HIGH SCHOOL EDITION
Journal of Student Research

Volume 11 Issue 3 (2022)

Comparing and Reviewing Modern Primality Tests

Ishaan Ganti[1] and Dr. Ethan Hutt[2#]

[1]Mission San Jose High School
[2]University of North Carolina at Chapel Hill
[#]Advisor

ABSTRACT

Primality tests refer to algorithms that determine whether a number is prime or composite. These tests are essential to modern encryption algorithms, including the widely used RSA public key encryption algorithm. However, with so many different primality tests out there, choosing the correct one for a given application can be challenging. This paper provides an overview of modern primality tests, detailing the differences between different types of tests and when one test may be preferable to another. Furthermore, implementations of popular primality tests are written and compared to one another graphically to better understand their differing performances. Lastly, we look at next steps for the field of primality tests due to the rise of quantum computing, which could serve as a means of creating even better primality tests.

Significance of Primes and Primality Tests

Prime numbers, numbers that can only be divided by one and themselves, are arguably among the most unique and powerful group of numbers in mathematics. Prime numbers are at the heart of the field of number theory, and they are key to cryptography. This is why much effort has gone into creating efficient algorithms for finding them---primality tests.

Primality tests are as important as they are not just because prime numbers are important, but also because big prime numbers are important. The prime numbers used in fields like cryptography are typically at least 40 digits long. To generate[1] primes this big, random odd numbers of a specified size are generated and then tested for primality. If a brute force method were used to test the primality of these big numbers, this process would be much too time consuming. These brute force methods generally involve checking an integer's divisibility by every integer greater than one preceding it. If it is divisible by any of these values, it is composite. Otherwise, it is prime. While this method can be refined, it still is not nearly fast enough for practical usage. The tests used today make use of results in number theory and group theory to drastically reduce running time.

Applications of Prime Numbers

The applications of prime numbers are extensive. They are at the heart of number theory, they are used in many pseudo-random number generators (such as Blum Blum Shub), and they are important in hashing. However, the most important and direct usage of prime numbers in a field has to be in encryption. Many of the most used encryption algorithms depend on the generation of very large prime numbers. For example, RSA, a popular encryption algorithm, depends on two keys: a private key and a public key. The public key is the product of two primes, and generally must

[1] Prime generation generally is performed by generating random odd numbers of a specified size and testing each number for primality until a number passes a give test. So, any mentions of prime generation imply the usage of a primality test.

be at least 1024 bits for decent encryption. This means that the two primes that are factors of the public key must both be approximately 512 bits. Generating primes of this size is not a simple task.

Types of Primality Tests

There are two contemporary types of primality tests: deterministic tests and probabilistic tests. Deterministic tests are tests that state with certainty whether a number is prime. The reason that not all primality tests are deterministic is due to time complexity involved in determining whether the number is prime. Time complexity for all primality tests will be provided using Big O notation.

In contrast, probabilistic tests return whether a number is prime or not with a degree of uncertainty, typically bounded by a known error. These tests tend to rely on randomness. Furthermore, the error of these tests can oftentimes be reduced through multiple rounds of the same test, or with a combination of multiple probabilistic tests. Unfortunately, some probabilistic tests, such as the Fermat primality test, have composite numbers that will pass the test no matter how many rounds of the test are done. Numbers like this are referred to as pseudo-primes. For applications where the generation of a prime every single time is necessary, probabilistic tests should then be used with caution if at all.

The Issue of Time Complexity

Most deterministic tests are too slow for use in practical applications. Most of them run in exponential time, except for AKS, an algorithm that will be discussed later. Other tests like it relied on conjectures such as the Riemann hypothesis.

Initial Deterministic Tests

The most used deterministic primality tests were only found after the 1980s. This came with the creation of the APR-CL test, which was later improved for modern implementations. Near the end of the 20th century came the creation of ECPP, arguably the most powerful deterministic primality test today.

But what came before all of this? Let's start with the brute force tests that were previously mentioned, involving checking a number's divisibility by every number less than it. As readers may infer, this test is horrendously slow. For instance, here's the runtime of a python implementation of this algorithm for numbers just larger than 20 bits:

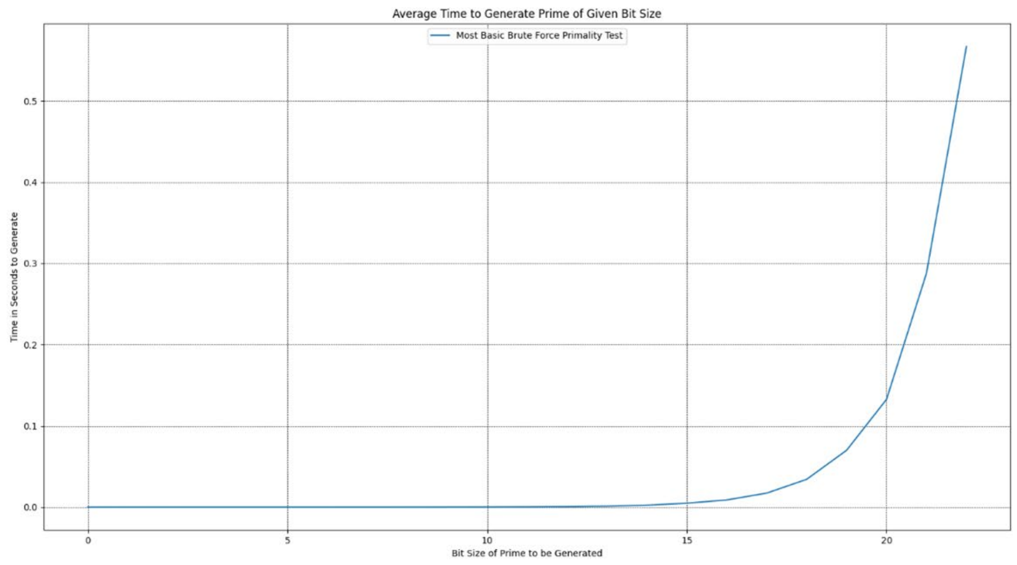

Figure 1. A plot of basic primality test runtime for increasing bit size requirements

Clearly, due to the exponential runtime of this algorithm, this won't be feasible for larger values of n. Major improvements can be made to this algorithm by only checking odd numbers up until $\sqrt{n}$. Here are the initial and revised algorithms compared to one another over a larger interval:

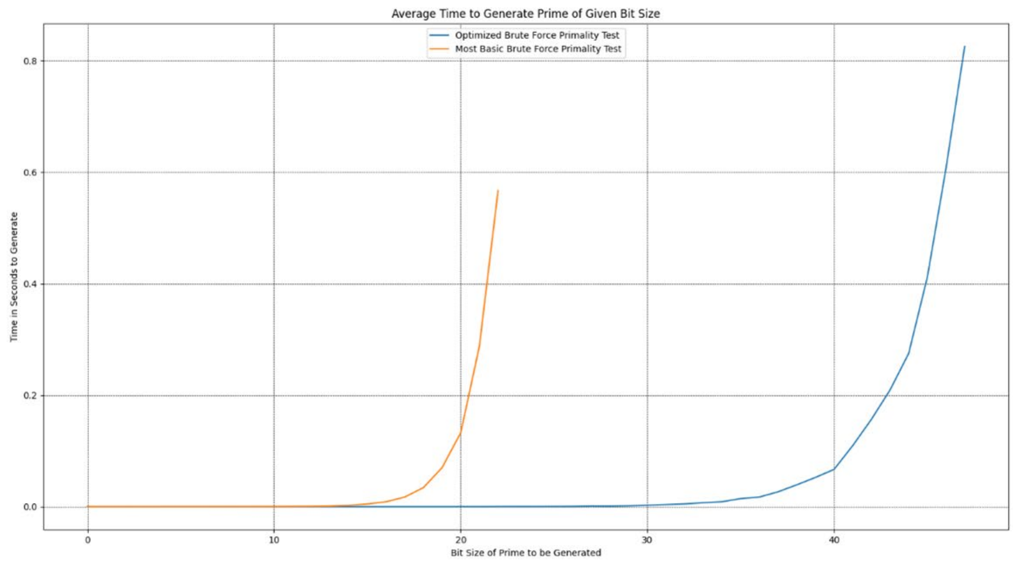

Figure 2. A graphical comparison of brute force primality tests

Even though this algorithm holds on for a little bit longer, its runtime still increases far too quickly to have any sort of practical usage.

Results in number theory, such as Wilson's theorem (which states that $(p-1)!+1 \equiv 0 \pmod{p}$ if and only if p is prime), have also been used to create deterministic tests. However, almost all these algorithms are far too slow for practical usage. Except for a few.

Practical Deterministic Tests

Elliptic curve primality proving and APR-CL are the two most used deterministic primality tests. APR-CL runs unconditionally in $(\log n)^{O(\log\log\log n)}$.This time complexity is rather interesting as it is technically exponential, but the $(\log\log\log n)$ exponent term grows so incredibly slowly that it is far faster than many other algorithms in practice.

Even still, the usage of APR-CL is limited to some software implementations. Why is this? Well, this is because ECPP runs a bit faster than APR-CL implementations in practice and it issues a primality certificate upon deeming a number to be prime. A primality certificate allows for a number to be quickly and independently verified as a prime, making it especially powerful for keeping track of large primes (e.g. for prime searching). The primes involved in prime searching, for some context, are typically larger than 20,000 digits long. In comparison, the primes used in cryptography are generally between 100 and 700 digits long.

The most significant drawback, however, of using ECPP is that its worst-case time complexity is unknown. In practice, its time complexity has been shown to be $\mathrm{O}((\log \mathrm{n})^{5+\epsilon})$ for some $\epsilon > 0$.

AKS: The Groundbreaking Deterministic Test

A third deterministic test is known as AKS. AKS, while revolutionary, does not serve much practical purpose anymore. Its significance is due to it being the first deterministic algorithm that runs in polynomial time without the use of any unproven conjecture (or something else along similar lines). It is based on the following relation:

$$(X+q)^n \equiv X^n + q \pmod{n}$$

which holds if and only if n is prime. It stems from Fermat's little theorem. A more intuitive explanation of this condition is in Appendix A. The time complexity of the original algorithm was calculated to be $\tilde{O}(\log(n)^{12})$.

The test is rather obsolete now. This is because some probabilistic tests, such as the Baillie–PSW test, are known to be deterministic for values under 2^{64}. For values larger than this, ECPP and APR-CL are far faster than AKS. This is evident from the following bit size versus time graph with AKS, APR-CL, and brute force implementations.

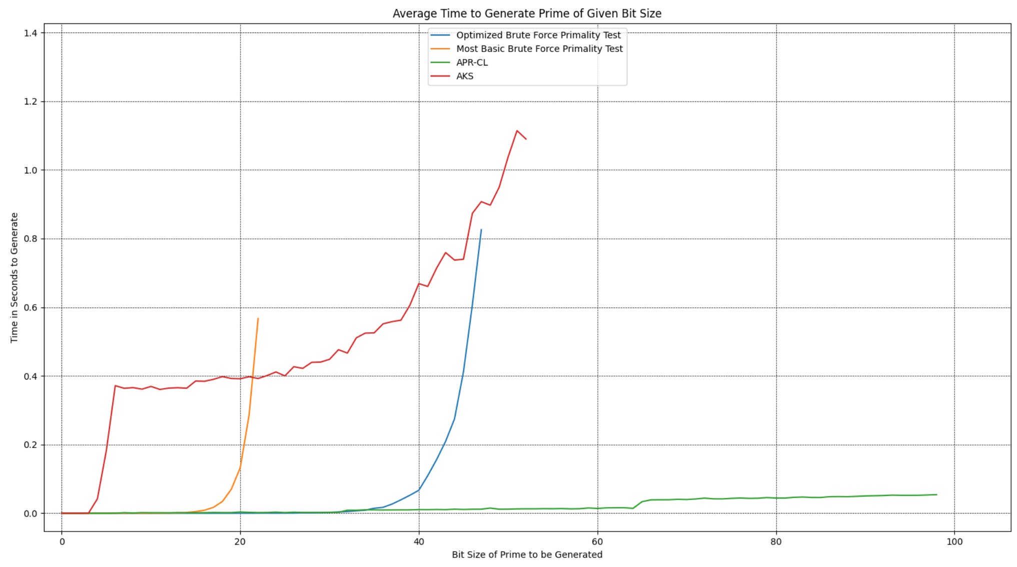

Figure 3. A graphical comparison of AKS and APR-CL with the brute force primality tests for reference

Even though AKS is outdated now, there have been significant improvements to the algorithm since its creation. For example, Agarwal, Kayal, and Saxena, the creators of the AKS primality test, showed that if a well-known conjecture pertaining to the distribution of Sophie Germain primes is true, then the time complexity of AKS can be reduced to $\tilde{O}(\log(n)^6)$. A second version of the paper brought the complexity of the algorithm down to $\tilde{O}(\log(n)^{7.5})$. Additionally, in the following months, mathematicians Carl Bernard Pomerance and Hendrik Willem Lenstra Jr. synthesized a version of AKS that ran in $\tilde{O}(\log(n)^6)$ through the usage of Gaussian periods.

Probabilistic Primality Tests

Primality tests that have a possibility of outputting an incorrect result are known as probabilistic primality tests. In general, these are the tests that are used the most in practice. The main reasons for this are ease of implementation and more importantly, speed. Probabilistic primality tests are much, much faster than deterministic tests. And, most of the time, the error of these tests can be minimized to a value that makes it nearly negligible.

Additionally, when these tests are used for practical purposes (e.g. encryption), the practical software that requires the generation of a prime number oftentimes has a sort of fail-safe. In the case of RSA (an enhanced explanation is in Appendix B), for example, if a composite number is generated for half of the public key, the private key will most likely be generated incorrectly. This makes decryption near impossible --- rendering it near impossible to recover the original message. However, for extraordinary composite selections of the public key components, the encryption still may work. This is especially bad because then the public key will be easier to factor, making the information more at risk to being decrypted by a malicious third party.

The Miller-Rabin Algorithm

The Miller-Rabin probabilistic primality test makes use of two results in number theory. First, Fermat's Little Theorem, which states

$$a^{p-1} \equiv 1 \pmod{p}$$

if p is prime. The second result is that the solutions to the following equation:

$$x^2 \equiv 1 \pmod{p}$$

when p is prime are $x \equiv 1, -1 \pmod{p}$.

Now, if we take some odd integer n to be tested for primality and we rewrite it as $2^s \cdot d + 1$, where s and d are both positive integers and d is odd we can test the primality of n by checking to see if at least one of the following modular relations (derived from the above-mentioned results) holds true:

$$a^d \equiv 1 \pmod{n}$$

And

$$a^{2^r \cdot d} \equiv -1 \pmod{n}$$

Note that a must be an integer such that $0 < a < n$.

A Python implementation of the Miller-Rabin test is in Appendix C. An important takeaway from the implementation is that the test requires a random number between 2 and $n - 1$, where n is the number being tested for primality, and a is the number to be used as a modular base.

The Miller-Rabin test is unique in that it is really a compositeness test rather than a primality test. This is because the Miller-Rabin test only proves if a number is composite. Numbers that pass the Miller-Rabin test are probably prime, but there is a chance that they are composite. Note that this means that the only type of error in this test is when a composite number is labelled as prime.

When the Miller-Rabin test is used in practice to test if a number is prime, it is generally run more than once. This is because running the test several times with different modular bases each time (with the bases chosen randomly from the previously mentioned range, $[2, n - 1]$) reduces the chance that the test outputs a false positive, meaning it claims that the inputted number is prime when it is actually composite.

The algorithm runs efficiently in polynomial time, with typical implementations running in $O(k \log^3 n)$ (where k is the number of times the test is run). However, some implementations that make use of FFT-based multiplication can reduce the runtime even more to $O(k \log^2 n \log \log n)$.

Due to the Miller-Rabin test's ease of implementation and its speed, it is used very often (e.g. implementations of the RSA encryption algorithm). Here's its runtime compared to the previously mentioned deterministic algorithms:

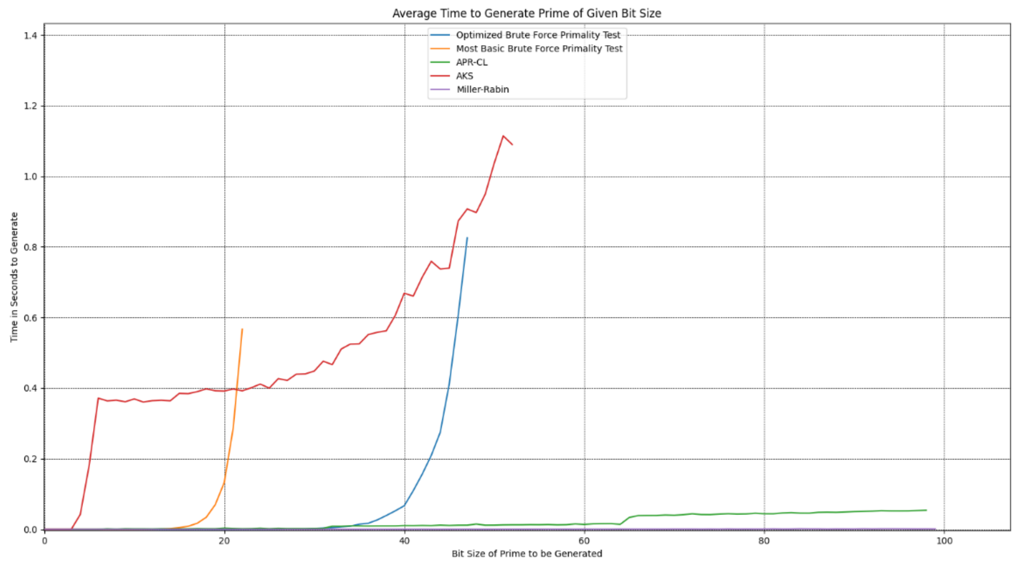

Figure 4. The runtime of the Miller-Rabin primality test compared to that of deterministic primality tests graphically.

A view of these runtimes on a logarithmic scale better illustrates how it compares to deterministic algorithms:

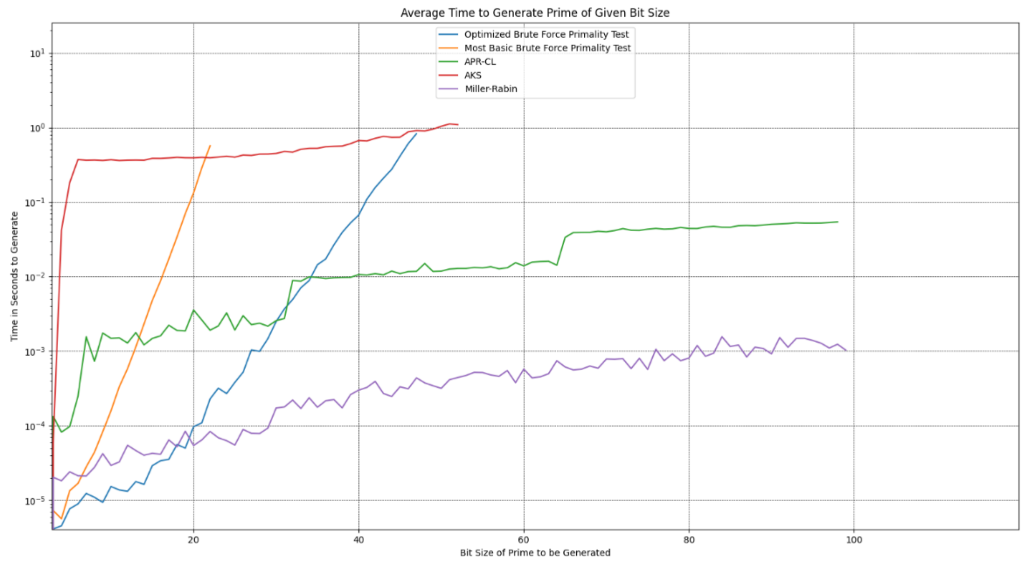

Figure 5. A comparison of the Miller-Rabin algorithm to the deterministic algorithms using logarithmic scaling on the vertical axis.

The Miller-Rabin algorithm is especially prominent in encryption software that requires prime number generation, such as code implementing the RSA algorithm.

Pseudoprimes

Generally, the probability of error for the Miller-Rabin Test is 4^{-k}, where k is again the number of times the test is run with a different modular base. However, there is more to this error probability than meets the eye, and it will be discussed in slightly greater depth in Appendix D. There has been work done on how to select these modular bases to reduce error even further. But, even with all of these measures there are composite numbers that manage to slip through the cracks and pass the test. These numbers are known as pseudoprimes--composite numbers that have some property that prime numbers have.

One of the most well-known sequences of pseudoprimes is known as the Carmichael numbers, which satisfy

$$Q^{q-1} \equiv 1 \pmod q$$

for all Q coprime to q. In other words, the Carmichael numbers are a sequence of composite numbers that manage to fully satisfy Fermat's Little Theorem. Due to this property, the Carmichael numbers have the potential to pass primality tests that make use of Fermat's Little Theorem (such as the Miller-Rabin test). And, in 1994, W.R. Alford, Andrew Granville, and Carl Pomerance proved that there are an infinite number of Carmichael numbers.

Baillie-PSW

The Baillie-PSW test is another widely used probabilistic test. It is really a combination of two other tests: one round of a Miller-Rabin test with 2 as the modular base, and a Lucas probable prime test with specifically calculated parameters. Lucas probable prime tests will not be discussed in depth in this paper. A rundown of Lucas's ideas regarding primality can be found Carl Pomerance's paper "Primality Testing: Variations on a Theme of Lucas."

The strength of this test comes from the fact that, as previously stated, it combines two existing probabilistic primality tests. This is important as different probabilistic primality tests may have very minimal amounts of overlap in the pseudoprimes that are capable of passing a given test. This can allow for more accurate results.

For example, take two hypothetical probabilistic tests P and Q. Let the set of pseudoprimes capable of passing test P be denoted by P_s and the set of pseudoprimes capable of passing test Q by Q_s. Then, if the intersection of these sets is the empty set, meaning $P_s \cap Q_s = \emptyset$, a new test implementing both test P and test Q would have no pseudoprimes. Therefore, this new test would be deterministic.

The Baillie-PSW test does a great job of showing off the power of combining two probabilistic tests. It has no pseudoprimes less than 2^{64}, meaning that it is deterministic on the interval $1 < n < 2^{64}$, where n is the odd integer to be tested for primality. Furthermore, there are no known pseudoprimes for the Baillie-PSW test.

Searching For New Probabilistic Tests

Even though Miller-Rabin and Baillie-PSW are incredibly powerful primality tests, the search for stronger and faster probabilistic tests continues. One lesser known (but promising) probabilistic primality test is the Frobenius primality test. This test has been proven to be deterministic for numbers less than 2^{64}, and there are currently no known pseudoprimes for the test at all. This means that the test probably has a very low chance of returning a false result, making it quite powerful. Its time complexity is roughly double that of a Miller-Rabin test.

This test is unique from many other probabilistic primality tests as it doesn't rely on standard number theory results like Fermat's little theorem. The Frobenius test exemplifies a relatively recent trend that is shifting into new fields for primality results, including (but not limited to) field theory, group theory, and graph theory.

Quantum Computing-A New Primality Test Breeding Ground

In recent years, quantum computing--which makes use of subatomic qubits rather than standard bits--has grown dramatically. Quantum computing's strength stems from a few key concepts, including uncertainty and entanglement.

The computing strength of quantum computers in comparison to our existing supercomputers is enormous. Tasks that a classical computer might take thousands of years to do a quantum computer could do in under ten minutes. Due to this extreme improvement in computational ability, lots of time and effort is being poured into research surrounding computer algorithms--especially cryptography protocols. The first such protocol is BB84, a simple quantum algorithm for securely distributing a key (which is then used to encrypt a message).

Shor's Algorithm

One of the most revolutionary quantum computing algorithms is Shor's algorithm. The algorithm, through a combination of results in number theory and concepts such as the quantum Fourier transform, is one that can theoretically factor a number in polynomial time.[2] Of course, this is devastating to most public key encryption methods today as practically all of them function under the assumption that factoring a number cannot be done in polynomial time.

One of the key reasons for which Shor's algorithm is as fast as it is is due to its usage of the quantum Fourier transform, which can be performed extremely quickly on a quantum computer. The standard classical Fourier transform works by taking a vector[3] in one "domain" and mapping it to another. For example, a common usage for the Fourier transform is taking a function in the spatial domain and converting it to the frequency domain.

The algorithm mathematically acts on a quantum state

$$|x\rangle = \sum_{q=0}^{N-1} x_q |q\rangle$$

And expresses it in another domain, which can be denoted as

$$|y\rangle = \sum_{q=0}^{N-1} y_q |q\rangle$$

According to the following formula:

$$y_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n \omega_N^{nk}$$

where $k = 0, 1, 2, \ldots, N-1$.

At a fundamental level, the power of the quantum Fourier transform, which Shor's algorithm harnesses extremely well, is derived from its ability to represent a vector in a way that spotlights certain qualities of the vector

[2] Readers might infer that Shor's algorithm, then, can be used as a primality test by itself. However, this is not the case. The algorithm will not function properly for inputs that are not composite numbers. Furthermore, the input must be odd, and it cannot be a perfect power.

[3] Note that functions can be expressed as infinitely long vectors by enumerating the functions values over a specific domain.

without altering the function at all (meaning that the two representations are equivalent). Researchers and mathematicians are attempting to see how this strength can be used for creating better primality tests, which in turn would benefit cryptographic algorithms.

Advancements in Quantum Cryptography

Even though Shor's algorithm generated fear due to its ability to break many public key cryptographic systems, it also created much excitement in the realm of primality generation.

Quick advancements in primality tests were made very soon after Shor published his algorithm. One such advancement had to do with the Pocklington–Lehmer primality test. This test, while being able to produce a primality certificate, requires a partial factorization of $N-1$, where N is the number being tested for primality. Prior to Shor's algorithm, the partial factorization of $N-1$ was a sort of bottleneck for the algorithm, preventing it from being applicable to fields like cryptography.

With the creation of Shor's algorithm, however, $N-1$ can be quickly factored. A quantum probabilistic variant of the Pocklington-Lehmer primality test (making use of Shor's algorithm, once again) has also been created, running in $O((\log N)^3 \log\log N \log\log\log N)$ time.

Another such algorithm that has been created more recently makes use of quantum order finding, and it was published in 2017. This algorithm needs to compute $O((\log n)^2 n^3)$ operations to test the primality of a number with n bits. This can be further reduced using fast multiplication.

Next Steps and Future Implications

The field of primality tests is one that is only going to grow. Technological advancements will keep the field thriving, and as many have already witnessed, the field will likely shift its focus from classical computers to quantum computers. Once the necessary hardware to create function quantum computers is developed, most existing cryptography algorithms will be in danger. This will affect the average person when it comes to sending any sort of private information over the internet (e.g. electronic commerce), and it will also threaten the security of information transmission at the governmental level. To address this, mathematicians and scientists will be forced to strengthen and create new encryption algorithms, and this will begin with improved primality tests.

Next steps in this field will likely focus on finding more efficient quantum primality tests along with searching for results about primality in fields besides number theory and group theory. One such field may be graph theory. With many possible tracks for new discoveries along with a looming necessity for stronger tests, work in primality tests is going to expand rapidly.

Acknowledgements

I would like to thank Dr. Ethan Hutt for his guidance during the entirety of my writing of this paper. His knowledge of how to keep track of resources, effectively extract information from papers, and general are what allowed me to produce this body of work.

References

Agrawal, M., Kayal, N., & Saxena, N. (2004). Primes is in p. *Annals of Mathematics*, *160*(2), 781–793. https://doi.org/10.4007/annals.2004.160.781

Atkin, A. O., & Morain, F. (1993). Elliptic curves and primality proving. *Mathematics of Computation*, *61*(203), 29–68. https://doi.org/10.1090/s0025-5718-1993-1199989-x

Baillie, R., & Wagstaff, S. S. (1980). Lucas pseudoprimes. *Mathematics of Computation*, *35*(152), 1391–1417. https://doi.org/10.1090/s0025-5718-1980-0583518-6

Bernhardt, C. (2020). *Quantum computing for everyone*. The MIT Press.

Chau, H. F., & Lo, H.-K. (1997). Primality test via quantum factorization. *International Journal of Modern Physics C*, *08*(02), 131–138. https://doi.org/10.1142/s0129183197000138

Damgard, I., Landrock, P., & Pomerance, C. (1993). Average case error estimates for the strong probable prime test. *Mathematics of Computation*, *61*(203), 177. https://doi.org/10.2307/2152945

Damgård, I. B., & Frandsen, G. S. (2003). An extended quadratic frobenius primality test with average and worst case error estimates. *Fundamentals of Computation Theory*, 118–131. https://doi.org/10.1007/978-3-540-45077-1_12

Grantham, J. (1998). A probable prime test with high confidence. *Journal of Number Theory*, *72*(1), 32–47. https://doi.org/10.1006/jnth.1998.2247

Ishmukhametov, S. T., Rubtsova, R., & Savelyev, N. (2018). The error probability of the Miller–Rabin Primality test. *Lobachevskii Journal of Mathematics*, *39*(7), 1010–1015. https://doi.org/10.1134/s1995080218070132

Lenstra, Jr., H., & Pomerance, C. (2019). Primality testing with gaussian periods. *Journal of the European Mathematical Society*, *21*(4), 1229–1269. https://doi.org/10.4171/jems/861

Rabin, M. O. (1980). Probabilistic algorithm for testing primality. *Journal of Number Theory*, *12*(1), 128–138. https://doi.org/10.1016/0022-314x(80)90084-0

Rajput, J., & Bajpai, A. (2019). Study on deterministic and probabilistic computation of primality test. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3358737

Schoof, R. (2008). Four primality testing algorithms. *ArXiv*. https://doi.org/10.48550/ARXIV.0801.3840

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings 35th Annual Symposium on Foundations of Computer Science*. https://doi.org/10.1109/sfcs.1994.365700