Quantum Algorithm for Square Root Convergence

Niall Dalton

February 2017

TABLE OF CONTENTS

1 ACKNOWLEDGEMENTS

I would like to thank Mrs. Taricco and Scott Aaronson for their help with the project and for contributing ideas to work with. Moreover, I would like to thank Dr. Wilson and Ms. Curran for guidance and input. Finally, I'd like to acknowledge Professor Hofri and Professor Aravind from WPI for discussion.

2 ABSTRACT

New quantum algorithms for classically inefficient problems must be developed to demonstrate the potential applications of quantum computation. The sum of square roots problem is one such problem; it compares the sum of the square roots of two sets of numbers. Fundamental aspects of quantum mechanics, such as superposition and interference, can be exploited for efficiency gains. Efficiency is measured through running time with respect to input size. The goal of this project was to engineer a quantum algorithm that efficiently solves the sum of square roots problem to a comparable or better precision than classical algorithms. Implementation was done through libquantum in C. The primary methodology involved a bounded logic function which modifies a superposition of convergent seeds for use in the Babylonian method of square root convergence. The superposition is then collapsed to determine an optimal seed with high probability. The algorithm resulted in a 94% success rate and demonstrated an efficiency of O(log2N). The results indicate that square root calculations and similar convergence problems may be more efficient on a universal quantum computer.

3 LITERATURE REVIEW

3.1 QUANTUM MECHANICS

Quantum mechanics is the study of atomic and subatomic particles, where classical mechanics cannot be used to accurately describe a system. Quantum systems arose from the strange nature of the quantum world and unexplainable occurrences, such as wave-particle duality. Starting with Max Planck, who first hypothesized the idea of a discrete packets of energy known as quanta, the ideas of quantum mechanics were established to account for the idiosyncrasies of the atomic world. De Broglie, another scientist, developed the idea that all particles could be described as waves and wave functions. Wave functions describe a quantum state. Schrödinger went a step further when he created his equation, aptly named Schrödinger's equation (Squires, 2016). The modern interpretation of his equation is that it describes the probability distribution of a particle in space (Peppe, 2013). Heisenberg's uncertainty principle, which states that the more we know about a certain condition of a particle, the less we know about a related condition, shows that a quantum state cannot be predicted deterministically, and thus probabilities must be employed (Squires, 2016).

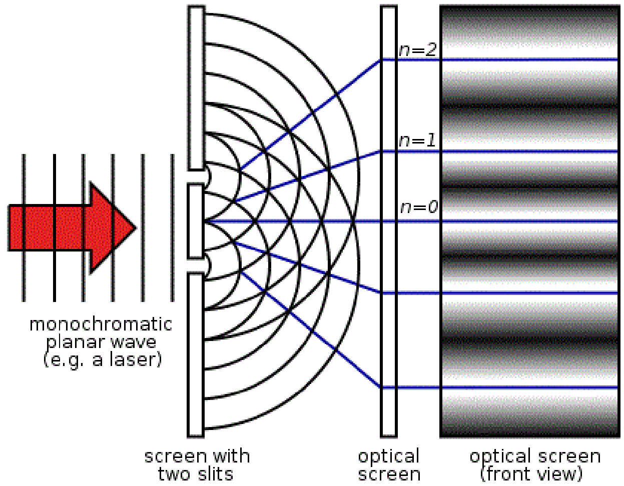


Figure 1. Double slit experiment. First performed by Thomas Young to demonstrate the wave nature of light. A wave, such as a laser or photon, is put through a screen with two slits, creating two new waves. The waves then interfere either constructively or destructively, thus amplifying or annihilating different waves. This is easy to observe by looking at the interference pattern to the right (Interference Diagram, n.d.).

The most important aspects of quantum mechanics are superposition, entanglement, interference, and measurement. Superposition refers to the ability of a particle to exist in a state that is a combination of classical states. Entanglement is used to characterize two particles that cannot be explained or described independent of each other; Einstein referred to it as "spooky action at a distance." Entanglement is essentially delocalized influence. The concept of entanglement often leads to the misconception that communication faster than light is possible. Such a notion would violate special relativity. Entanglement is useful however, in knowing a certain condition of a paired particle, because one can, in theory, instantaneously derive information about the other particle by knowing something about the original particle. This deduction is based on an entangled state, but no information is transmitted, thus preserving special relativity (Quantum computing 101, 2013). Interference relates to how the probability amplitudes of a wave function can cancel each other out. Interference is due to the nature of probability amplitudes, which can be negative or complex numbers. Technically, interference is how waves can interact, or interfere, with each other to form a new wave with different amplitudes. However, the cancelation of amplitudes, and thus the limitation of possible states, is the most important feature in regards to quantum computation. Figure 1 shows an example of interference in light waves. Finally, a particle can stay in a superposition until it is measured, thus causing the particle to collapse, or in other words, it must “chose” a classical state. The probability by which a particle will collapse into a specific state is described by the square of the probability amplitude of that state within the wave function (See Figure 2, in Quantum Logic and Gates). Consequently, the collapse of a particle into a classical state is also known as wave function collapse. Comparably, the period in which a particle exists in a quantum state, loosely interchangeable with superposition, is known as coherence. Therefore, the collapse of a particle is also known as decoherence. Maintaining coherent particles to use within quantum experimentation is particularly useful in quantum computing. However, this has proven to be a difficult roadblock for quantum computer scientists. Implementations of qubits, or effective particles for a task like quantum computing, will prove to be an important milestone in achieving a universal quantum computer (Aaronson, n.d.).

3.2 Complexity Theory and Algorithmic Efficiency

Complexity theory is a field of computer science that deals with the difficulty in solving any particular problem. Different measures of complexity may be used, based on how much of a certain resource is used, such as time, processors, gates used, and so on. Algorithmic efficiency is another quantifier of complexity that uses the number of iterations of an algorithm to describe the efficiency of the algorithm. Complexity theory and algorithmic efficiency are used to measure the practicality of computation. Complexity theory has many unanswered problems. The most prominent of which is "does P = NP?" P and NP are two complexity classes and the question is asking if the two classes are equivalent, meaning that any problem in P is also in NP, and vice versa. P is the complexity class that contains all the problems that can be efficiently solved with a classical algorithm. For reference, the P in both cases stands for polynomial time, which is the amount of time used by an efficient algorithm. Polynomial time includes all the cases when n, the number of iterations, is upper bounded by n to the power of some constant k. Big O notation is often used to describe algorithmic efficiency. Specifically, it describes the worst-case scenario for that algorithm, meaning it must go through some maximum number of iterations, i.e. an upper bound. For reference, polynomial time can be referred to as:

$$O(n^k)$$

where O is order/ Big O (running time), n is input size, and k is some constant power. NP, however, is slightly different from P, in that it stands for non-deterministic polynomial time. Non-deterministic in this case refers to a non-deterministic Turing machine. In contrast, the P class technically refers to a classical algorithm on a deterministic Turing machine. To understand this, the Church-Turing Thesis must first be introduced.

The Church-Turing thesis states that "any real-world computation can be translated into an equivalent computation involving a Turing machine" (Rowland, n.d.). A Turing machine is a computing device theorized by Alan Turing that models mathematical calculations by changing the state of an active "head," which changes the cell below the head on a tape and may be moved one unit left or right at a time (Weisstein, n.d.). Turing machines are described by a set of instructions given to the machine. A non-deterministic Turing machine differs in that it may have a separate set of instructions or rules for a situation that can lead to multiple paths of action, whereas a deterministic Turing machine can only perform one specified action based on a given situation. However, it is important to note that a deterministic Turing machine is, in a sense, equivalent to a non-deterministic one in terms of computation. A deterministic Turing machine can compute the same computation tree as a non-deterministic machine by going through all the different combinations or possibilities contained within the non-deterministic Turing machine's computation tree. Generally, a deterministic simulation of a non-deterministic Turing machine is exponentially long, and thus a non-deterministic Turing machine is thought to be more powerful in terms of time complexity. Despite the apparent difference in deterministic and non-deterministic Turing machines, the problem, does P=NP, is unresolved. A non-deterministic Turing machine is different from a quantum computer, however. Moreover, it is theorized that quantum computers are not as powerful as non-deterministic Turing machines, but this has not been definitively proven (Tušarová, 2004). It is also further theorized that a quantum Turing machine could have benefits over a classical computer (See Quantum Complexity Theory, paragraph 2).

3.3 Quantum Complexity Theory

Bounded Error Quantum Polynomial Time (BQP) is the complexity class that contains problems solvable by a quantum computer, in polynomial time, and with a chance for error less than or equal to 1/3 (Complexity Zoo:B, n.d.). Unlike classical computers, quantum computers have probabilistic output; they will have different possible outputs for the same input. The probability of each unique output is equal to the square of the amplitudes of each combination of possible outputs, or states, in the superposition. These probabilities represent the same concept as superposition of wave-particles (See Quantum Mechanics section) except that each possible state is a combination of qubits (Aaronson, n.d.).

Additionally, quantum computers are theorized to be more powerful in terms of time complexity in certain structured problems, likely problems between P and NP-complete. Therefore, simulations of quantum computers through classical computers are exponential in terms of time complexity. For example, Shor's algorithm, named after Peter Shor, has been shown to find the prime factors of a number in exponentially less time than the best known classical algorithms, such as the general number field sieve. Although Shor's algorithm can only factor very small numbers due to the small number of qubits available in current quantum computers, this is by far the most significant quantum algorithm because it not only provides a speedup but it has actual application in real world scenarios (Aaronson, n.d.). The primary application of prime factorization is using it to break RSA encryption, which is used worldwide in myriad of different online services. RSA encryption relies on multiplying two large prime numbers to create a key. The amount of time it would take for a classical computer to find those factors is intractably large, but a theoretical universal quantum computer or quantum Turing machine, one that essentially has no physical or technical limitation, could find those factors in exponentially less time and thus would "break" RSA encryption (Weisstein, n.d.).

Another famous quantum algorithm is Grover's algorithm, which can find an element of an unsorted databases in the square of the time of the best known classical algorithm for searching unsorted databases, which must look at each element in the worst case (Aaronson, n.d.). Thus, classical algorithm worst case is given by $O(n)$ while Grover's algorithm is given by $O(\sqrt{n})$. As a side note, time here is loosely used, as it really refers to the number of iterations of the algorithm; a quantum computer will likely complete its iterations slower, but the number of iterations will begin to show a speedup in terms of time when given a reasonably large input. Thus, both quadratic and exponential speedups are potentially possible with quantum computers, but physical limitations are still a large issue. The final point to reiterate is that the idea of quantum supremacy, which is to say that quantum computers have potential for speedups over classical computers, is not yet decisively proven, but evidence is promising.

3.4 Quantum Logic and Gates

A quantum state is often described using Paul Dirac's bra-ket notation. A ket is |ψ> (pronounced ket psi) where ψ is some quantum state, most often a superposition, and a bra is < φ |. A ket is sometimes described by a column matrix or vector of the probability amplitudes of any *n* possible states (See Figure 2 below). A bra is like a ket, except it is the conjugate transpose of a ket. Thus, a bra is a row vector containing the complex conjugate of the probability amplitudes. The inner product is a bra times a ket, a bra-ket, and is written as < φ| ψ> (Hayward, n.d.).

$$\sum_{i=0}^{n-1} a_i|x_i>$$

$$|\psi>= \begin{pmatrix} a_0 \\ a_1 \\ . \\ . \\ . \\ a_{n-1} \end{pmatrix}$$

Figure 2. The summation equation represents |ψ> as the sum of all n states with some arbitrary complex probability amplitudes, a_i, as the coefficients of the corresponding state, x_i. Likewise, |ψ> can also be represented by a column matrix or vector of n probability amplitudes.

As opposed to traditional logic implementations, quantum logic gates and circuits have some special restrictions. The fundamental difference is that quantum logic must maintain reversibility. This is because a quantum system cannot lose information and therefore, classical gates, like an AND gate, cannot be implemented by normal means.

Table 1. A and B are input bits, the output column describes A ^ B, A and B. A and B is only true, or 1, if both input bits are also 1. Otherwise, the output is false, or 0.

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Per the truth table for an AND gate in table 1 above, it is impossible to decisively determine what inputs were used when given any particular output. To deal with this issue, new logic gates are used in quantum circuits. For example, the CNOT, or controlled not, gate is used to implement an XOR without information loss.

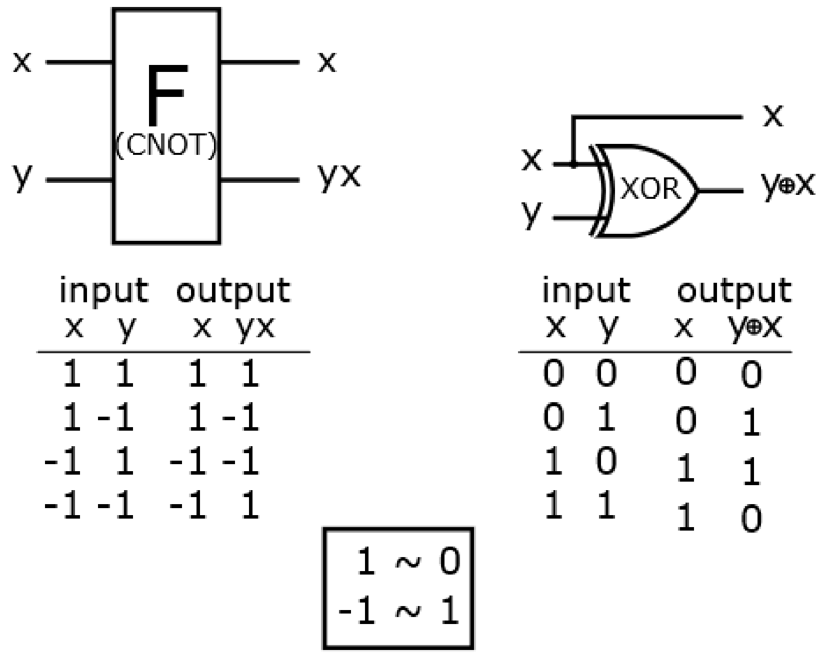


Figure 3. The diagram to the right shows a normal XOR gate, which essentially flips the target bit, y, if the control bit, x, is true (state of 1). The two outputs are x, which exists to retain information, and y⨁x, which is x XOR y. Outputting x is not normally useful or even used, but the CNOT, the quantum analogue of the XOR with x as an additional output, needs this extraneous output in order to retain information. The legend describes how different states may be used in a quantum system (Cnot-compared-to-xor, n.d.).

The CNOT effectively implements entanglement because the target and control bit cannot be described independently of each other (See Figure 3 above). Another example is the Hadamard gate, which puts a qubit into an equal superposition of two states.

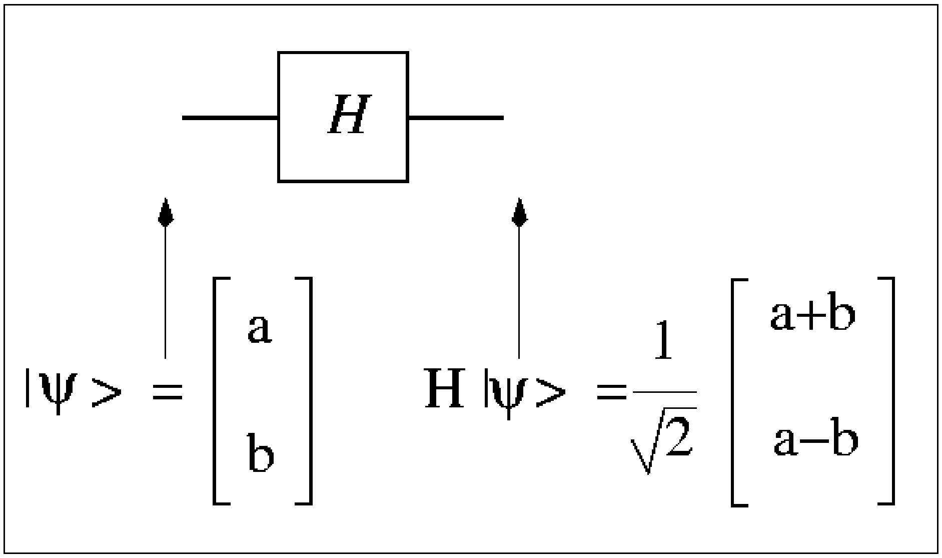


Figure 4. The above shows a Hadamard gate run on a single qubit. The Hadamard gate creates a new superposition where the two basis states given have equal probability amplitudes. Recall that the probability is the square of the probability amplitude – the coefficient to the state. Thus, squaring (1/√2) gives you a ½ probability of the qubit collapsing into either state (The Hadamard Gate, n.d.).

Given two basis vectors on the horizontal and vertical axis that represent two possible states – i.e. 0 and 1 – an equal superposition between these two states can be represented by applying the Hadamard matrix, as shown in Figure 4 above, of the horizontal basis vector (Hayward, n.d.). Recall that cos 45 equals sin 45, which both equal √2/2, meaning that a vector that is equal parts state 0 and state 1 must also exist at 45 degrees and thus also must be defined as [√2/2*state 0, √2/2* state 1]. Additionally, recall that √2/2 equals 1/√2, which we find in the Hadamard gate above. It is written in such a manner to better visualize the connection between the probability amplitude and the probability, which is the square of the amplitude. It follows that $(1/\sqrt{2})^2$ equals ½ and this demonstrates the equal chance of finding either basis state, which can be seen in Figure 4.

The power of quantum computers lies in exploiting exponential parallelism through the modification of qubits. For example, two qubits, with 2 states, 0 and 1, can exist in 4 states – 00, 01, 10, or 11. Similarly, an arbitrary *n* number of qubits can exist in up to 2^n states, given two possible states. Thus, those 2^n states must also necessarily have 2^n probability amplitudes if those states exist in a superposition, like they can with qubits. It follows that modification of $|\psi>$ will affect the whole superposition and its 2^n probability amplitudes, although only *n* qubits are used. Herein it is hypothesized that a universal quantum computer would have the potential for exponential speedups over classical computers.

Quantum algorithms use properties of quantum logic to gain speedups. Both CNOT gates and Hadamard gates are widely used. A variety of other quantum gates are popular as well, such as the Pauli spin matrices, which affect the direction of spin in a certain direction – x, y, or z. Combined with the physical realization of a quantum computer that is scalable to a reasonable extent, quantum logic will likely be extremely useful to programmers in the future.

4 Engineering Plan

4.1 Engineering Problem Statement

Existing classical algorithms are not able to efficiently solve some problems. Therefore, new quantum algorithms need to be developed to improve upon current computing capabilities and demonstrate quantum capabilities for future applications.

4.2 Engineering Goal

The goal of this project was to engineer a quantum algorithm that efficiently solves the sum of square roots problem through square root convergence to a comparable or better precision than classical algorithms.

Sum of squares problem: Given two sequences $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ of positive integers, is $A := \sum_i \sqrt{a_i}$ less than, equal to, or greater than $B := \sum_i \sqrt{b_i}$?

By engineering a new quantum algorithm that exploits quantum mechanics, a speedup over similar classical algorithms is achieved. The sum of squares problem has several applications in subroutines of larger problems, most notably in problems involving Euclidean distance.

4.3 Development

An algorithm was programmed and implemented with the help of a quantum computation library. Engineering of the algorithm was done on a laptop computer and that computer simulated the quantum system evolved by the algorithm. The algorithm was engineered by considering the aspects of quantum mechanics and exponential parallelism, as well as linear algebra, in order to deduce a more efficient route than classical solutions.

4.4 Design Criteria

The algorithm implemented must have a high success rate and be efficient. These two criteria are determined by the speed at which the algorithm executes its task and how often it gives the correct answer, as the algorithm is probabilistic.

4.5 Testing

The algorithm was tested by running the algorithm on a computer many times and comparing the results to classical analogues.

5 Methodology

5.1 Programs and Libraries Used

A series of code examples of existing classical square root calculation methods written in C++ were used to compare the relative efficiency of each method. A zip file was provided from a webpage detailing the code (El-Magdoub). A programming library for C known as libquantum was also used to simulate quantum computation. Libquantum provides a wide array of functions to simulate qubits, quantum gates, and arbitrary quantum algorithms.

5.2 Iterative Design

The algorithm was designed by adapting current existing classical algorithms for square root calculations by adding quantum subroutines and modifying existing subroutines. Different quantum subroutines were added to try to achieve speedups. Multiple versions of the algorithm were created using various methods. The first version (V1) used a quantum subroutine to test multiple seeds to achieve faster convergence in a modified version of the Babylonian method of computing square roots. The quantum subroutine used amplitude amplification to modify the amplitudes of the base states. The second version (V2) used a quantum subroutine to estimate the square root of a number by applying the Pauli Z transform at conditional intervals. Both algorithms were written in C with the usage of libquantum.

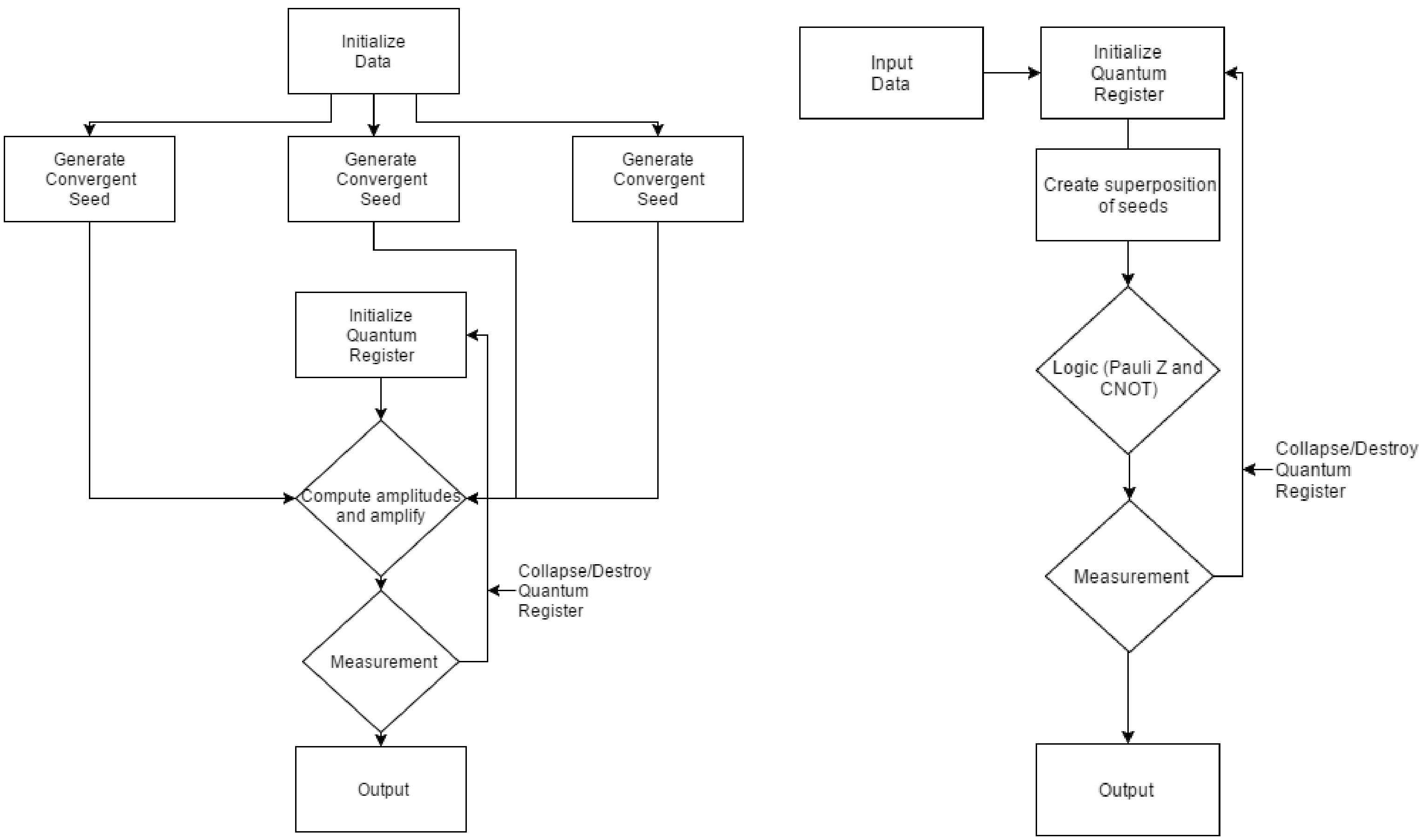


Figure 5. The diagram to the left shows the process of convergence where amplitude amplification is used for a favorable probability distribution of results and is representative of version 1. The diagram to the right shows the process of by which an optimal seed is found through Pauli Z transforms and the CNOT gate and is representative of version 2.

5.3 Testing

The various versions of the algorithm were tested by determining the time taken to compute many square roots. This was done by finding the difference in the computer's internal clock before and after the calculations. A bash script was used to execute the algorithms many times over, and for the second version, with different configurations. Efficiency was also measured by the number of iterations required to complete the algorithm. The success rate of the first version algorithm was measured by comparing the output of the algorithm to the known actual square root and looking at how often the output was as or more precise than a control classical method. The success rate of the second version of the algorithm was measured by comparing the output of the algorithm to the known actual square root against a $\log_2$ result.

5.4 Comparing Versions

Iterations were compared by looking at their success rate and efficiencies. Efficiency was valued over success rate as success rate is easier to account for and fix in later revisions. A success rate of greater than 66% is necessary for an algorithm to exist in complexity class BQP, which describes problems that have an efficient corresponding quantum algorithm.

6 Results

6.1 Output

6.1.1 Version 1

Table 2. The outputs for version one over 10 trials and the average running time over those trials for $n = 12$.

Trial	Output
1	3.5
2	3.471622
3	3.471622
4	3.464286
5	3.5
6	3.5
7	4.173077
8	4.173077
9	3.464286
10	3.464286
V1 Time	
Average	0.087927

Version 1 outputted its seed averaged with a $\log_2$ approximate and gave that to a Babylonian iteration. This was compared against just a $\log_2$ approximate given to a Babylonian iteration. Version 1 was fairly limited in scope and was not extensively tested like version 2.

6.1.2 Version 2

Table 3. The titles of the outputs measured and/or calculated

Number	Width	Actual Sqrt	Seed Alg.	Output from Seed Alg.	Error Alg.
Time Alg.	Log2 Seed	Output from Log2 Seed	Error Log2 Seed	Time Log2 Seed	Success Rate

The following matrix was used to gather data. The titles correspond to the columns of the table below. The iterations of the algorithm resulted output the following sample results for n = 18595:

Table 4. A few sample outputs

18595	15	136.3635	144	136.566	0.148491	0.248303	14.18	662.6469	385.9416	0	1
18595	15	136.3635	144	136.566	0.148491	0.253934	14.18	662.6469	385.9416	0	1
18595	15	136.3635	144	136.566	0.148491	0.255872	14.18	662.6469	385.9416	0	1
18595	15	136.3635	196	145.4362	6.65335	0.259539	14.18	662.6469	385.9416	0	1
18595	15	136.3635	196	145.4362	6.65335	0.261429	14.18	662.6469	385.9416	0	1
18595	15	136.3635	196	145.4362	6.65335	0.261898	14.18	662.6469	385.9416	0	1
18595	15	136.3635	128	136.6367	0.200372	0.255478	14.18	662.6469	385.9416	0	1
18595	15	136.3635	128	136.6367	0.200372	0.258868	14.18	662.6469	385.9416	0	1

The number n for each result was randomly chosen between 1 and 32,767. Width is the number of qubits in the quantum register and is the ceiling of $log_2 n$. The actual square root was computed by using a seed put through 10 Babylonian iterations, which was deemed an appropriate precision. Output from the algorithm was measured by collapsing the superposition of seeds to give a single seed. That seed is then put through a single Babylonian iteration and the final output is displayed, as seen above. The error was computed against the actual square root and the time it took for the algorithm to run was recorded.

6.2 Efficiency

Moreover, the second version algorithm had a running time relative to input size n as shown in the graph below:

Figure 6. The running time of the algorithm relative to the size of the quantum register, the width. It is generally exponential is terms of complexity through simulation, which is how the algorithm was measured.

Efficiency is measured more accurately, however, through analysis the code. This is due to differences in processing power and variance, among other factors, which can portray an incomplete or inaccurate result of algorithmic efficiency (See Literature Review – Algorithmic Efficiency). A similar classical algorithm for optimizing the Babylonian seed that runs through all possible integral seeds through n would have running time $O(n)$.

Table 5. The computational complexity of the different versions of the algorithm.

Computational Complexity	
Classical	$O(n)$
V1	$O(n)$
V2	$O(log_2 n)$

Simulations of quantum systems on classical computers are exponentially expensive computationally, as demonstrated in Figure 6. The running time of the quantum algorithm V2 on a classical computer is $O(2^{log_2 2^n})$, which simplifies to an exponential order running time. Herein the time efficiencies are shown to show the effect of simulation, but it is noted that the efficiency in Table 5 should be referred to for a more legitimate characterization of efficiency. These were found by considering the order of the running time involved with the number of repetitions within the algorithm.

6.3 Success Rate

The two versions of the algorithm, which outputted a relatively optimal seed, was compared to a random bounded seed and a log 2 seed. Furthermore, the success rate was compared as the number of qubits was varied. The second algorithm involved bounds for its logic function and as such the success rate of 3 distinct bounds were measured.

Table 6. The success rate of version 1 and the extreme configurations of versions 2 are compared.

Success Rate	
V1	0.4333
V2 - High Decoherence Parameter	
Bound 0.75	0.192
Bound 1.0	0.072
V2 - Low Decoherence Parameter	
Bound 0.75	0.939
Bound 1.0	0.939

Success rate was measured by comparing the outputted seed, and a classically computed log 2 seed, both given to one Babylonian iteration. Success was defined when the quantum algorithm's output had the lower percent error relative to a control constant measured through 10 Babylonian iterations, which gives the correct square root with very high precision. Note that a third bound of 0.5 was not compared because it often gave null results and was ruled out for its high error. An important consideration to make it that the first algorithm did not consider decoherence, so the low decoherence parameter configurations of V2 are a better comparison against V1.

Table 7. The percent error of V1 and the various configurations of V2.

Percent Error	
V1	4.490%
V2 - High Decoherence Parameter	
Bound 0.75	2405.594%
Bound 1.0	4012.910%
V2 - Low Decoherence Parameter	
Bound 0.75	6.642%
Bound 1.0	7.779%

Finally, percent error was calculated and compared to test the absolute accuracy of the algorithm. Like the other tests, various qubit sizes were tested as well as different inputs and different bounds for the second algorithm. Recall that the low decoherence parameter configurations are a better comparison to V1.

Table 8. The matrix used to score the different versions of the algorithm.

Scoring Matrix			
	Weight	V1	V2
Efficiency	16	4	16
Success Rate	12	5.200	11.273
% Error	6	5.731	5.601
Total	34	14.931	32.875

The above matrix was used to compare the algorithms. Efficiency was scored relative to the descriptions in table 5. Success rate was simply multiplied by the weight to get the scores for success rate. The best success rate for version 2 was used. Likewise, percent error was calculated in the same manner. The scores were then added for a total score. According to these totals, found in table 8 above, version 2 is superior to version one.

7 ANALYSIS

The data shows that version two is likely superior to classical algorithms. With quantum error correction (QEC), longer coherence times could be achieved with higher success rates. The success rates in the second version indicate that optimizing seeding for convergence problems can fall under BQP and thus be solved efficiently.

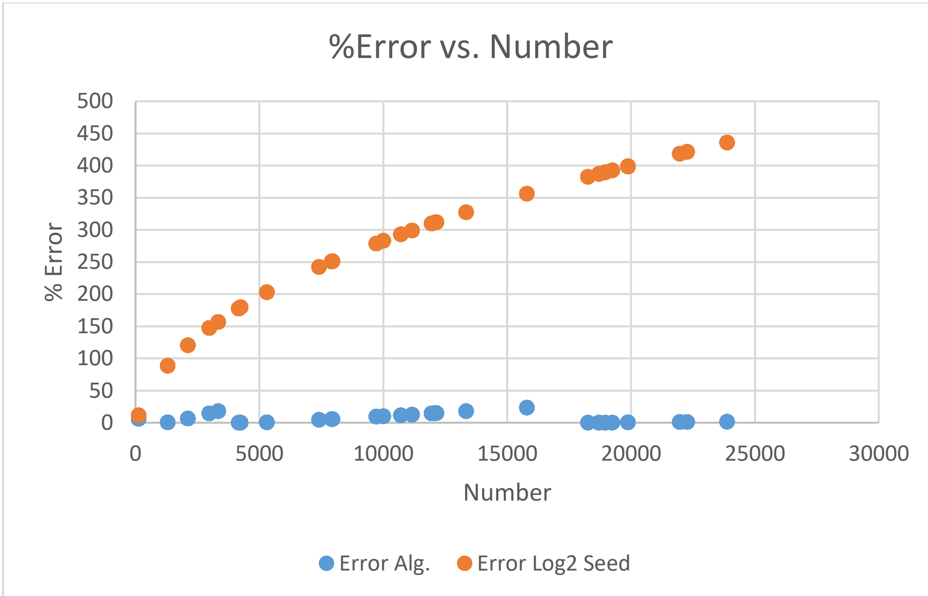


Figure 7. Percent error is shown against number size. Error tends to grow slowly for the quantum algorithm's seed and much quicker for a $\log_2$ seed.

The second version of the algorithm showed superior results for a bound of 0.75 and for low coherence times. Error went down significantly, as seen in Figure 8. Moreover, Figure 8 clearly shows the reduced error within a bound of 0.75 versus 1.0. A bound of 0.5 was also considered, but not picture due to extreme error and deviations.

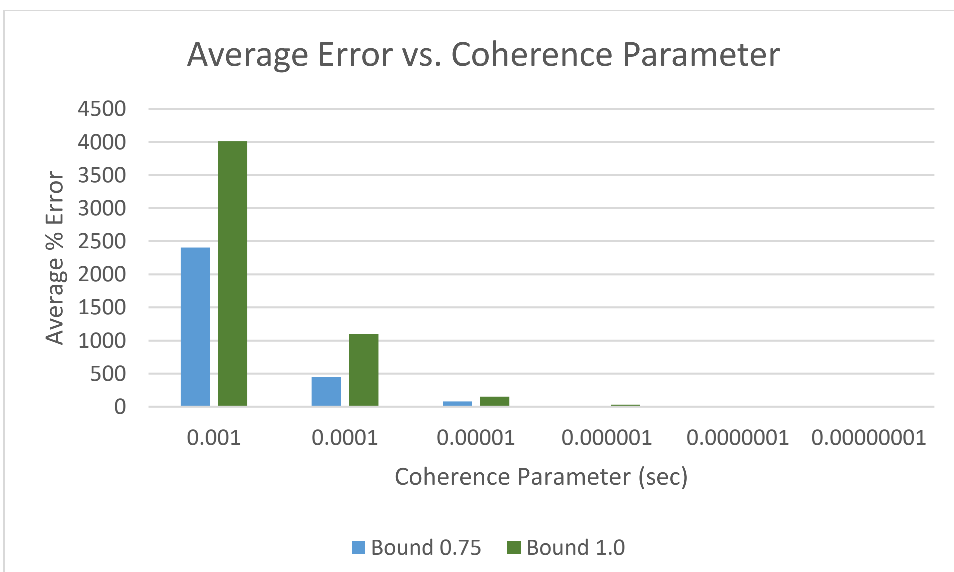


Figure 8. Average error against coherence parameter. The coherence (decoherence) parameter is "phase coherence time as seen experimentally. [The parameter] is the upper bound on the length of time over which a complete quantum computation can be executed accurately" (DiVincenzo, 1995).

However, it is noted that no definitive conclusions can be drawn from the data due to the inherent flaws of simulation. Thus, such data is only a model of what a quantum computer can hope to achieve and may not have the same outputs under realistic circumstances, i.e. on a physical quantum computer.

Further improvements could be done to achieve a higher success rate and QEC could be added to better deal with the effects of decoherence. Other bounds and higher amounts of qubits could be tested to see how such changes affect the output of the algorithm. Floating point representations could be added in as well to achieve better precision. Testing could also be done on a real quantum computer for a more lifelike test, rather than just simulation. Moreover, adapted versions of the algorithm could be used to efficiently solve other convergence based problems.

8 CONCLUSION

Square root convergence is an interesting problem that can be modified to other convergence problems and it serves as a basis for some square root calculations. The quantum algorithm developed for optimizing square root convergence demonstrates the potential for quantum algorithms with bounded logic functions in convergence problems. Large computations and subroutines, as well as mainstream applications like graphics processing, would find efficiency speedups useful. Furthermore, quantum algorithms in regards to certain problems may inspire others to engineer new quantum algorithms for practical purposes.

9 REFERENCES

Aaronson, S. (n.d.). Quantum Computing. Lecture. Retrieved November 15, 2016, from http://www.scottaaronson.com/democritus/lec10.html

Alder, G. (2017). Flowchart Maker & Online Diagram Software. Draw.io. Retrieved 3 February 2017, from https://www.draw.io/

B., K. (2015, November 12). BASH Syntax error near unexpected token 'done' Retrieved February 3, 2017, from https://stackoverflow.com/questions/18367708/bash-syntax-error-near-unexpected-token-done

Cnot-compared-to-xor [Digital image]. (n.d.). Retrieved December 4, 2016, from https://upload.wikimedia.org/wikipedia/en/5/58/Cnot-compared-to-xor.svg

Coffin, Jerry. "Generate a Random Number within Range?" C - Generate a Random Number within Range? Stack Overflow, 8 June 2010. Web. 2 Feb. 2017.

Complexity Zoo:B. (n.d.). Retrieved November 14, 2016, from https://complexityzoo.uwaterloo.ca/Complexity_Zoo:B

DiVincenzo, D. (1995). Two-bit gates are universal for quantum computation. Physical Review A, 51(2), 1015-1022. http://dx.doi.org/10.1103/physreva.51.1015

El-Magdoub, Mahmoud. "Best Square Root Method - Algorithm - Function (Precision VS Speed) - Codeproject". Codeproject.com. N.p., 2017. Web. 7 Jan. 2017.

Hayward, M. (n.d.). Quantum Computing and Shor's Algorithm. Retrieved December 01, 2016, from https://quantum-algorithms.herokuapp.com/299/paper/node1.html

[Interference Diagram]. (n.d.). Retrieved from https://upload.wikimedia.org/wikipedia/commons/thumb/8/8b/Two-Slit_Experiment_Light.svg/400px-Two-Slit_Experiment_Light.svg.png

Jeffε. "Problems Between P and NP." Theoretical Computer Science Stack Exchange. N.p., 29 Dec. 2010. Web. 15 Nov. 2016.

Kemp, P. (2013, April 6). What is the best way to seed srand()? Retrieved February 2, 2017, from https://stackoverflow.com/questions/15846433/what-is-the-best-way-to-seed-srand

Peppe (2013) (http://physics.stackexchange.com/users/10323/peppe), Why is quantum mechanics based on probability theory?, URL (version: 2013-07-01): http://physics.stackexchange.com/q/69730

Quantum computing 101. (2013). Retrieved November 15, 2016, from https://uwaterloo.ca/institute-for-quantum-computing/quantum-computing-101

Rowland, Todd. (n.d.). "Church-Turing Thesis." From MathWorld--A Wolfram Web Resource, created by Eric W. Weisstein. http://mathworld.wolfram.com/Church-TuringThesis.html

Squires, G. L. (2016). Quantum mechanics. In Encyclopædia Britannica. Retrieved November 15, 2016, from https://www.britannica.com/science/quantum-mechanics-physics

The Hadamard Gate [Digital image]. (n.d.). Retrieved December 4, 2016, from http://people.math.gatech.edu/~jeanbel/4803/Lectures/Gates/qgates.html

Tušarová, Tereza (2004). "Quantum complexity classes". arXiv:cs/0409051

Weisstein, Eric W. (n.d. a)."RSA Encryption." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/RSAEncryption.html

Weisstein, Eric W. (n.d. b). "Turing Machine." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/TuringMachine.html

10 Appendices

10.1 Assumptions

The functionality of the algorithm assumes that an algorithm exists that solves the problem, the algorithm can be efficient, and that the algorithm can use quantum mechanics to quantify itself as a quantum algorithm. It is furthered assumed that the libquantum documentation is accurate and that libquantum is an accurate model in describing and simulating quantum mechanics and in executing the algorithm. Finally, it is assumed that any data gathered is a representative sample and is accurate.

10.2 Limitations

Some limitations of the project include that it is simulation based, so it is not necessarily wholly representative of real word conditions and outputs. The algorithm may also be hard to implement on a real quantum computer due to the underdevelopment of such existing machines and the status of quantum programming. Moreover, the time evolution of the algorithm must be simulated as well and is also not necessarily representative of real world conditions.

10.3 Search Terms and Engines

- Google
    - Quantum algorithms
    - Grover's algorithm
    - Shor's algorithm
    - Computational Complexity
    - Methods for square root calculations
    - Bash scripting
- WPI Gordon Library Search
    - Quantum computing
    - Quantum algorithms
    - Quantum gates and logic

10.4 Code

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

#include <quantum.h>

int main(int argc, char **argv) {

	int numStates = 10;
	int width = 1;
	//float num = 8;
	float num = atof(argv[1]);
	int approx = 2;
	unsigned int result = 0;

	int sum = 0;
	float sum2 = 0;
	int runs = 2;
	float result2 = 0;
	float num2 = 0;

	//for (int j = 0; j < runs; j++) {

		quantum_reg reg;
		do {

			reg = quantum_new_qureg_sparse(numStates, width);

			srand(time(0));

			for (int i = 0; i < numStates; i++) {
				reg.state[i] = i + 1;
				printf("State: %u\n", reg.state[i]);
			}

			float approxNew = approx;//(approx + (num / approx)) / 2.0;

			printf("New approx is: %f\n", approxNew);

			for (int i = 0; i < numStates; i++)
				reg.amplitude[i] = 0.31;

			for (int i = 0; i < numStates; i++) {
				float temp = 1 - (abs(i + 1 - approxNew) / approxNew);
				float temp2 = temp / 9.25;

				printf("Amplitude %i: 0.31 + %F\n", i + 1, temp2);
				if (temp2 > 0)
					reg.amplitude[i] += temp2 * 3;
				else
					reg.amplitude[i] += temp2 / 3;
			}

			//reg.amplitude[5] = 1;

			for (int i = 0; i < numStates; i++) {
				float prob = quantum_prob(reg.amplitude[i]);
```

```
                        printf("Probability %i is: %f\n", i + 1, prob);
                }

                //int result = quantum_bmeasure(0, &reg);
                result = quantum_measure(reg);
                result2 = ((int)result + num / (int)result) / 2.0;

                //quantum_delete_qureg(&reg);
        } while (result > 10);

        sum += (int)result;
        sum2 += result2;
        //printf("Result 2: %f Sum 2: %f Average 2: %f\n", result2, sum2,
(float)sum2 / runs);

        printf("Result: %i\n", result);
        printf("Result 2 is: %f\n", result2);

        quantum_delete_qureg(&reg);
    //}

    printf("Sum 1: %i\n", sum);
    printf("Sum 2: %f\n\n", sum2);

    //quantum_delete_qureg(&reg);

    sum2 = (sum2 + (num / sum2)) / 2.0;
    char output[50];
    snprintf(output, 50, "%f", sum2);

    FILE *fp;

    fp = fopen("C:/Users/Niall/Desktop/STEM/outputs.txt", "a+");
    fprintf(fp, "\n");
    fprintf(fp, output);
    fclose(fp);

    printf("Sqrt: %f\n", sum2);


    //quantum_delete_qureg(&reg2);

    return 0;
}
```

Above is Sqrt2.c (Version 1)

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

#include <quantum.h>

// https://stackoverflow.com/questions/2999075/generate-a-random-number-within-
range/2999130#2999130
int rand_lim(int limit) {
	/* return a random number between 0 and limit inclusive.
	*/

	int divisor = RAND_MAX / (limit + 1);
	int retval;

	do {
		retval = rand() / divisor;
	} while (retval > limit);

	return retval;
}

int main(int argc, char **argv) {

	int num = atoi(argv[1]);
	int width = quantum_getwidth(num);
	double i;
	double number = (double)num;

	srand(time(0));

	struct timeval  tv1, tv2;
	gettimeofday(&tv1, NULL);

	quantum_reg reg;

	//quantum_qec_encode(1, width, &reg);

	reg = quantum_new_qureg(0, width);

	quantum_set_decoherence(0.000001);

	for (i = 0; i<reg.width; i++)
		quantum_hadamard(i, &reg);

	//quantum_print_qureg(reg);

	for (i = 0; i < reg.width; i++) {
		if ( pow(2,i) < (number + number*0.75) && pow(2, i) >(number -
number*0.75)) {

				//quantum_phase_kick(i / 2, 180, &reg);

				//quantum_print_qureg(reg);

				quantum_sigma_z(i / 2, &reg);
```

```
//quantum_r_z(i / 2, 180, &reg);

                    for (int j = 0; j < i; j++) {
                           if (pow(2, j) + pow(2, i) < (number + number*0.75) && pow(2,
j) + pow(2, i) > (number - number*0.75)) {
                                  //quantum_sigma_z(j / 2, &reg);
                                  //quantum_sigma_z(i / 2, &reg);
                                  //quantum_sigma_x(i / 2, &reg);
                                  quantum_cnot(reg.width - i - j - 1, reg.width - i - j,
&reg);
                                  quantum_sigma_z(i / 2, &reg);
                                  quantum_cnot(reg.width - i - j - 1, reg.width - i - j,
&reg);
                                  //quantum_cnot(reg.width - i - j - 1, reg.width - i - j
- 2, &reg);
                           }
                    }

             }
      }

      //quantum_print_qureg(reg);

      for (i = 0; i<reg.width; i++)
             quantum_hadamard(i, &reg);

      //quantum_qec_decode(1, width, &reg);

      //quantum_print_qureg(reg);

      int result = quantum_measure(reg);
      double result2 = ((double)result + ((double)number / (double)result)) / 2.0;

      //quantum_qec_decode(1, width, &reg);

      printf("Result: %i \t\t Result 2: %f\n", result, result2);

      quantum_delete_qureg(&reg);

      gettimeofday(&tv2, NULL);

      printf("Total time = %f seconds\n",
             (double)(tv2.tv_usec - tv1.tv_usec) / 1000000 +
             (double)(tv2.tv_sec - tv1.tv_sec));

      int seed = rand_lim(number - 2) + 1;
      float result3 = (seed + (number / seed)) / 2.0;

      printf("Seed: %i \t\t Result 3: %f\n", seed, result3);

      double seed2 = log2(number);
      double result4 = (seed2 + (number / seed2)) / 2.0;

      printf("Seed 2: %.2f \t\t Result 4: %f\n", seed2, result4);

      struct timeval  tv3, tv4;
      gettimeofday(&tv3, NULL);
      double j;
```

```
	//for (j = 0; j < 1000000000; j++)
		log2(number);
	gettimeofday(&tv4, NULL);

	printf("Total time = %f seconds\n",
		(double)(tv4.tv_usec - tv3.tv_usec) / 1000000 +
		(double)(tv4.tv_sec - tv3.tv_sec));


}
```

Above is sqrt6.c (Version 2)

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

// From Square Root Video gamer developer
float run4(int x) {
	union
	{
		int i;
		float x;
	} u;

	u.x = x;
	u.i = (1 << 29) + (u.i >> 1) - (1 << 22);
	return u.x;

}

double run3(int num) {
	double result;
	double exp = 0.5*log(num);

	result = pow(M_E, exp);

	return result;
}
```

```
// Code-snippet from wiki contributer
short run2(short num) {
	short res = 0;
	short bit = 1 << 14;


	while (bit > num)
		bit >>= 2;

	while (bit != 0) {
		if (num >= res + bit) {
			num -= res + bit;
			res = (res >> 1) + bit;
		}
		else
			res >>= 1;
		bit >>= 2;
	}
	return res;
}


float run(int num) {
	int approx = 2;
	float sqrt;

	float result;

	result = (approx + num / approx) / 2.0;
	sqrt = result;

	sqrt = (sqrt + (num / sqrt)) / 2.0;

	return sqrt;
}


int main(int argc, char **argv) {

	float num = atof(argv[1]);

	struct timeval  tv1, tv2;
	gettimeofday(&tv1, NULL);

	for (double i = 0; i < 100000000; i++) {
		run(num);
	}

	gettimeofday(&tv2, NULL);


	printf("Total time = %f seconds\n",
		(double)(tv2.tv_usec - tv1.tv_usec) / 1000000 +
		(double)(tv2.tv_sec - tv1.tv_sec));

	float val1 = run(num);
```

```
printf("run1 output is: %f\n\n", val1);

// ************************************************************ SQRT 2

struct timeval tv3, tv4;
gettimeofday(&tv3, NULL);

for (double i = 0; i < 100000000; i++) {
       run2(num);
}

gettimeofday(&tv4, NULL);

printf("Total time = %f seconds\n",
       (double)(tv4.tv_usec - tv3.tv_usec) / 1000000 +
       (double)(tv4.tv_sec - tv3.tv_sec));

short val2 = run2(num);

printf("run2 output is: %d\n\n", val2);

// ************************************************************ SQRT 3

struct timeval tv5, tv6;
gettimeofday(&tv5, NULL);

for (double i = 0; i < 100000000; i++) {
       //run3(num);
}

gettimeofday(&tv6, NULL);

printf("Total time = %f seconds\n",
       (double)(tv6.tv_usec - tv5.tv_usec) / 1000000 +
       (double)(tv6.tv_sec - tv5.tv_sec));

double val3 = run3(num);

printf("run3 output is: %f\n\n", val3);

// ************************************************************ SQRT 4

struct timeval tv7, tv8;
gettimeofday(&tv7, NULL);

for (double i = 0; i < 100000000; i++) {
       run4(num);
}

gettimeofday(&tv8, NULL);

printf("Total time = %f seconds\n",
       (double)(tv8.tv_usec - tv7.tv_usec) / 1000000 +
       (double)(tv8.tv_sec - tv7.tv_sec));

double val4 = run4(num);
```

```
	printf("run4 output is: %f\n", val4);

	printf("\n\nFinished\n");

	// **************************************************************** FILE OUTPUT

	char output[50];
	snprintf(output, 50, "%f", (double)(tv8.tv_usec - tv7.tv_usec) / 1000000 +
		(double)(tv8.tv_sec - tv7.tv_sec));

	FILE *fp;

	fp = fopen("C:/Users/Niall/Desktop/STEM/sqrt4.txt", "a+");
	fprintf(fp, "\n");
	fprintf(fp, output);
	fclose(fp);

	return 0;
}
```

Above is sqrtControl.c

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

#include <quantum.h>

// https://stackoverflow.com/questions/2999075/generate-a-random-number-within-
range/2999130#2999130
int rand_lim(int limit) {
	/* return a random number between 0 and limit inclusive.
	*/

	int divisor = RAND_MAX / (limit + 1);
	int retval;

	do {
		retval = rand() / divisor;
	} while (retval > limit);

	return retval;
}

int main(int argc, char **argv) {

	int num = atoi(argv[1]);
	int width = quantum_getwidth(num);//atoi(argv[2]);
	int dec = atoi(argv[2]);
	double decPara = 1 / (double)dec;
	double i;
	double number = (double)num;

	srand(time(0));

	struct timeval  tv1, tv2;
	gettimeofday(&tv1, NULL);

	quantum_reg reg;

	//quantum_qec_encode(1, width, &reg);

	reg = quantum_new_qureg(0, width);

	quantum_set_decoherence(decPara);

	for (i = 0; i<reg.width; i++)
		quantum_hadamard(i, &reg);

	//quantum_print_qureg(reg);

	for (i = 0; i < reg.width; i++) {
		if (pow(2, i) < (number + number*1) && pow(2, i) >(number - number*1)) {

			quantum_sigma_z(i / 2, &reg);

			for (int j = 0; j < i; j++) {
				if (pow(2, j) + pow(2, i) < (number + number*1) && pow(2, j) +
pow(2, i) > (number - number*1)) {
```

```
                                    quantum_cnot(reg.width - i - j - 1, reg.width - i - j, 
&reg);
                                    quantum_sigma_z(i / 2, &reg);
                                    quantum_cnot(reg.width - i - j - 1, reg.width - i - j, 
&reg);
                             }
                      }

              }
       }

       //quantum_print_qureg(reg);

       for (i = 0; i<reg.width; i++)
              quantum_hadamard(i, &reg);

       //quantum_qec_decode(1, width, &reg);

       //quantum_print_qureg(reg);

       int result = quantum_measure(reg);
       double result2 = ((double)result + ((double)number / (double)result)) / 2.0;
       double actual = result2;
       for (int k = 0; k < 10; k++) actual = ((double)actual + ((double)number / 
(double)actual)) / 2.0;

       //quantum_qec_decode(1, width, &reg);

       //Number is: %i\t\t Width is: %i\t\tActual is %0.4f\t\t
       printf("%i\t\t%i\t\t%0.4f\t\t", num, width, actual);

       double error1 = (result2 - actual) / actual * 100.0;

       //Seed: %i \t\t Result 2: %f\t\tError is: %f\t\t
       printf("%i\t\t%f\t\t%f\t\t", result, result2, error1);

       quantum_delete_qureg(&reg);

       gettimeofday(&tv2, NULL);

       //Total time = %f seconds\t\t
       printf("%f\t\t",
              (double)(tv2.tv_usec - tv1.tv_usec) / 1000000 +
              (double)(tv2.tv_sec - tv1.tv_sec));

       // **** end quantum

       /*int seed = rand_lim(number - 2) + 1;
       float result3 = (seed + (number / seed)) / 2.0;

       printf("Seed: %i \t\t Result 3: %f\n", seed, result3);*/

       // **** end random

       double seed2 = log2(number);
       double result4 = (seed2 + (number / seed2)) / 2.0;
       double error2 = (result4 - actual) / actual * 100.0;
```

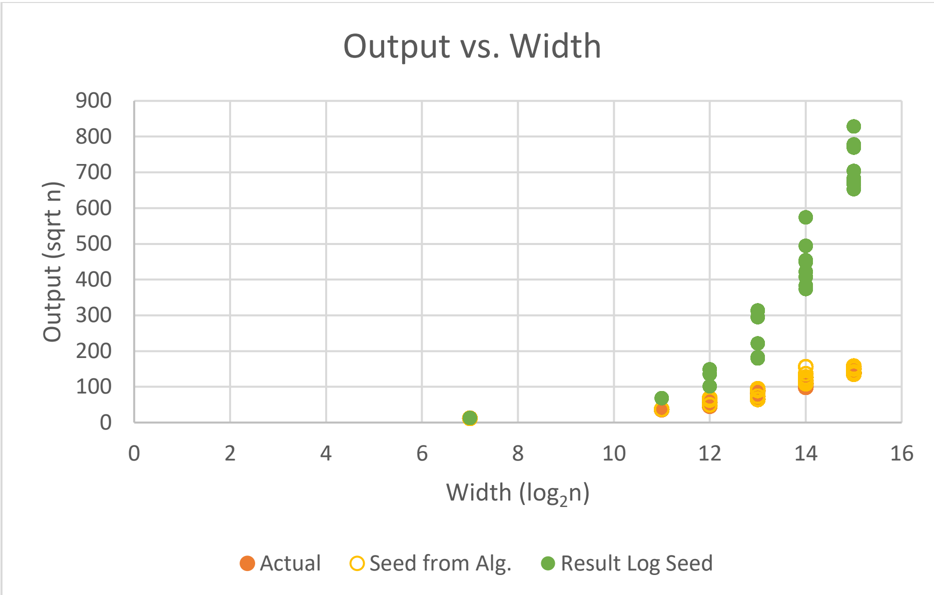
```c
        //Seed: %.2f \t\t Result 4: %f\t\tError is: %f\t\t
        printf("%.2f\t\t%f\t\t%f\t\t", seed2, result4, error2);

        struct timeval  tv3, tv4;
        gettimeofday(&tv3, NULL);
        double j;
        //for (j = 0; j < 1000000000; j++)
        log2(number);
        gettimeofday(&tv4, NULL);

        //Total time = %f seconds\n
        printf("%f\n",
               (double)(tv4.tv_usec - tv3.tv_usec) / 1000000 +
               (double)(tv4.tv_sec - tv3.tv_sec));

        //printf("dec is %f\n", decPara);

}
```

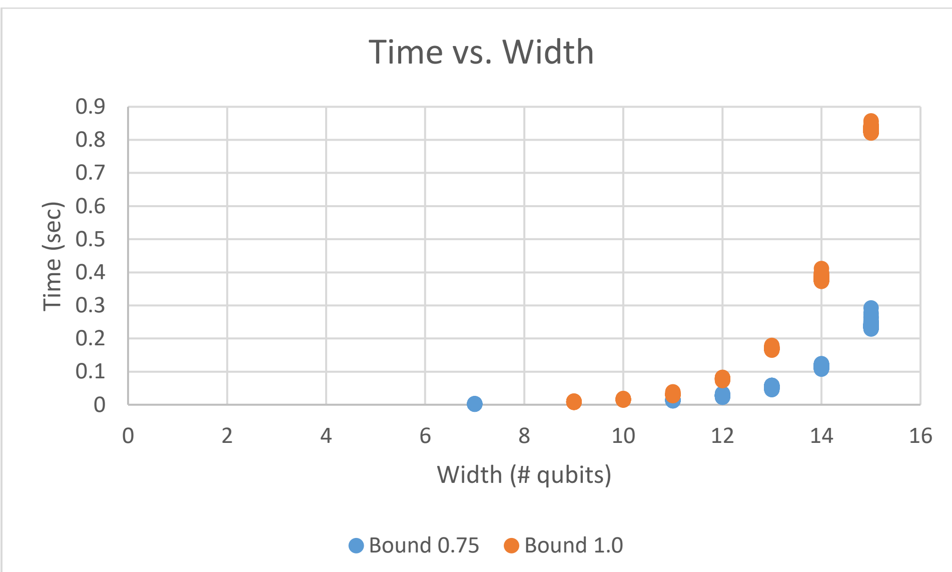
Above is fin1.c (Version 2 final, for use with script)

```bash
#!/bin/bash

COUNTER=100

while [  $COUNTER -lt 100000000 ]; do

        for i in `seq 1 30`; do

                RAND=$RANDOM

                for i in `seq 1 30`; do

                        ./fin1 $RAND $COUNTER >>
C:/Users/Jim/Desktop/Output/output$COUNTER.txt

                done

                echo -e "\n" >> C:/Users/Jim/Desktop/Output/output$COUNTER.txt

        done

        let COUNTER=COUNTER*10

done
```

Above is the script used for testing version 2.

10.5 EXTRANEOUS DATA





10.6 Poster Information

ENGINEERING PROBLEM

- Existing classical algorithms are not able to efficiently solve some problems. Therefore new quantum algorithms need to be developed to improve upon current computing capabilities and demonstrate quantum capabilities for future applications.

ENGINEERING GOAL

- The goal of this project was to engineer a quantum algorithm that efficiently solves the sum of square roots problem through square root convergence to a comparable or better precision than classical algorithms.

PURPOSE

The algorithm will be able to compute square roots to an arbitrary precision faster than a classical computer. This will allow calculations that require large amounts of square roots to be done faster.

Usage of Square Roots:

- Formulas - quadratic formula, distance formula
- Physical laws
- Euclidean Norm
- Graphics processing

METHODOLOGY

- The algorithm was designed by adapting current existing classical algorithms for square root calculations by adding quantum subroutines and modifying existing subroutines. Different quantum subroutines were added to try to achieve speedups. Multiple iterations of the algorithm were created using various methods.
- The first version (V1) used a quantum subroutine to test multiple seeds to achieve faster convergence in a modified version of the Babylonian method of computing square roots. The quantum subroutine used amplitude amplification to modify the amplitudes of the base states.
- The second version (V2) used a quantum subroutine to estimate the square root of a number by applying the Pauli Z transform at conditional intervals. Both algorithms were written in C with the usage of libquantum.

METHODOLOGY

- Square Root Approximations
  - Babylonian Method
    - Some number S of which to find the square root
    - Guess of x (random or not)
    - S = $(x+\text{error})^2$
    - Reduces to $x_1 = \frac{x+\frac{S}{x}}{2}$
    - Thus converges to a better and better approximation
  - Digit by digit method
    - Some number S of which to find square root
    - Calculate the greatest possible tenths place, then ones place, etc.

METHODOLOGY

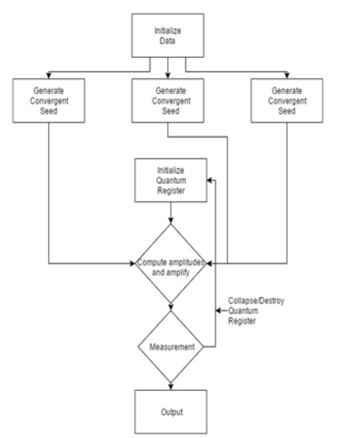


Figure 1. This diagram shows the process of convergence where amplitude amplification is used for a favorable probability distribution of results.

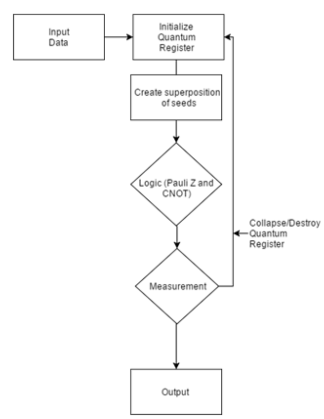


Figure 2. This diagram shows the process of by which an optimal seed is found through Pauli Z transforms and the CNOT gate.

RESULTS

Table 1. The outputs for version one (V1) over 10 trials and the average running time over those trials for $n = 12$.

Trial	Output
1	3.5
2	3.471622
3	3.471622
4	3.464286
5	3.5
6	3.5
7	4.173077
8	4.173077
9	3.464286
10	3.464286
V1 Time	
Average	0.087927

Table 2. The computational complexity of the different versions of the algorithm.

Computational Complexity	
Classical	$O(n)$
V1	$O(n)$
V2	$O(log_2 n)$

RESULTS

Table 3. The success rate of version 1 and the extreme configurations of versions 2 are compared.

Success Rate	
V1	0.4333
V2 - High Decoherence Parameter	
Bound 0.75	0.192
Bound 1.0	0.072
V2 - Low Decoherence Parameter	
Bound 0.75	0.939
Bound 1.0	0.939

Table 4. The percent error of V1 and the various configurations of V2.

Percent Error	
V1	4.490%
V2 - High Decoherence Parameter	
Bound 0.75	2405.594%
Bound 1.0	4012.910%
V2 - Low Decoherence Parameter	
Bound 0.75	6.642%
Bound 1.0	7.779%

DATA ANALYSIS

Figure 3. Percent error is shown against number size. Error tends to grow slowly for the quantum algorithm's seed and much quicker for a $\log_2$ seed.

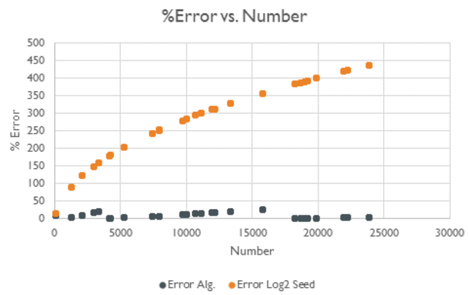
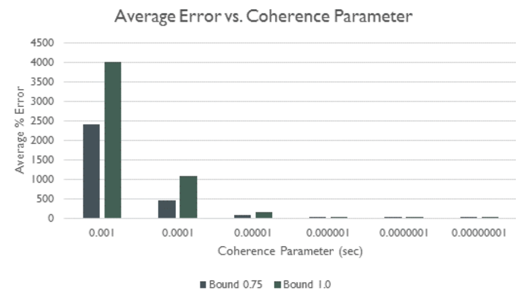


Figure 4. Average error against coherence parameter. The coherence (decoherence) parameter is "phase coherence time as seen experimentally. [The parameter] is the upper bound on the length of time over which a complete quantum computation can be executed accurately" (DiVincenzo, 1995).



DATA ANALYSIS

- The data shows that version two is likely superior to classical algorithms. With quantum error correction (QEC), longer coherence times could be achieved with higher success rates. The success rates in the second version indicate that optimizing seeding for convergence problems can fall under BQP and thus be solved efficiently.
- The second version of the algorithm showed superior results for a bound of 0.75 and for low decoherence times. Error went down significantly, as seen in Figure 4. Moreover, Figure 4 clearly shows the reduced error within a bound of 0.75 versus 1.0. A bound of 0.5 was also considered, but not picture due to extreme error and deviations.

Table 3. The matrix used to score the different versions of the algorithm.

Scoring Matrix			
	Weight	V1	V2
Efficiency	16	4	16
Success Rate	12	5.200	11.273
% Error	6	5.731	5.601
Total	34	14.931	32.875

CONCLUSION & EXTENSIONS

- Square root convergence is an interesting problem that can be modified to other convergence problems and it serves as a basis for some square root calculations. The quantum algorithm developed for optimizing square root convergence demonstrates the potential for quantum algorithms with bounded logic functions in convergence problems. Large computations and subroutines, as well as mainstream applications like graphics processing, would find efficiency speedups useful. Furthermore, quantum algorithms in regards to certain problems may inspire others to engineer new quantum algorithms for practical purposes.
- Further improvements could be done to achieve a higher success rate and QEC could be added to better deal with the effects of decoherence. Other bounds and higher amounts of qubits could be tested to see how such changes affect the output of the algorithm. Floating point representations could be added in as well to achieve better precision. Testing could also be done on a real quantum computer for a more lifelike test, rather than just simulation. Moreover, adapted versions of the algorithm could be used to efficiently solve other convergence based problems.

10.7 Notes

Last Updated: Monday, February 13, 2017

10.7.1 Timothy Prickett Morgan, Is Quantum Computing Set for an Investment Boom?

Original URL	http://www.nextplatform.com/2015/09/03/is-quantum-computing-set-for-an-investment-boom/		
File name of PDF	N/A		
Date written	September 3, 2016	**Date accessed**	September 15, 2016
Type of paper	Secondary source		
Goal of the paper	To discuss the possibility of an investment boom in quantum computing		
Major Findings	• Quantum computing necessitates a large amount of funding • D-Wave (quantum computing company) has received $139 million in funding • ~$2b dollars to build a quantum computer on a scale capable of implementing Grover's algorithm • Normal supercomputer needed for error detection and correction for a quantum computing, Intel thus has invested $50 million to research • Quantum processors may be usable as a coprocessor once economically viable • More funding and time still needed		
Notes on the paper	• Scalability challenges • Quantum annealer D-Wave vs. Universal quantum computer • QuTech research? • Many parties interested in investing • Intel research/ • How large does a parallel conventional computer for error detection have to be/ how much • Similar scaling to silicon based transistors, possible? • Quantum coprocessor rather than fully quantum • Many manufacturing challenges need solving		
Biases of the authors	Website is focused on high-end computing, may be trying to promote such developments for its own interests		
My opinions on the paper	Contains highly relevant information, has succinct explanations, outlines future implications of quantum computing and the funding for it that is ongoing Overall covers good information		
Follow up Questions and Ideas	• Funding on what scale exactly would be needed to develop more universal quantum computers? • Reducing costs of quantum parts once research is done • Look into applications of Grover's algorithm		
Keywords	Qubits, conventional computers, investment		

10.7.2 Noson S. Yanofsky, An Introduction to Quantum Computing

Original URL	https://arxiv.org/pdf/0708.0261v1.pdf		
File name of PDF	160915_NosonSYanofsky_IntroToQuantumComputing.pdf		
Date written	August 2, 2007	**Date accessed**	September 15, 2016
Type of paper	Review		
Goal of the paper	Introduce quantum computing topics and explanations of quantum mechanics		
Major Findings	• Simplified modeling and explanation of quantum architectures, quantum mechanics, and Deutsch's algorithm		
Notes on the paper	• Infinite dimensional quantum mechanics? • Further applications of basic toy models, look into • Based on text *Quantum Computing for Computer Scientists*		
Biases of the authors	None noted		
My opinions on the paper	Good introductory paper, uses more simple math and basic models to convey fundamental ideas about quantum computing		
Follow up Questions and Ideas	• Weighted graphs vs non-weighted? • Double-slit experiment?		
Keywords	Quantum computing, algorithm, qubit, quantum gates, computation, Deutsch's algorithm, quantum mechanics		

10.7.3 Shu-Shen Li, Quantum Computing

Original URL	https://www.ncbi.nlm.nih.gov/pmc/articles/PMC59812/?tool=pmcentrez		
File name of PDF	160925_Shu-ShenLi_QuantumComputing		
Date written	September 18, 2001	**Date accessed**	September 25, 2016
Type of paper	Review		
Goal of the paper	Introduce basic concepts, recent developments, and decoherence in possible quantum dot realization.		
Major Findings	• The uses and power of quantum computers are extremely potent and while a lot of work still has to be done, it is feasible		
Notes on the paper	• Many physical issues associated with quantum computers • Quantum computers most likely to be built using solid state technologies		
Biases of the authors	None noted		
My opinions on the paper	Contains topical information and in-depth generalizations of the field as it stood in 2001. Quantum dot section and explanation of generalizing Grover's algorithm are overly technical and difficult to understand however.		
Follow up Questions and Ideas	• Look into quantum dot qubits • What is the significance of decoherence timing? • Grover's algorithm and phase inversions (?) to solve issue of being probabilistic		
Keywords	QC (quantum computer), decoherence, quantum dot, qubit		

10.7.4 Marufa Rahmi, Basic Quantum Algorithms and Applications

Original URL	http://research.ijcaonline.org/volume56/number4/pxc3882868.pdf		
File name of PDF	160925_MarufaRahmi_BasicQuantumAlgorithmsAndApplications		
Date written	October 2012	**Date accessed**	September 25, 2016
Type of paper	Review		
Goal of the paper	To summarize major quantum algorithms and their applications		
Major Findings	• The applications of quantum algorithms as significant speedups both in general and as subroutines of more complex problems have good outlooks for future usage		
Notes on the paper	• Shor's and Grover's algorithms both have practical uses, with many various applications and spin-offs for different purposes • Simon's and Deustch/ Deustch-Jozsa's algorithms are more of proofs of quantum speedups • Database sorting, factorization/ prime factorization, searching over a set of possibilities, graph theory, RNG, pattern matching, are all significant applications of quantum algorithms		
Biases of the authors	None noted		
My opinions on the paper	Very information dense, good overviews for each section, although math gets very specific and hard to follow		
Follow up Questions and Ideas	• Look into further understanding technical aspects • Further implementations of quantum algorithms?		
Keywords	Qubit, Black box quantum computer known as an Oracle, Hadamard Transformation, Hadamard Gates, Superposition, Eigen value, Eigenstate		

10.7.5 Andrea Morello, Quantum Computing Concepts – Quantum Logic

Original URL	https://www.youtube.com/watch?v=YTNug9tQOzU		
File name of PDF	N/A		
Date written	N/A	**Date accessed**	September 28, 2016
Type of paper	Secondary source (video)		
Goal of the paper	Explain quantum logic		
Major Findings	• N/A		
Notes on the paper	• Quantum system can never lose information – some classical gates are not implementable on a QC, such as AND gate • Essentially have to be able to derive input based on output and type of logical operation • CNOT (controlled NOT) gate has no loss of information and is the fundamental gate of quantum computing • CNOT can be implemented physically through spin qubits, say given some A and B spin qubit, B can be affected by a magnetic field and the frequency at which B is affected is determined by the orientation of A in its vicinity (because of A's magnetic field), therefore the state of A affects B • This entangles A with B • CNOT can implement any logic function on a quantum computer		
Biases of the authors	None noted		
My opinions on the paper	Morello gives a highly effective presentation of quantum computing logic, easy to follow but also very informative		
Follow up Questions and Ideas	• Look into the implementations of CNOT to perform logic functions • Physical limitations of CNOT gates • Using CNOT gate to emulate AND gate?		
Keywords	CNOT, NOT, NAND		

10.7.6 Seth Lloyd, Quantum algorithm for solving linear equations

<table>
<tr><td>Original URL</td><td colspan="3">https://www.youtube.com/watch?v=KtIPAPyaPOg</td></tr>
<tr><td>File name of PDF</td><td colspan="3">N/A</td></tr>
<tr><td>Date written</td><td>February 7, 2011</td><td>Date accessed</td><td>October 2, 2016</td></tr>
<tr><td>Type of paper</td><td colspan="3">Original research (video lecture based on paper)</td></tr>
<tr><td>Goal of the paper</td><td colspan="3">Demonstrate how a quantum algorithm can prove to be superior to the best classical algorithm for solving linear equations</td></tr>
<tr><td>Major Findings</td><td colspan="3">• Exponential speedup, quantum algorithm runs in O (Log N, kappa) compared to O (N sqrt(kappa))</td></tr>
<tr><td>Notes on the paper</td><td colspan="3">• Problem: given a matrix A and a vector b, find a vector x such that Ax=b
• X = A^-1 b
• Can find A^-1 through diagonalization and finding eigenvalues and eigenvectors using quantum phase algorithm
• Quantum method provides exponential speedup</td></tr>
<tr><td>Biases of the authors</td><td colspan="3">None noted</td></tr>
<tr><td>My opinions on the paper</td><td colspan="3">Good information, well presented. Some parts are very technical</td></tr>
<tr><td>Follow up Questions and Ideas</td><td colspan="3">• Look into quantum phase algorithm
• Matrix diagonalization</td></tr>
<tr><td>Keywords</td><td colspan="3">Eigenvalues, eigenvectors, Hamiltonians, diagonalization, sparsity, inverse matrix, phasing</td></tr>
</table>

10.7.7 Bob Eagle, Quantum Mechanics Concepts: 1 Dirac Notation and Photon Polarization

Original URL	https://www.youtube.com/watch?v=pBh7Xqbh5JQ		
File name of PDF	N/A		
Date written	August 20, 2013	**Date accessed**	October 2, 2016
Type of paper	Secondary source		
Goal of the paper	Educational		
Major Findings	• N/A		

Notes on the paper	• Ket (denoted |a> for any a) vectors are complex column matrices represented by a_1 … a_n terms, where n is the number of dimensions
	• Bra (denoted <b| for any b) vectors are the conjugate transposes of ket vectors (transposed and complex conjugate) and vice versa
	• Bra-ket aka a bra times a ket, is called an inner product
	• Square complex matrix M with i rows and j columns, position marked by M_{ij}
	• Matrix conjugate is M_{ij}*
	• Matrix transpose is M_{ji}
	• Matrix conjugate transpose is M†, aka, M_{ji}*
	• Hermitian matrix H is when M = M†
	• Dot product of matrix and ket vector is new ket vector
	• Identity matrix I = matrix with 1s on diagonals, else is 0
	• Unitary matrix is U, UU* = I = U*U, aka, its conjugate transpose is also its inverse
	• Determinant of matrix $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$ is ad-bc
	• Det I (Identity) is 1
	• Trace of matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is a+d
	• H |a> = λ_a |a> where lambda sub a is a real number and H is a Hermitian matrix
	    o If this is true for some vector |a>, that vector is said to be an eigenvector, and the number is an eigenvalue
	    o Each Hermitian matrix has two separate eigenvectors which gives rise to two distinct eigenvalues
	• Hermitian matrix is measurable
	• Eigenvectors represent the states of the system
	• Eigenvalues are the results when you make a measurement		

Biases of the authors	None noted		
My opinions on the paper	Very easy to follow and has good explanations		

Follow up Questions and Ideas	• Why does Dirac's notation work and how did he come up with it?
Keywords	Bra, ket, vector, inner product, dot product, identity matrix, determinant

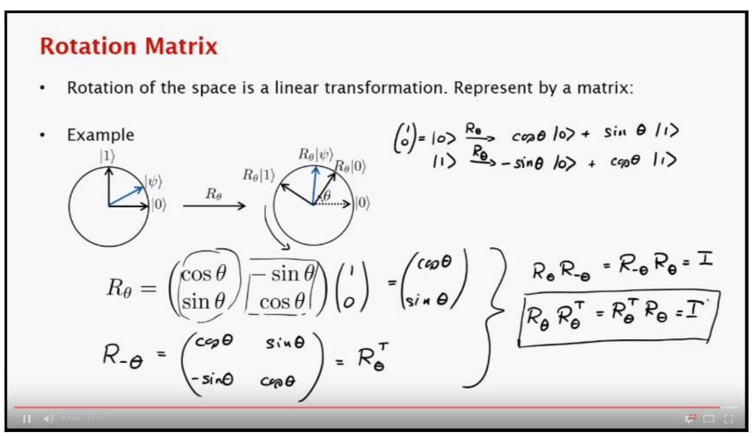
10.7.8 Steve Spicklemire, Lesson 38 Quantum Computing, Deutsch's Problem

<table>
<tr><td>Original URL</td><td colspan="3">https://www.youtube.com/watch?v=5xsyx-aNClM</td></tr>
<tr><td>File name of PDF</td><td colspan="3">N/A</td></tr>
<tr><td>Date written</td><td>December 5, 2012</td><td>Date accessed</td><td>October 3, 2016</td></tr>
<tr><td>Type of paper</td><td colspan="3">Secondary source (video)</td></tr>
<tr><td>Goal of the paper</td><td colspan="3">Educational</td></tr>
<tr><td>Major Findings</td><td colspan="3">N/A</td></tr>
<tr><td>Notes on the paper</td><td colspan="3">
Hadamard gate is essentially 90-degree rotation about y-axis
|0> = (|0> + |1>)/sqrt(2) |1> = (|0> - |1>)/sqrt(2)
Phase shift is an arbitrary rotation about z axis
|0> -> |0> |1> -> e^(iΦ)*|1>
Deutsch’s problem involves binary function f(x) with 2-bit input, thus 4 possible outputs
Question: Is f(x) constant or balanced?
Constant – outputs are independent of inputs
f(0) = f(1)
Balanced – inputs are dependent on inputs
f(0) = f~(1)
Classical needs two determinations, two tries of f(x)
Quantum needs one determination of f(x)
Uses Hadamard gates to force qubits into superposition
Uses f-CNOT, uses f(x) ⊕ y instead of x ⊕ y, can go through all possible combinations of the input because Hadamard gates put qubits into superposition
If the final superposition post f-CNOT is factorable in a certain way, that means that f(x) is constant
</td></tr>
</table>

	▪ You can factor the final superposition another way which will mean that f(x) is balanced ▪ Apply final Hadamard gate to x^{th} qubit and check the final x register to determine if balanced or constant • 1 – constant • 0 – balanced
Biases of the authors	None noted
My opinions on the paper	Well put together video, good explanations
Follow up Questions and Ideas	• Look into phase shifts and further understanding Hadamard gate
Keywords	CNOT, quantum register, entanglement, Hadamard gate, Deutsch's problem

10.7.9 Umesh Vazirani, Time Evolution of a Quantum State (Quantum Mechanics and Computation)

Original URL	https://www.youtube.com/watch?v=jy5XrK6vWmI		
File name of PDF	N/A		
Date written	May 20, 2014	**Date accessed**	October 6, 2016
Type of paper	Secondary Source (video)		
Goal of the paper	Educational		
Major Findings	• N/A		

Notes on the paper	• 3 axioms of quantum mechanics
	  ○ Superposition principle
	    ▪ Allowable states of k-level system: unit vector in k-dimensional complex vector space (Hilbert space)
	  ○ Measurement
	    ▪ Specified by choosing orthonormal basis
	    ▪ Probability of each outcome is the square of the length of the projection onto the corresponding basis vector
	    ▪ State collapses to observed basis vector
	  ○ Unitary evolution
	    ▪ State evolves over time via a rotation of the Hilbert space
	    ▪ I.E. applying unitary transformation (R_0) on $\lvert\psi>$
	    ▪ Rigid body rotation, angle b/w vectors are preserved
			

Biases of the authors	None noted		
My opinions on the paper	Good info, somewhat fast		

Follow up Questions and Ideas	• Get good understanding of probability square relation to coefficients vs actual probability
Keywords	Hilbert space, superposition, orthonormal, vector

10.7.10 Dave Bacon, CSE 599d - Quantum Computing Shor's Algorithm

Original URL	https://courses.cs.washington.edu/courses/cse599d/06wi/lecturenotes11.pdf		
File name of PDF			
Date written	None given	**Date accessed**	October 9, 2016
Type of paper	Secondary source		
Goal of the paper	Explain Shor's algorithm		
Major Findings	• N/A		
Notes on the paper	• Shor's algorithm works by first reducing factoring to order finding • Does this through Euclid algorithm • "We are given positive integers x and N with x < N and x coprime to N (they share no common factors.) Then the order finding problem is to find the smallest positive integer r such that x r mod N = 1. r is called the order of x in N" • Find order which can find factors through complex mathematics • Discusses potential implications of Shor's algorithm and quantum computers		
Biases of the authors	None noted		
My opinions on the paper	Interesting conclusion section, covers concept at a more in-depth level than I was looking for		
Follow up Questions and Ideas	• Look into Nielsen and Chuang "Quantum Computation and Quantum Information"		
Keywords	Order finding, factoring, Euclid's algorithm, gcd		

10.7.11 Grant Sanderson, Essence of Linear Algebra

Original URL	https://www.youtube.com/watch?v=kjBOesZCoqc&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab		
File name of PDF	N/A		
Date written	2016	**Date accessed**	October 2016, multiple dates
Type of paper	Secondary source (video series)		
Goal of the paper	Educational		
Major Findings	• Discussses the following • Chapter 1: Vectors, what even are they? • Chapter 2: Linear combinations, span, and bases • Chapter 3: Linear transformations and matrices • Chapter 4: Matrix multiplication as composition • Footnote: Three-dimensional linear transformations • Chapter 5: The determinant • Chapter 6: Inverse matrices, column space and null space • Footnote: Nonsquare matrices as transformations between dimensions • Chapter 7: Dot products and duality • Chapter 8: Cross products • Chapter 8 part 2: Cross products in the light of linear transformations • Chapter 9: Change of basis • Chapter 10: Eigenvectors and eigenvalues • Chapter 11: Abstract vector spaces		

Notes on the paper	• This series has helped formalize my understanding of linear algebra and its importance in the context of quantum mechanics
	• Quantum gates as linear transformations on qubit matrices
	• Quantum algorithm for finding answer to a linear system makes more sense now (See Seth Lloyd notes above)
	    ○ Ax = b where A are the coefficients, x is the column matrix encoding the variables (unknowns) and b is a column matrix encoding the constants
	    ○ Find x by finding A inverse, allows you to say $x = A^{-1} * b$		

Biases of the authors	Works for Khan Academy		

My opinions on the paper	Informative, focuses on conceptual understanding and big picture meaning of many concepts, highly useful for generalizing, esp. to generalize into QM.
Follow up Questions and Ideas	• Use of eigenbases in quantum computation

10.7.12 Scott Anderson*, Quantum Complexity Theory

*Notes compiled by MIT students

<table>
<tr><td>Original URL</td><td colspan="3">https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-845-quantum-complexity-theory-fall-2010/lecture-notes/</td></tr>
<tr><td>File name of PDF</td><td colspan="3"></td></tr>
<tr><td>Date written</td><td>Fall 2008</td><td>Date accessed</td><td>October 2016, multiple dates</td></tr>
<tr><td>Type of paper</td><td colspan="3">Secondary source</td></tr>
<tr><td>Goal of the paper</td><td colspan="3">N/A</td></tr>
<tr><td>Major Findings</td><td colspan="3">N/A</td></tr>
<tr><td>Notes on the paper</td><td colspan="3">
PDF 1:
Central problem in complexity theory is does P equal NP?
“NP (Nondeterministic Polynomial-Time) is the class of problems where, if the answer is “yes, “then there exists a proof that’s verifiable in polynomial time (though it might be very hard to find)”
P “contains all decision problems that can be solved by a deterministic Turing machine using a polynomial amount of computation time, or polynomial time” (Wikipedia)
Quantum mechanics, minus the physics, “it’s what you would inevitably get if you tried to generalize classical probability theory by allowing things that behaved like probabilities but could have minus signs”
Matrix multiplication to advance a computation, need to use unitary matrices to preserve norms and thus amplitudes (square of which gives us probabilities of each outcome)
“Conservation of probability requires unitary (i.e. the columns of the matrix should be orthogonal and have unit norm, or equivalently $A^{-1} = A^T$”
Implies reversibility, conserves information

PDF 2:
BQP, Bounded Error Quantum Polynomial Time, is the complexity class that deals with quantum computers
Defined roughly as set of problems efficiently solvable, with high probability, on a QC
Quantum state as “$\alpha_1 |1> + \ldots + \alpha_N |N>$, Where $|1>, \ldots, |N>$ are the basis vectors and $\alpha 1 \ldots \alpha N$ are complex numbers.”
Normalized, summation of a_1 thru a_N for $|a_i|^2$ equals 1
$|a_i|^2$ represents probability of $|i>$

</td></tr>
</table>

<table>
<tr><td></td><td>

Unitary evolution and measurement can be thought to be the only two principles of quantum mechanics
Different explanations for why measurement can alter the state (collapse of superposition), none accepted fully
Complex numbers are required to have continuous unitary transformations
Separable states are not entangled; non-separable states can be described to be entangled
Separable meaning that a multi qubit state which can be factorized into a sequence of single-qubit states
Quantum entanglement vs. classical correlation (does qm contradict relativity, i.e. is ftl communication possible?)
Bell inequality shows that quantum communication is not always possible classically, ftl communication however still not possible
Cannot duplicate quantum states, req’s nonlinear transformation
No cloning theorem

</td></tr>
<tr><td>Biases of the authors</td><td>None noted</td></tr>
<tr><td>My opinions on the paper</td><td>Lecture notes contain lots of information, but some explanations are skipped over, makes it hard to follow at times</td></tr>
<tr><td>Follow up Questions and Ideas</td><td>Questions regarding Bell’s inequality</td></tr>
</table>

10.7.13 Vladimir E. Korepin, Simple Algorithm for Partial Quantum Search

<table>
<tr><td>Original URL</td><td colspan="3">https://arxiv.org/pdf/quant-ph/0504157.pdf</td></tr>
<tr><td>File name of PDF</td><td colspan="3">161030_VladimirEKorepin_SimpleAlgorithmForPartialQuantumSearch</td></tr>
<tr><td>Date written</td><td>February 1, 2008</td><td>Date accessed</td><td>October 26, 2016</td></tr>
<tr><td>Type of paper</td><td colspan="3">Original Research</td></tr>
<tr><td>Goal of the paper</td><td colspan="3">To find a scenario in which the quantum search algorithm can be improved</td></tr>
<tr><td>Major Findings</td><td colspan="3">A performance improvement can be made under a modification for a partial search (i.e. finding the block that contains a target item rather than the target item itself)</td></tr>
<tr><td>Notes on the paper</td><td colspan="3">Problem considered as such: database of N items is separated into K blocks of size b = N/K elements, algorithm has to find the block containing the item of interest
0.34*sqrt(b) fewer iterations than quantum search algorithm with this algorithm
In large databases, usually only some aspect(s) of the entity are required, for example, the address of an entity rather than all the information
Split into global search for whole db, and local search within the target block, done in all blocks in parallel
Simple version of a previous partial search algorithm by Grover and Radhakrishnan, improves upon that algorithm for very many blocks
Tradeoff of precision for speed, i.e. only need the first few bits of the address of the target entity
Questions which partial search algorithm will be optimal for a finite number of blocks</td></tr>
<tr><td>Biases of the authors</td><td colspan="3">Grover developed the original Grover algorithm and the previous partial search algorithm mentioned in the paper, partially sponsored by the NSA</td></tr>
<tr><td>My opinions on the paper</td><td colspan="3">Summary and conclusion well defined and easy to understand, explanation seems to assume a lot of prior knowledge of Grover's algorithm</td></tr>
<tr><td>Follow up Questions and Ideas</td><td colspan="3">Selective inversion and inversion about the average</td></tr>
<tr><td>Keywords</td><td colspan="3">Inversion, selective, quantum algorithm, local and global search, database, amplitude</td></tr>
</table>