# Effects of the Development of Quantum Computing on NP-Complete Problems and the RSA Encryption Algorithm

Matt Fletcher

**Abstract**—Since the 1980's, the RSA Encryption algorithm has been used as a near foolproof method of encrypting sensitive data sent over the Internet. It relies on the relatively slow computing power of modern day computers as well as prime factorization being an NP-complete math problem . However, with quantum computers on the horizon of being invented, many cryptologists worry about the safety of encrypted data. Furthermore, many unsolved math problems are so because of their difficulty to solve them with current computational methods. This paper analyzes why the safety of encryption has both increased and decreased over the age of computers, as well as the effects of quantum computing on unsolved math problems.

**Index Terms**—Quantum computing, Cryptography, Encryption, P vs NP, RSA, Prime Factorization

## 1 Introduction

In 1955, mathematician John Nash penned a letter to the National Security Agency that would change the face of computing forever. TODO add citation In this letter, he presented a theory about the computational power required to find the solutions to various mathematical problems. In particular, he focused on the art of cryptography, and the length of time necessary to crack an encryption. Up to that point, computer scientists who created algorithms concentrated solely on the required time to solve a particular length of problem. Nash suggested that instead of fixating on the length of time necessary to solve a problem of a particular length, scientists should consider the rate of difficulty growth of the problem resolution time given the length of the inputs grew at a linear rate. Furthermore, in lieu of computation ti me, a scientist should focus on the number of computational steps required to solve the problem. A computational step is the number of state changes that the machine processes in carrying out the set of steps to solve the problem. The shape of this growth curve would help classify the difficulty of the problem. For example, the addition of numbers is a problem with a characteristic of linear growth. Doubling the number of digits to be added results in approximately double the computation time. Therefore, these problems grow at a linear rate[1]. However, problems such as multiplication grow at a slightly faster pace. Doubling the length of the inputs results in a growth of $2^2$ times the number of computational steps. Tripling the length of the input results in a growth of $3^2$ times the number of computational steps. This growth may be somewhat rapid, but can be expressed as a polynomial[2]. The exponent of the growth rate is always constant. Polynomial growth can therefore be displayed as some combination of terms in the form $n^k$, where n is a variable and $k$ is a constant.

Another type of problem is anything similar to cracking a password by brute force. For simplicity sake, assume the password is a PIN composed only of digits 0-9. In the worst case scenario, for a 1 digit PIN, the computer would take $10^1 = 10$ guesses. For a 2 digit PIN, the computer would take $10^2 = 100$ guesses. For a 4 digit pin, the computer would take $10^4 = 10\,000$ guesses. This is growing at a rate of $10^k$, where k is some constant. This growth pattern is known as exponential growth, where the variable is in the exponent. The resultant curve is extremely sharp. For context, if a password was able to consist of any numbers, letters, or special characters on a standard US keyboard, with only an eight character password, the number of computational steps required skyrockets up to $82^8$, or 2 with fifteen zeros following it.

### 1.1 Classifying P and NP Problems

Mathematicians had been noticing problems with similar growth patterns in all different fields of math and science. So, they decided to classify these problems.

#### *1.1.1 P-Type problems*

The first type is known as a problem with difficulty P. To find the largest number in a list of numbers, a computer must iterate through each element in the list of numbers one time[3]. Assume this uses $n$ computational steps. If the length of the list is doubled, the computer must go through $2n$ steps, or double the number of steps, in order to determine the largest element in the list. Tripling the length results in $3n$ computational steps. Therefore, increasing the list from $n$ elements to $k \cdot n$ elements will result in having to use $k$ times the number of computational steps[4]. This increase results in a linear growth. By computer standards, this is considered

1. Linear growth is still technically considered a polynomial growth of the form $n^k$ with $k = 0$.
2. A polynomial is an expression that can be written in the form $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$
3. For each element in the list, check if it's larger than the previous element. If it is, store that value as the largest value.
4. A sample Python script that does this calculation is in Appendix A.

a slow growth. Mathemeticians decided to classify these problems as type P problems, as they could be solved in **P**olynomial time. Not only are these problems easy to solve, but they are also easy to confirm if a potential solution is correct.

#### *1.1.2 NP-type Problems*

"Nondeterministic Polynomial time problems" (also known as NP problems) are not only difficult to solve, but also easy to check a solution to in polynomial time[5]. One of the most common examples of an NP problem is finding a subset of a list that satisfies a given requirement. This is often referred to as the hackey-sack problem. If someone is presented with a collection of small sacks with specified random weights, and then asked to find a subset of those sacks that result in a given weight, the only way to approach this problem with current computational methods is a brute force algorithm, guessing a random subset, and checking the result. Unlike the P-type problem of finding the largest number in a list, the number of computational steps required for this problem results in a much faster growth. Assume that initially, just 2 sacks are given. Call these $c_1$ and $c_2$. To figure out which combination yields the correct answer, one must try the following combinations:

- $c_1$
- $c_2$
- $c_1 + c_2$

This is only 3 iterations, which isn't terrible. Mathematically, the number of iterations required can be represented with the following expression, where $r$ is the number of sacks in the initial collection.

$$\sum_{k=1}^{r} {}^{r}C_k$$

However, the number of computational steps in this problem grows at an extremely fast rate. For 4 sacks in the initial collection, the number of iterations required jumps to $\sum_{k=1}^{4} {}^{4}C_k = 15$. Doubling the length of the input quintupled the number of computational steps. Through the use of Pascal's Triangle, one can find that the required number of computational steps for an initial collection of $k$ sacks is $2^k - 1$. For an idea of the rate of growth, a collection of 30 sacks would require over 1 billion computational steps.

Yet another example of an NP problem is the factoring of a number into its prime factors. In his 1801 book *Disquisitiones Arithmeticae*, mathematician Frederick Gauss proved that any number has exactly 1 prime factorization. It is extremely easy to check if a given factorization for a number is correct (the multiplication of said numbers is a P-difficulty problem). However, in order to find the unique factorization for a number, the only method currently known is guessing and checking every single factor for that number. As the length of the target number grows, so does the length of computation time, but the computation time grows exponentially. As the growth of computation time is not of a polynomial form, the factorization problem is considered an NP-difficulty problem. But what application does this have in the real world?

5. "Easy" and "difficult" in this case mean that they are respectively solvable and not solvable by a computer in a reasonable amount of time.

### 1.2 RSA Encryption

As a shopper enters their credit card details for an online store website, the shopper is assured that their credit card details and other personal data are protected if the green lock icon is present in their browser. The only reason this lock icon has any significant value is thanks to two major factors: 1, Prime factorization is classified as an NP problem, and 2, the difficulty of cracking this encryption (known as RSA encryption) is extremely difficult and time consuming using current computation methods. In 1978, three computer scientists, Ron Rivest, Adi Shamir and Leonard Adleman, publicly released an encryption algorithm titled after their names, RSA. The basis of this encryption technique[6] is two secret prime numbers (known as private keys) multiplied together to form a public key. To decrypt the message, the two original numbers must be known. Finding these original two numbers is only possible through repetitive brute force calculations[7]. For an idea of the size of these numbers, the public key length for most banking systems nowadays is over six hundred digits long[8]. To complete these calculations with current methods and computing technologies would require somewhere on the order of 6.4 quadrillion years to try all possible combinations.

### 1.3 Evolution of Computational Methods

The phrase "with current methods" is used throughout this paper, in the context of methods used to solve a problem. This aligns with a computing device known as a Turing computer to solve a problem. In 1936, English computer scientist and cryptologist Alan Turing proposed the idea of a theoretical computing machine that operates off of a memory tape with infinite length divided into discrete cells. Each cell contains a basic instruction. The reading head above the tape identifies the value of the cell on the memory tape, then moves the tape either to the left, to the right, or terminates the program. Although this may seem very simple, determining if an algorithm can be run on a Turing machine constitutes one of the most important problems in the field of computer science. A Turing computer is the most basic computer possible, taking in bits one at a time, and outputting a result based on the bit. These computers are powerful. However, they are insufficient for solving NP problems. Because the difficulty and number of computational steps for NP-type problems grows with non-polynomial growth, the computation time grows rapidly to an unmanageable amount. However, this challenge will change with the introduction of quantum computers.

6. Please note that this is a highly simplified explanation of the encryption algorithm. To describe this in full detail would take far more in-depth explanations and mathematics than would be appropriate for this paper.

7. Although some algorithms (such as Shour's Algorithm) exist to reduce the work slightly using methods to eliminate groups of guesses, it is still a polynomial reduction on an exponential growth, which means the reduction is almost unnoticeable.

8. A 617 digit long public key is the key length for 2048 bit encryption.

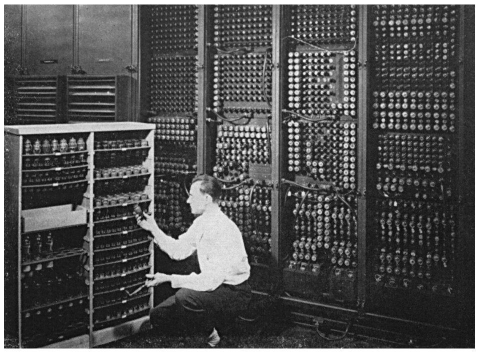
Fig. 1. An ENIACS computer, which operated off of vacuum tubes.

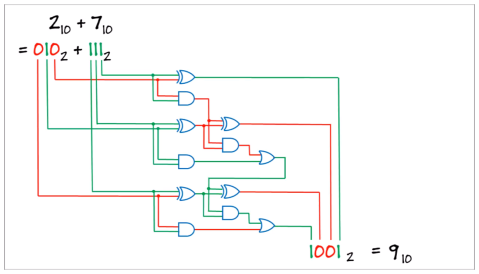

Fig. 2. A sample setup using logic gates to add 2 numbers.

Computers have evolved drastically over the decades. In first generation computers, or computer technology designed from 1939 to 1954, computers operated on vaccuum tubes. Even a basic computer with hardware specifications orders of magnitude weaker than a modern day pocket calculator would occupy an entire room, and cost hundreds of thousands of dollars. These vacuum tubes, due to the excess heat they created, were extremely unreliable and were prone to failure every few hours. Computers soon transitioned to using microswitches known as transistors. Compared to the vacuum tubes, transistors could be packed far more densely into the same area. Due to Moore's law[9], the processing power of computers grows at an exponential rate. Eventually, transistors were changed to a special type of transistor known as a MOSFET[10]. These operate on the same underlying principle as a regular transistor, with a binary existance of a 1 or a 0 for on and off. Due to their low power usage, MOSFET's could be packed into extreme densities. Eventually, these transistors were integrated into the layers of silicon in the processing chip, which allowed the size of an individual transistor to be in the tens of nanometers[11] However, regular computers will soon reach the limit of how small these transistors can get. Current CPU architecture is based off of 14 nanometer transistors TODO cite. The next generation of transistors will reach 10 nanometers. Computer scientists predict that transistors will reach their physical limit size sometime in the mid 2020's. This limit is due to the fact that the transistors are approaching the size of atoms. TODO CITE

## 1.4 How Transistors Run A Computer

How does a microscopic electronic component that can only be in 2 states operate a device that can perform billions of calculations per second? A transistor is a switch that can either be on or off to represent a 1 or a 0, respectively[12] . It will exist in exactly one of these states at any given time (in other words, a transistor cannot exist in a quasi-on state). One transistor by itself cannot do much. However, combining the transistors can result in extremely powerful calculations. There are 7 base combinations of transistors: AND, OR, XOR, NOT, NAND, NOR and XNOR. For an AND gate, the input is a pair of bits, and the output is a single bit. When both input bits are on, the output is is on, but otherwise, the output is off. Another type of gate is the OR gate. This has 2 inputs and 1 output. If either of the inputs is true, or both are true, the output is true. Only if both are false is the output false.

For a NOT gate, the input is a single bit and the output is also a single bit. The output is simply the opposite of the input. The output is simply the opposite of the input. By placing a NOT gate on the output of an AND gate, a NAND gate is formed, which outputs the opposite of the AND gate.

One other type of gate is the XOR gate, also known as the eXclusive OR gate. Unlike the OR gate, the XOR gate will only show true if exactly 1 input is true. This is also known as the either/or gate.

The combination of these logical gates results a boolean table known as a Truth Table. This allows for very low level instructions, such as STORE, LOAD, ADD, or COMPARE. See Figure 2 TODO above for a sample addition machine.

These instructions can be used to address the data stored in RAM. The language used for these instructions is called 'assembly language' or 'machine code'[13]. A quantum computer operates uses a slightly different principle. Instead of operating on a physical piece of hardware (the transistor) that produces a bit, it operates by observation of a photon that represents a qubit[14]. A qubit is a special type of bit that can either represent a 1, a 0, or a superposition of a 1 and a 0. But how can something exist in both a 1 and a 0 at the same time? To understand this, one must first understand the principle behind Schrödingers cat.

The infamous Schrödinger's cat experiment is performed as follows. Imagine a living cat inside a sealed bunker, with a sealed beaker of poisonous material, a Geiger counter attached to a hammer, and a small piece of radioactive material. The radioactive material has a 50% chance of having one of its atoms decay in the next hour (which would set off the Geiger counter, smashing the hammer into the vial of gas and releasing it into the bunker, killing the cat).

9. Moore's law is based off of Intel co-founder Gordon Moore's observation that the density of transistors on microprocessors had approximately doubled every year since the early age of computers. Moore's law states that this trend will continue every year into the future.

10. Metal Oxide Semiconductor Field Effect Transistors

11. For example, the Intel i7 Quad Core CPU packs 731 million transistors into a board only 0.63" by 0.63".

12. Most computers use a 5V charge to signal an ON, and a 0V charge to signal an OFF TODO cite

13. This language is referred to as a low-level language, while more common languages like Python, FORTRAN, C/C++, and Java are referred to as high-level languages. This refers to the logical distance between the syntax of the language and the machine language, along with the amount of abstraction used in the language.

14. Qubit is a juxtaposition of QUantum BIT

As long as the bunker is sealed and unobserved, the cat has a 50% chance of being dead and a 50% chance of being alive. When the bunker is opened, however, the cat must either be dead or alive. The instant before the bunker is opened, the cat exists in a quantum superposition of both dead and alive.

Just like the cat, as long as the qubit is unobserved, it exists in a quantum superposition both a 1 and a 0. As soon as the qubit is observed, though, it "snaps" to either a 1 or a 0. Another way of thinking about this is like a coin that is flipped and is spinning in midair. At any point in the air, it is impossible to figure out if it is heads or tails. However, smashing a fist down on top of the coin will force it to go to either a heads or a tails orientation. TODO explain this better. But how does this help a quantum computer work faster than a regular computer?

### 1.5 How a Quantum Computer can solve NP-type problems

The operations of a computer can be compared to an office worker sitting at their desk. The storage (hard drive) is comparable to a file cabinet for long term storage. The Random Access Memory (RAM, also known as memory) is comparable to the top of the desk, where papers can be placed when they are being worked on. Finally, the CPU is comparable to the person sitting at the desk, actually performing the tasks. The person can only work on something on the top of their desk, as trying to read a piece of paper sitting in a file cabinet would be awkward and slow[15]. Therefore, the person pulls an item out of their file cabinet and places it on their desk. The desk can only hold so many papers, so when the desk runs out of room, some papers have to be put back into the file storage.

Similiarly to the desk analogy, a CPU instructs the hard drive to load certain pieces of information from the hard drive into the RAM. When the computer runs out of free memory, the CPU removes information from the RAM that isn't being used. Unlike the desk analogy, however, the CPU does not directly operate from the RAM. Instead, the CPU loads a small bit of information from the RAM into the register of the CPU. The register is a very small amount of extremely fast storage directly tied to the CPU. For example, an Intel i7 CPU has 64 bit, 80 bit, and 120 bit registers[16]. Once in the register, the bits go through the transistors discussed earlier, where the logical gates perform instructions based on the data going through them.

A classical computers "bits" can either store a 1 or a 0. This is relatively efficient, but storing an N-bit number requires N bits. In other words, storing a 64 bit value takes 64 bits, and storing a 256 bit value takes 256 bits. The transistors in the CPU only recognize the 2 states, and operate serially (in series) on each bit.

A quantum computer, on the other hand, uses bits that exist in a state that is either a $|1\}$, $|0\}$, or a superposition that can be represented with the expression $a|1\} + b|0\}$. The state of the bit can be expressed using complex vector addition. However, due to a quantum theory known as quantum entanglement, the state of 1 qubit affects the other qubits in the system. Two particles can be entangled even if they are separated by physical space. A very critical point about this: quantum computers do not measure the values of the individial qubits. Instead, they measure the 'correlations', or interactions, between the qubits. Just like Schrödinger's cat, as soon as the qubits themselves are observed, the system changes from its original state. The cutting link to success came when two computers scientists, David Deutsch and Richard Jozsa, determined an algorithm that allowed for a computer to deduce what the original state of the qubits was most likely to have been before observation. Think of it like a house of cards that cannot be observed without them falling into a pile. The Jozsa-Deutsch algorithm allows for a comupter to determine the most likely original state of the cards.

This ability is what makes a quantum computer so powerful. Instead of operating serially on a stream of bits, the computer can operate in parallel, on multiple bits simultaneously.

15. Interestingly enough, trying to work on a paper while the paper is still in the file cabinet can be compared to a pagefile, where the CPU pulls data directly from storage.

16. Compare these register sizes to the RAM in an average desktop, which is measured in gigabytes.

## 2 Quantum Computing Pros and Cons

What are the advantages and disadvantages that come with the rise of quantum computing?

### 2.1 Advantages

The United States Military expresses a large amount of interest in quantum computing, primarily for its use in cryptanalysis, or code-breaking. Because quantum computers can analyze large numbers of bits simultaneously, they are excellent for problems that require brute force calculations.

Another use of quantum computing arises in the use of autonmous vehicles and route generation. Finding the shortest route between a set of points is considered an NP-hard problem [17]. In math terms, this problem is attempting to find the most efficient Hamiltonian Cycle to take through a set of N nodes. In more simplified terms, this is finding the shortest route between a set of random points. This has a real life application, however; finding the shortest route between a set of points is nothing more than a navigation application such as Google Maps. The use of quantum computing in similar navigation apps would allow for more accurate calculations in arrival time, as well as more accurate methods of finding the fastest route possible.

One highly important section of the field of quantum computing is its application in the field of artificial intelligence. According to IBM, "Making facets of artificial intelligence such as machine learning much more powerful when data sets are very large, such as in searching images or video." Machine learning such as IBM's Watson relies on searching a vast database of information to calculate results. Through the use of quantum computing, these database searches can be completed on orders of magnitude faster than would be possible with standard computing methods. TODO add citation

17. See Appendix C

Yet another application that quantum computing will change dramatically is sociopoliticial and socioeconomic fields. In sociopolitical fields, especially when it comes to political elections, races are won largely through the use of algorithms to determine which groups of people stand the most chance of being swayed to vote for that particular party. Current algorithms, although they are relatively accurate, can only go so far in how accurate they can be. But with the use of quantum computing, algorithms can be made that will be able to precisely target groups.

What about applications that could potentially save lives? Current meteorological prediction techniques TODO add more.

### 2.2 Disadvantages

One of the largest problems with a quantum computer is that it is still not feasible. The power required to run the processor combined with the temperatures measured in milliKelvin to cool the hardware results in astronomical costs. Furthermore, the difficulty of actually setting up the hardware would be prodigiously difficult.

What about encryption? RSA encryption relies on the fact that computers are unable to brute force calculate the private keys. If a quantum computer is produced by a malicious source, then the effects could be devastating. Private keys could be calculated almost instantaneously, allowing for hackers to have access to data that is thought to be private.

TODO keep expanding this.

## 3 Conclusion

ONce a reliable quantum computer is available, the face of mathematics will be changed forever. Math and science problems that would not have been possible before will now be able to be accomplished in mere seconds.

## Appendix A
## Sample Code for Largest Element

Assume the list of numbers is stored in a list named *num_list*. The following is written in Python. Note how there is only 1 FOR loop.

```
greatest_element = num_list[0]
for element in num_list:
        if element>greatest_element:
                element = greatest_element
return element
```

## Appendix B
## Proof of iteration count for sacks

Start with a test case, then prove for a general solution. Let the number of sacks in the initial collection be 4.

To figure out the total possible number of sacks, use a case by case scenario. Every final count of sacks will either contain 1, 2, 3, or 4 sacks.

- **Case 1**, picking 4 sacks.
  There are 4 possible choices for the first sack, 3 possible for the second, 2 for the third, and 1 for the fourth. Mathematically represented, there are $^4P_4 = 4! = 4\cdot3\cdot2\cdot1$ possible combinations. However, as the order the sacks are picked in does not matter, divide by 4!. The mathematical represenatation of this compensation is $^4C_4$.
  $$\frac{4!}{4!} = 1$$
  . This makes sense, because there's only 1 way to pick a collection of all 4 sacks.
- **Case 2**, picking 3 sacks.
  With the same logic as the previous case, there are $^4P_3 = 4 \cdot 3 \cdot 2$ possible unordered combinations. However, each choice of 3 sacks can be arranged in 3! ways, so therefore divide the total number of collections $(4!)$ by the number of arrangements of each $(3!)$.
  $$\frac{4!}{3!} = \frac{24}{6} = 4$$
  .
- **Case 3**, picking 2 sacks.
  $$^4C_2 = \frac{^4P_2}{2!} = 6$$
- **Case 4**, picking 1 sack.
  $$^4C_1 = \frac{^4P_1}{1!} = 4$$

Adding each of these together results in $1+4+6+4 = 15$.
The number of possible combinations for each case is not a random number. These numbers actually come from the Binomial Coefficients of Pascal's Triangle.

```
n = 0:                 1
n = 1:              1     1
n = 2:           1     2     1
n = 3:        1     3     3     1
n = 4:     1     4     6     4     1
```

The row $n = 4$ has the same values as each of the cases discussed above. (The only number that does not show is the first 1, as this represents the number of ways to pick zero elements from the list of initial sacks. As this is not a valid solution to the problem, it is ignored.)

## Appendix C
## Traveling Salesman NP Proof

A simple explanation behind this statement: To find the shortest route between a set of nodes $N_1, N_2, \cdots N_Q$ of size Q, start at any node $N_1$. At this point, $Q$ guesses have been checked. Now, there are $Q-1$ points left to try. After another point $N_2$ has been selected, $Q(Q-1)$ guesses have been attempted. After another point, the number of guesses goes to $Q(Q-1)(Q-2)$. This is increasing at a rate of $Q!$, which is a non-polynomial growth.

## Appendix D

Insert Appendix text here

## Acknowledgments

The authors would like to thank Dunkin Donuts, Reddit ELI5, Wikipedia, and whoever invented copy/paste.

## References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.