of cryptographic systems, but for computing itself.

In 1970 it was barely possible to factor "hard" 20-digit numbers. In 1980, in the heyday of the Brillhart-Morrison continued fraction factoring algorithm, factoring of 50-digit numbers was becoming commonplace. In 1990, the quadratic sieve factoring algorithm had doubled the length of the numbers that could be factored, the record having 116 digits [10].

By 1994 the quadratic sieve had factored the famous 129-digit RSA challenge number that had been estimated in Martin Gardner's 1976 Scientific American column to be safe for 40 quadrillion years (though other estimates around then were more modest). But the quadratic sieve is no longer the champion. It was replaced by John Pollard's number field sieve (also known as general number field sieve or GNFS) in the spring of 1996, when that method successfully split a 130-digit RSA challenge number in about 15% of the time the quadratic sieve would have taken [10].

IV. Quantum Computing

Before moving to the details of Shor's algorithm we discuss some concepts about quantum computing. This discussion includes Bra-ket notation, k-levels quantum system and qubit [11].

A. Bra-ket notation

A special notation, named ket notation is used to represent quantum state. The quantum state of a particle is represented by $|\Psi>$, which is read as "ket-shai". This notation came from the physicist Paul Dirac, who wanted a concise shorthand way of writing formulas that occur in quantum physics [11].

B. k- level quantum system

Let's consider a system of k distinguishable states. We can think of a Hydrogen atom. A Hydrogen atom has one electron. Let, this electron is allowed to be in one of a discrete set of energy levels, starting with the ground state, the first excited state, the second excited state and so on [11]. If we assume a suitable upper bound on the total energy, then the electron is restricted to being in one of k different energy levels – the ground state or one of k-1 excited state.

Now, if we denote ground state $|0>$ and successive excited states $|1>$, $|2>$, $|3>$, … , $|k\text{-}1>$, then according to the superposition principle, the quantum state of the electron is,

$$|\Psi> = \alpha_0\,|0> + \alpha_1\,|1> + \ldots + \alpha_{k-1}\,|k\text{-}1> \qquad (1)$$

where $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{k-1}$ are complex numbers normalized so that $\sum_{j=0}^{k-1} |\alpha_j|^2 = 1$.

C. Qubit

Qubit or quantum bit is the unit of quantum information. Qubit is the 2-level quantum system. For example, if we set $k = 2$ at (1), the electron in the Hydrogen atom can be in the ground state or the first excited state, or any superposition of the two. There are three basic axioms of quantum mechanics.

1. Superposition principle: It explains how a particle can be superimposed between two states at the same time.

2. Measurement principle: It tells us how measuring a particle changes its state and how much information can be accessed by measurement.

3. Unitary evolution: It governs how the state of the quantum system evolves in time.

Now let's see how a qubit follows three axioms of quantum mechanics. According to the superposition principle of quantum mechanics, the quantum state of the qubit can be represented as,

$$|\Psi> = \alpha\,|0> + \beta\,|1> \quad (2)$$

where α and β are complex numbers and $|\alpha|^2+|\beta|^2 = 1$.

According to the measurement principle of quantum mechanics, from a qubit, |Ψ> = α |0> + β |1>, we can measure |0> with probability $|\alpha|^2$ or, we can measure |1> with probability $|\beta|^2$. One important aspect of the measurement process is that it alters the state (α |0> + β |1>) of a qubit. The effect of the measurement is that the new state is exactly the outcome of the measurement. If the outcome of the measurement of |Ψ> = α |0> + β |1> yields |0>, then following the measurement, the qubit is in the new state |0>, (i.e. no more α |0> + β |1>). This implies that we can't collect any additional information about α and β by repeating the measurement. For a single qubit, it also proves the measurement axiom of quantum systems (Measurement destroys the quantum state).

Unitary transform is another very important operation on qubit. Quantum state evolves in time by unitary transform. In unitary transform, the qubit (represented as a column matrix) is multiplied by an unitary matrix. The result is a new qubit. A matrix U is unitary if, $UU^\dagger=U^\dagger U=I$. Here, $U^\dagger$ is the conjugate transpose of matrix U and I is the identity matrix.

V. Shor's Factoring Algorithm

In this section, we discuss Shor's factoring algorithm. Quantum Fourier transform is an integral part of Shor's algorithm. So, we first discuss QFT and period finding problem before presenting the steps of Shor's algorithm.

A. Quantum Fourier Transform

The quantum Fourier transform is almost the same transformation as discrete Fourier transform, although the conventional notation for the quantum Fourier transform is somewhat different [8]. The quantum Fourier transform on an orthonormal basis |0>, |1>, |2>, …, |*N*-1> is defined to be a linear operator with the following action on the basis states,

$$|j> \xrightarrow{QFT_N} \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{\frac{2\pi ijk}{N}} |k> \quad (3)$$

B. Period finding problem

The prime factorization problem is stated below [1].

Prime Factorization Problem: Given a composite odd positive integer *n*, find its prime factors.

It is well known that factoring *n* can be reduced to the task of choosing at random an integer *m* relatively prime to *n*, and then determining its modulo *n* multiplicative order *p* [1], i.e., to finding the smallest positive integer *p* such that,

$$m^p = 1(\text{mod } n) \quad (4)$$

Here, *p* is the called the period of the periodic function $f(p) = m^p$ mod *n*.

Finding the period of (4) is the bottleneck of prime factorization problem. Shor's algorithm uses quantum computing approach in finding the period of (4).

C. Shor's Algorithm

Shor's factoring algorithm consists of five steps (steps 1 to 5) [1] [11]. Only step 2 requires the use of a quantum computer. The remaining four steps can be performed in classical computer.

Step 1: Choose a random positive integer *m*. Use the polynomial time Euclidean algorithm to compute the greatest common divisor gcd (*m, n*) of *m* and *n*. If gcd (*m, n*) ≠ 1, then we have found a non-trivial factor of *n*, and we are done. If, on the other hand, gcd (*m, n*) = 1, then proceed to the next step.

Step 2: Use a quantum computer to determine the unknown period *p* of the function, $f(p) = m^p$ mod *n*.

Step 3: If p is an odd integer, then go back to Step 1. [The probability of p being odd is $(\frac{1}{2})^k$, where k is the number of distinct prime factors of *n*.] If p is even, then proceed to Step 4.

Step 4: Since p is even,

$$\left(m^{p/2} - 1\right)\left(m^{p/2} + 1\right) = m^p - 1 = 0\ (mod\ n) \quad (5)$$

If $\left(m^{p/2} + 1\right) = 0(mod\ n)$, then go back to step 1. If $\left(m^{p/2} + 1\right) \neq 0(mod\ n)$, then proceed to step 5. It can be shown that the probability that $\left(m^{p/2} + 1\right) = 0(mod\ n)$ is less than $(\frac{1}{2})^{k-1}$, where *k* denotes the number of distinct prime factors of *n*.

Step 5: Use the Euclidean algorithm to compute d_1 =gcd($m^{p/2} - 1, n$). Since $\left(m^{p/2} + 1\right) \neq 0(mod\ n)$, it can easily be shown that, d_1 is a non-trivial factor of *n*. Get another factor, $d_2 = n/d_1$.

In preparation for the discussion of step 2 of Shor's algorithm, we construct two quantum registers - register 1 and register 2. The size of the register 1 is L qubits such that $n^2 \leq Q = 2^L \leq 2n^2$. This register holds values {0, 1, 2, …, *Q*−1} and is used at the *Q*-point quantum Fourier transform, QFT_Q. Register 2 holds function values of $f(p) = m^p$ mod *n*. The size of register 2, M=floor($\log_2 n$)+1.

|REG1> |REG2> = |*p*> |*f(p)*>= |*a*>|*b*>= $|a_0, a_1, a_2, \ldots, a_{L-1}> |b_0, b_1, b_2, \ldots, b_{M-1}>$.

The step 2 of Shor's algorithm is explained below.

Step 2.1: Initialize register 1 and 2.

$|\Psi_0>$ = |REG1> |REG2> = |0> |0> = |000…0> |000…0>

Step 2.2: Apply the *Q*-point Fourier transform F to register 1.

$$|\Psi_0> = |0>|0> \xrightarrow{QFT_Q} |\Psi_1> = \frac{1}{\sqrt{Q}} \sum_{x=0}^{Q-1} |x>|0>$$

Hence, register 1 holds all the integers 0,1,2,…,*Q*-1 in superposition.

Step 2.3: Let U_f be the unitary transformation that takes |*x*>|0> to |*x*>|*f(x)*>. Apply the linear transformation U_f to the two registers. The result is:

$$|\Psi_1> = \frac{1}{\sqrt{Q}}\sum_{x=0}^{Q-1}|x>|0> \xrightarrow{U_f}$$

$$|\Psi_2> = \frac{1}{\sqrt{Q}}\sum_{x=0}^{Q-1}|x>|f(x)> = \frac{1}{\sqrt{Q}}\sum_{x=0}^{Q-1}|x>|m^x mod\ n>$$

The state of the two registers is now more than a superposition of states. In this step, we have quantum entanglement of the two registers.

Step 2.4: Measure register 2. It will create a periodic superposition in register 1. Then $|f>$ must collapse into some value $f(x_0)$. The state of $|x>$ will also collapse into the pre-image of $f(x_0)$. As f is periodic, the pre-image of $f(x_0)$ is $\{x_0, x_0 + p, x_0 + 2p, x_0 + 3p, \ldots, x_0 + (\frac{n}{r} - 1)p\}$.

We get,

$$|\Psi_3> = \frac{1}{\sqrt{Q}}\sum_{x=0}^{Q-1}|x>|f(x)> \xrightarrow{measure\ register\ 2}$$

$$|\Psi_4> = \sqrt{\frac{p}{Q}}\sum_{i=0}^{\frac{Q}{p}-1}|x_0 + ip>|f(x_0)>$$

Step 2.5: Apply the Q-point Fourier transform F to register 1.

$$|\Psi_4> = \sqrt{\frac{p}{Q}}\sum_{i=0}^{\frac{Q}{p}-1}|x_0 + ip> \xrightarrow{QFT_Q} |\Psi_5> = \frac{1}{\sqrt{p}}\sum_{i=0}^{p-1}|i\ \frac{Q}{p}> \emptyset i$$

Here, $\emptyset_i$ is some unimportant phase associated with each term due to the linear shift x_0.

Step 2.6: Measure register 1. Measured value is a multiple of Q/p. Save this value.

Go back to step 2.1. Repeat the algorithm several times to get distinct multiples of Q/p. Once enough values are found, calculate the gcd to retrieve Q/p. As Q is known, p is easily achieved.

VI. Factoring using GNFS

In this section, we discuss the execution of a software package named *GGNFS* to factor a 100 digit composite number [12]. Complex mathematical details of these algorithms are intentionally skipped. For that, the reader is referred to the many papers on the subject. The software package was used as a "black box" in this experiment in order to compare the performance of GNFS against Shor's algorithm. To execute GNFS, the software goes through five steps.

1. Polynomial selection.
2. Sieving.
3. Processing relations.
4. Finding dependencies among relations by solving matrices.
5. Computing the final factorization by square rooting.

Three components were used to implement GNFS.

i. An integrating Python code named factmsieve.py, downloaded from, http://gladman.plushost.co.uk/oldsite/computing/factmsieve.76.zip
ii. A software package named *GGNFS* for polynomial selection, processing relations and matrix solve, downloaded from, http://gilchrist.ca/jeff/factoring/ggnfs-svn413-win32-p4.zip
iii. A sieving tool: msieve.exe, downloaded from http://gilchrist.ca/jeff/factoring/msieve150_win32.zip

The Python script factmsieve.py is executed on the command prompt. This Python code uses other two tools. The machine and environment on which the Python script was run is described below.

System and program information:

- ⇨ Processor: Intel(R) Core(TM) i3-2330M
- ⇨ Speed: 2.20 GHz (4 CPUs)
- ⇨ Memory: 4GB
- ⇨ Operating system: Windows -7-6.1.7600
- ⇨ Python version: 2.7.3
- ⇨ File name: factmsieve.py
- ⇨ Free space required: 650 MB.
- ⇨ Other tools: GGNFS package and msive.exe
- ⇨ I/O: File.

Input and Output:

Input:

n (100 digits) =
2881039827457895971881627053137530734638790825166127496066674320241571446494762386620442953820735453

Output:

p (45 digits) =
618162834186865969389336374155487198277265679

q (55 digits) =
4660648728983566373964395375209529291596595400646068307

TABLE I. Execution Time Environment Details of GGNFS Software Package

Function	Average CPU Utilization (%)	Average core temperature (°C)	Running time (hours)
Polynomial	25	65	0.40
Sieving	100	82	1.33
Relations	100	82	0.02
Matrix	100	82	0.06
Square root	100	82	0.01

We can conclude the discussion of the GNFS implementation by the following remarks.

- Total running time (to factor a 100 digit composite): 1 hour 50 minutes.
- Except Polynomial selection all other functions are multithreading supported.
- After first 24 minutes, temperature of more than one core is simultaneously above 80 degree Celsius.

- High temperature for such long time can cause serious damage to the system.
- Sophisticated cooling system is required for running such programs.
- The presence of GPU can increase performance of this software many times.
- The latest version of msieve.exe uses CUDA technology of Nvidia GPU.

VII. Comparing against Shor's Algorithm

Asymptotic complexity of GNFS is $O(e^{[(\log n)^{1/3}(\log(\log n))^{2/3}]})$, when n is the integer to factor. Thus, this is a super-polynomial time algorithm in the number $O(\log n)$ of digits in n [1] [10].

On the other hand, Shor's algorithm factors integer in polynomial time. Shor's algorithm takes $O((\log n)^2(\log\log n)(\log\log\log n))$ steps on a quantum computer which is polynomial time in the number of digits $O(\log n)$ of n [1] [9].

The performance of Shor's algorithm on a quantum computer using the Beckman-Chari-Devabhaktuni-Preskill (BCDP) modular exponentiation algorithm [13] with various clock rates is compared to classical computers running the GNFS [14]. In Fig. 1, both horizontal and vertical axes are log scale. The horizontal axis is the size of the number being factored.

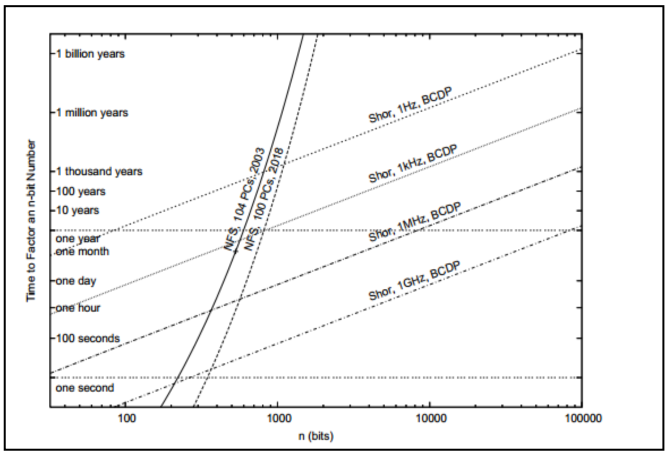

Fig. 1. Scaling of general number field sieve (GNFS) on classical computers and Shor's algorithm for factoring on a quantum computer, using BCDP modular exponentiation with various clock rates [14].

The steep curves are for GNFS on a set of classical computers. The left curve is extrapolated performance based on a previous world record, factoring a 530-bit number in one month, established using 104 PCs and workstations made in 2003. The right curve is speculative performance using 1,000 times as much computing power. This could be 100,000 PCs in 2003, or, based on Moore's law, 100 PCs in 2018. From these curves it is easy to see that Moore's law has only a modest effect on our ability to factor large numbers. The shallower curves on the figure are predictions of the performance of a quantum computer running Shor's algorithm, using the BCDP modular exponentiation routine, which uses $5n$ qubits to factor an n-bit number, requiring $\sim 54n^3$ gate times to run the algorithm on large numbers. The four curves are for different clock rates from 1 Hz to 1 GHz [14].

We have factored a number of 100 decimal digits (equivalent to 375 binary digits) in a 2.2 Giga Hertz classical computer using GGNFS software package which took 1 hour and 50 minutes with almost 100% CPU usage. But from Fig. 1, we can approximate that factoring a number of 375 bits using Shor's algorithm, implemented on quantum computers of different clock speeds, would have taken following times.

TABLE II. Approximate running times of Shor's algorithm in factoring an integer of 375 bits

Clock Speed	Approximate Time
1 GHz	$< 1\ second$
1 MHz	$\approx 120\ seconds$
1 KHz	$\approx 15\ days$
1 Hz	$\approx 10\ years$

As the actual implementation of quantum computer is still in research, we cannot run our test cases in quantum computer. Rather, we compare the runtime found in the GNFS execution against Shor's algorithm's predicted runtimes found from Fig. 1. We used Fig. 1 as an approximating tool. It is not a derived curve of this experiment, rather it compensates our resource lacking.

Another noticeable fact in Fig. 1 is that GNFS is executed using distributed computing, while Shor's algorithm running in standalone quantum computer of various speeds. But, in our experiment, we executed GNFS in a standalone computer. Certainly, the performance of distributed processing is much faster than the performance of standalone computer. But, here we actually showed how hard it is to factor RSA with key of 100 decimal digits in a personal computer using most efficient classical algorithm. Recall that the security of the most widely used cryptographic system RSA entirely depends on the difficulty of factoring large integers [15].

VIII. Conclusion

We see that both in standalone and distributed cases, classical computers, armed with most efficient algorithms, cannot factor integers which are 1000 decimal digits long. On the other hand, Shor's quantum factoring algorithm, although theoretically superior than any classical algorithm, depends on the architectural features like gate clock speed of quantum computer. Nevertheless, we conclude by saying that if actual implementation of quantum computer comes in reality, the public key cryptographic systems like RSA will be under great threat.

REFERENCES

[1]. S. J. Lomonaco. “A Lecture on Shor’s Quantum Factoring Algorithm”, arXiv:quant-ph/0010034v, Oct 19 2000.

[2]. P. Benioff, "The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines," Journal of Statistical Physics, Vol. 22, pp. 563-591, 1980.

[3]. R. Feynman, "Simulating Physics with Computers," *International Journal of Theoretical Physics,* Vol. 21, Nos. 6/7, 1982.

[4]. D. Deutsch, “Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer”in *Proceedings Royal Society London*, vol. 400 no. 1818 97-117, 1985.

[5]. D. Deutsch and R. Jozsa, "Rapid Solution of Problems by Quantum Computation,”in *Proceedings Royal Society London*, Vol. 439A, pp. 553-558, Dec 1992.

[6]. A. Berthiaume and G. Brassard, "The quantum Challenge to Complexity Theory," Proceedings of the 7th IEEE Conference on Structure in Complexity Theory, pp. 132-137, 1992.

[7]. M. Hayward, “Quantum Computing and Shor□s Algorithm.” Sydney: Macquarie University Mathematics Department. 2008.

[8]. M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information - 10th Anniversary Edition,* Cambridge University Press, Cambridge, UK, 2010.

[9]. P. Shor, “Algorithms for quantum computation: Discrete logarithms and factoring” in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, Nov 1994.

[10]. Carl Pomerance, “A Tale of Two Sieves”, *Notices of the AMS* **43** (12). pp. 1473–1485, Dec 1996.

[11]. U. Vazirani, “Quantum Mechanics and Quantum Computation”, Lecture notes on the online course. http://www.edx.org, 2012.

[12]. C. Monico, “GGNFS Documentation”, documentation of the GGNFS software package, Dec 31 2008

[13]. D. Beckman, A. N. Chari, S. Devabhaktuni, and J. Preskill, “Efficient Networks for Quantum Factoring,” Phys. Rev. A 54, 1034, arXiv:quant-ph/9602016v1, Feb 21 1996.

[14]. R. Meter, K. Itoh, T. Ladd, “Architecture Dependent Execution Time of Shor’s Algorithm”, arXiv:quant-ph/0507023v2, May 25, 2006.

[15]. W. Stallings, *Cryptography and Network Security – Principles and Practices,4th Ed.*, Prentice-Hall Inc., Upper Saddle River, New Jersy, USA, 2006.

[16]. C. Zalka, “Shor’s algorithm with fewer (pure) qubits,” arXiv:quant-ph/0601097, 2006.

[17]. Z. Zhou and M. Geller, “Factoring 51 and 85 with 8 qubits,” arXiv:1304.0128v1 [quant-ph], Mar 30, 2013.

[18]. L. Vandersypen, M. Steffen, G. Breyta, C. Yannoni, M. Sherwood, and I. Chuang, “Experimental realization of Shor’s quantum factoring algorithm using nuclear magnetic resonance”, Nature, pages 883–887, arXiv:quantph/0112176, 2001.

[19]. B. P. Lanyon, T. J. Weinhold, N. K. Langford, M. Barbieri, D. F. V. James, A. Gilchrist, and A. G. White, “Experimental demonstration of a compiled version of Shor’s algorithm with quantum entanglement,” arXiv:0705.1398v2[quant-ph], 2007.

[20]. Chao-Yang Lu, D.E. Browne, T. Yang, and Jian-Wei Pan, “Demonstration of a compiled version of Shor’s quantum factoring algorithm using photonic qubits,” Phys. Rev. Lett., 99:250504, Dec 2007.

[21]. Enrique Martin-Lopez, A. Laing, T. Lawson, Xiao-Qi Zhou, and J. O’Brien, “Experimental realization of Shor’s quantum factoring algorithm using qubit recycling,” Nature Photonics, 6:773–776, arXiv:1111.4147, 2012.

[22]. E. Lucero, R. Barends, Y. Chen, J. Kelly, M. Mariantoni, A. Megrant, P. OMalley, D. Sank, A. Vainsencher, J. Wenner, T. White, Y. Yin, A. N. Cleland, and John M. Martinis. “Computing prime factors with a Josephson phase qubit quantum processor,” Nature Physics, 8:719–723, 2012.