Primality Tests and Factoring Algorithms

Alexa Wingate, Sarah Fujimori

December 9, 2018

1 Introduction

Much of modern cryptography rests on our ability to factor a number, especially numbers that are the products of two large primes. To do this, we employ primality tests and factoring algorithms that use number theory to accomplish their goal. In this paper, we discuss the most-well known primality tests and factoring algorithms.

2 Primality Tests

There are two types of primality tests: deterministic and probabilistic. Deterministic tests are always correct in distinguishing primes and composites, while probabilistic tests can be wrong sometimes (though rarely).

Definition 2.1. *A pseudoprime is a composite number which passes a primality test.*

2.1 The Fermat Primality Test

The most basic primality test is simply testing if a number has any factors. However, this method is extremely inefficient and we will not be discussing it further. Instead, another simple but faster probabilistic test is the Fermat primality test, which tests if $s^{n-1} \equiv 1 \pmod{n}$ (Fermat's Little Theorem) for some value of a. If the theorem holds true, n is probably prime, but if it's not, n is composite. Therefore, the Fermat primality test is more of a compositeness test than a primality test.

Unfortunately, there are Fermat pseudoprimes that satisfy Fermat's Little Theorem for all a. These are called Carmichael numbers. One can find Carmichael numbers using Korselt's Criterion:

Theorem 2.1 (Korselt's Criterion)**.** *Let $n \geq 2$ be a composite number. Then n is a Carmichael number if and only if n is squarefree and for all primes p dividing n, $p-1|n-1$.*

Proof:

Assuming n is a Carmichael number, we will show that it must be squarefree. If n is not squarefree, it can be written as $p^k n'$, where the power of k is greater than 1 and $\gcd(p, n') = 1$. Our goal is to show that p can only be 1. The Chinese Remainder Theorem tells us that there exists an a such that $a \equiv 1+p \pmod{p^k}$ and $a \equiv 1 \pmod{n'}$. Then $\gcd(a, n) = 1$, so we can use Fermat's Little Theorem (or the definition of Carmichael numbers): $a^{n-1} \equiv 1 \pmod{n}$. From the first congruence above, we can reduce this relationship to $(1+p)^{n-1} \equiv 1 \pmod{p^2}$, and then applying the Binomial Theorem, we get $(1+p)^{n-1} \equiv 1 + p(n-1) \pmod{p^2}$. Finally, since n is divisible by p, $1 + p(n-1) \equiv 1 - p \pmod{p^2}$. Therefore $1 - p \equiv 1 \pmod{p}$, which is a contradiction, and thus $k = 1$ n is squarefree.

Now we have to show that $p-1|n-1$ for all primes p dividing n. n is squarefree, so p and $\frac{n}{p}$ are coprime. Then, choose b such that b has order $p-1$ in $\mathbb{Z}/p\mathbb{Z}$, meaning that $p-1$ is the smallest number for which $b^{p-1} \equiv 1 \pmod{p}$. The Chinese Remainder tells us that there is an $a \in \mathbb{Z}$ for which $a \equiv b \pmod{p}$ and $a \equiv 1 \pmod{n/p}$. Therefore $a^{n-1} \equiv 1 \pmod{n}$. Reducing both sides mod p, we get $b^{n-1} \equiv 1 \pmod{p}$. It follows that $p-1|n-1$.

Fortunately, proving this for the other direction is much shorter. Let n be squarefree such that $p-1|n-1$ for every prime p dividing n. If we choose an a where $\gcd(a, n) = 1$ for all the ps, then $\gcd(a, p)$ is also 1.

This allows us to say that $a^{p-1} \equiv a \pmod{p}$. But also since $p-1$ is a factor of $n-1$, $a^{n-1} \equiv a \pmod{p}$. This is true for *all* primes dividing n, and n is squarefree, so $a^{n-1} \equiv a \pmod{n}$. Therefore n is a Carmichael number.

Corollary 2.1.1. *Let $n = (6k+1)(12k+1)(18k+1)$, where $6k+1$, $12k+1$, and $18k+1$ are prime. Then n is a Carmichael number.*

This is one method of easily calculating Carmichael numbers without going through the trouble of finding individual numbers that fit the criteria above.

Proof:

We can just verify that Korselt's Criterion hold for numbers of this form. Of course, n is always squarefree. Expanding n and subtracting 1 gives us $n - 1 = 1296k^3 + 396k^2 + 36k$, which is clearly divisible by $6k$, $12k$, and $18k$.

2.2 The Lucas-Lehmer Test

Unlike the Miller-Rabin and Fermat tests that we have discussed above, the Lucas Lehmer is a deterministic test for Mersenne primes.

Definition 2.2. *A Mersenne number is defined as a number that can be written as $M_p = 2^p - 1$ for some prime p, and the sequence S_n is defined by $S_n = S_{n-1}^2 - 2$ and $S_1 = 4$.*

Lemma 2.2. *$S_n = \omega^{2^{n-1}} + \overline{\omega}^{2^{n-1}}$ for $\omega = 2 + \sqrt{3}$ and $\overline{\omega} = 2 - \sqrt{3}$.*

It follows from this lemma that if $S_{p-1} \equiv 0 \pmod{M_p}$, then $\omega^{2^{p-2}} + \overline{\omega}^{2^{p-2}} = RM_p$ for some integer R. Multiplying this equation by ω^{2p-2} and squaring both sides, we get $\omega^{2^p} = (RM_p\omega^{2p-2} - 1)^2$. This will become important later.

Lemma 2.3. *X is a set with an associative binary operation and has an identity. Then X^* is the set of invertible elements in X, which forms a group.*

Lemma 2.4. *If G is a finite group then the order of an element in G is at most the order of the group. If $x \in G$ and $x^r = 1$ then the order of x divides r.*

Theorem 2.5. *The Mersenne number M_p is prime if it divides S_{p-1}.*

Proof:

Assume that M_p is composite. Let q be a prime factor of M_p with $q^2 \leq M_p$. Then let Z_q be the set of residue classes mod q, and X be the set $\{a + b\sqrt{3} : a, b \in Z_q\}$. Two binary operations on X are addition, for which X is an abelian group, and multiplication, which has identity 1 in X. Finally, let $X*$ be the group of invertable elements in X (for multiplication). By Lemma 1.4, $X*$ is a group. By Lemma 3, the order of any element in $X*$ is at most $q^2 - 1$. We also know that $\omega = 2 + \sqrt{3}$ is an element in X. Since q divides M_p, $RM_p\omega^{2^{p-1}}$ is 0 in X. Therefore, we can simplify the equations from before into $\omega^{2^{p-1}} = -1 \implies \omega^{2^p} = 1$. This means that ω is in $X*$ with order 2^p (which follows Lemma 1.5). By Lemma 1.5 again we know that $2^p \leq q^2 - 1$. But $q^2 - 1 \leq M_p - 1 = 2^p - 2$ and we have a contradiction.

2.3 The Miller-Rabin pseudoprime test

The Miller-Rabin test is similar to the Fermat primality test, both being based off Fermat's Little Theorem, except the Miller-Rabin test is much more accurate and efficient.

Theorem 2.6. *Let $n = 2^e k + 1$ for odd n. Then if $a^k \equiv 1 \pmod{n}$ or $a^{2^i k} \equiv -1 \pmod{n}$ for $1 \leq a \leq n-1$ and $i \in \{0, ..., e-1\}$, n is prime.*

Proof:

We can prove this easily by factoring $x^{n-1} - 1$. Let $n - 1 = 2^e k$ for and odd n. The polynomial $x^{n-1} - 1 = x^{2^e k} - 1$ can be factored as long as the coefficient of k is a power of 2:

$$\begin{aligned}
x^{2^e k} - 1 &= (x^{2^{e-1}k})^2 - 1 \\
&= (x^{2^{e-1}k} - 1)(x^{2^{e-1}k} + 1) \\
&= (x^{2^{e-2}k} - 1)(x^{2^{e-2}k} + 1)(x^{2^{e-1}k} + 1) \\
&= (x^{2^{e-3}k} - 1)(x^{2^{e-3}k} + 1)(x^{2^{e-2}k} + 1)(x^{2^{e-1}k} + 1) \\
&... \\
&= (x^k - 1)(x^k + 1)(x^{2k} + 1)(x^{4k} + 1)...(x^{2^{e-1}k} + 1)
\end{aligned}$$

If n is prime and $1 \leq a \leq n-1$, then $a^{n-1} - 1 \equiv 0 \pmod{n}$ by Fermat's Little Theorem. Using the factorization above we have:

$$(a^k - 1)(a^k + 1)(a^{2k} + 1)(a^{4k} + 1)...(a^{2^{e-1}} + 1) \equiv 0 \pmod{n}$$

One of the factors has to be $0 \pmod{n}$, so $a^k \equiv 1 \pmod{n}$ or $a^{2^i k} \equiv -1 \pmod{n}$ for $i \in \{0, ..., e-1\}$. Like the Fermat test, the this one is an extremely accurate compositeness test instead of directly finding primes.

2.4 The AKS Primality Test

The AKS primality test was created in 2002 and settled the historic question of whether or not primality could be solved in polynomial time, placing PRIMES in P. The complete time analysis of the test are somewhat long and complex to discuss in this paper, so we will just present the algorithm by itself.

Theorem 2.7. *Find an a such that $\gcd a, n = 1$. Then n is prime if and only if $(x+a)^n \equiv x^n + a \pmod{n}$.*

Proof:

We define polynomials to be congruent $\pmod{n}$ if all the coefficients are equivalent $\pmod{n}$. For $0 < k < n$, the coefficient of x^i in $(x+a)^n - (x^n + a)$ is $\binom{n}{k}a^{n-k}$. First, suppose n is prime. The numerator of $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ divides n but the denominator does not, so all the coefficients are $0 \mod n$. Now suppose n is composite. Then let q be a prime factor of n such that $q^i | n$. We also know that q^k does not divide $\binom{n}{k}$ and is coprime to a^{n-q}. Therefore the coefficient of x^q is not equivalent to $0 \pmod{n}$, and the theorem does not hold true.

Let n be prime so we can construct a primality test around Theorem 1.7. Since all the coefficients of $(x+a)^n - (x^n + a)$ are divisible by n, this must still be true when we group some of the terms together. We can define $f_a(x)$:

$$\begin{aligned}
f_a(x) &= (x+a)^n - (x^n + a) \\
f_a(x) &= \sum_{k=0}^{n-1} b_k x^k
\end{aligned}$$

We can group together every rth term (for a certain well chosen r) and see if the sum of the coefficients in that grouping is $0 \pmod{n}$. Then we can define $h_a(c)$ for every $0 \leq c \leq r$:

$$h_a(c) = \sum_{\substack{0 \leq k \leq n-1 \\ k \equiv c \pmod{r}}} b_k$$

If, for some c and r, $h_a(c) \not\equiv 0 \pmod{n}$, n is composite.

Now that this looks slightly more like a proper primality test (though a painfully slow one), here is the actual AKS algorithm:

1. Check if $n = a^b$ for some a, b
2. Find the smallest $r \pmod{n} > \log(n)^2$
3. If $1 < \gcd a, n < n$ for some $1 \leq a \leq r$, then n is composite.
4. If $r \geq n$, n is prime
5. Then, for every a $1 \leq a \leq \lfloor \log(n)\sqrt{\phi(r)} \rfloor$ and c $0 \leq c \leq r$, if $h_a(c) \not\equiv 0$, then n is composite
6. n is prime

3 Factoring Algorithms

Definition 3.1. *Big O notation is used to convey how fast an algorithm is for arbitrarily large numbers. A function $f(x)$ is $O(g(x))$ if for large values of x, $f(x)$ is approximately $g(x)$.*

There are several categories of common functions when discussing speed: constant (which is $O(1)$), logarithmic (which is $O(\log n)$), polynomial (which is $O(n^a)$ for some integer a), exponential (which is $O(a^n)$ for some integer a).

While there are currently no polynomial-time algorithms to factor number, i.e. we can't factor a b-bit number in speed $O(b^a)$ for some constant a, it has not been proven that there do not exist such algorithms. Currently, the hardest number to factors are products of two large primes.

In this paper, we will discuss several algorithms with different speeds.

3.1 Slow factoring algorithms

The most obvious way to try to factor a number n is to simply try all the primes up to $\sqrt{n}$; this method is called trial division and will thus have time $O(\sqrt{n})$. Although this is kind of efficient if we know n is the product of a bunch of small primes, it can take a long time if n is prime or a product of large primes.

3.2 Pollard's ρ algorithm

We now introduce a faster algorithm, Pollard's ρ algorithm:

Let $f(x) = x^2 + a$ be a function for some integer a not equal to 0 or -2. Let x_0 be a random integer between 0 and $n-1$, inclusive, and for all positive integers i, let x_i be the least positive residue of $f(x_{i-1})$. Then, the sequence x_n eventually repeats. This is because of the pigeon hole principle: there are finitely many values that $f(x) \pmod{n}$ can be, so since each term of the sequence only depends on the last, the sequence is eventually periodic.

Let d be a non-trivial divisor of n. Since there are a smaller number of residue classes mod d than residue classes mod n, the sequence x_i is more likely to repeat sooner mod d than mod n. Pollard's ρ algorithm rests on this idea, and it is as follows:

1. Pick a and x_0 (we usually use $a = 1$ and $x_0 = 2$).
2. Set $x = y = x_0$.
3. During each iteration of the algorithm, set $x = f(x)$ and $y = f(f(x))$.
4. Calculate $d = \gcd(|x - y|, n)$.
5. If $1 < d < n$, then we have found a non-trivial divisor of n. If $d = 1$, then we try again, and if $d = n$, then the algorithm is a failure and we pick a different value of a or x_0 (This happens in the rare case when the sequence x_i happens to repeat mod d and mod n at the same time).

An example is shown below, with $a = 1$, $x_0 = 2$, and $n = 323 = 17 * 19$:

i. $x = 5$, $y = 26$; $d = \gcd(|5 - 26|, 323) = 1$

ii. $x = 26$, $y = 316$; $d = \gcd(|26 - 316|, 323) = 1$

iii. $x = 31$, $y = 240$; $d = \gcd(|31 - 240|, 323) = 19$

which shows that this algorithm is pretty fast, especially if the prime factors of n are relatively small. More specifically:

Theorem 3.1. *Let p be a prime divisor of n. The Pollard ρ algorithm takes time $O(\sqrt{p})$ to find the divisor p.*

Note that the numbers we generated in this case are not actually truly random; they are pseudo-random. We will prove Theorem 2.3 in the case that these numbers are random. We do this with the following lemma, which is a generalization of the birthday paradox:

Lemma 3.2. *Say there are k random numbers, out of p possibilities. Then, the probability that no two of the numbers match is about $e^{\frac{-k^2}{2p}}$.*

Proof. We first choose the first number, which can be anything. When choosing the second number, there are $p - 1$ valid choices out of p, since we do not want the second number and the first number to be the same. Similarly, for the third number, the probability that it doesn't match the second or first is $\frac{p-2}{p}$. Thus, the probability that no two of the k numbers match is:

$$\begin{aligned}&(1)(\tfrac{p-1}{p})(\tfrac{p-2}{p})\cdots(\tfrac{p-(k-1)}{p})\\ &= (1)(1-\tfrac{1}{p})(1-\tfrac{2}{p})\cdots(1-\tfrac{k-1}{p})\end{aligned}$$

Recall that:

$$e^x = 1 + x + \tfrac{x^2}{2!} + \tfrac{x^3}{3!} + \cdots$$

Although $1 + x$ is a very bad estimate of e^x for large values of x, we can see that it is an okay estimate for small values of x, since the other terms will approach 0 much faster. Thus, $1 - \frac{1}{p} \approx e^{\frac{-1}{p}}$, $1 - \frac{2}{p} \approx e^{\frac{-2}{p}}$, and so on:

$$\begin{aligned}(1)(1-\tfrac{1}{p})(1-\tfrac{2}{p})\cdots(1-\tfrac{k-1}{p}) &\approx (e^{\frac{-1}{p}})(e^{\frac{-2}{p}})\cdots(e^{\frac{-(k-1)}{p}})\\ &= e^{\frac{-1}{p}(1+2+\cdots+k-1)}\\ &= e^{\frac{-k(k-1)}{2p}}\\ &\approx e^{\frac{-k^2}{2p}}\end{aligned}$$

as desired.

□

We want the probability that at least two of the numbers match to be at least $\frac{1}{2}$, so we can solve for k to get an approximation of the speed of the algorithm:

$$\begin{aligned}1 - e^{\frac{-k^2}{2p}} &= \tfrac{1}{2}\\ e^{\frac{-k^2}{2p}} &= \tfrac{1}{2}\end{aligned}$$

Taking the natural logarithm of both sides:

$$\begin{aligned}\tfrac{-k^2}{2p} &= -\ln 2\\ \tfrac{k^2}{2p} &= \ln 2\end{aligned}$$

We then multiply by $2p$ and take the square root of both sides:

$$k^2 = 2p \ln 2 \; k = \sqrt{2pln2}$$

so we conclude that the speed of the algorithm is $O(\sqrt{p})$.

3.3 Elliptic curves

Using the theory of elliptic curves, we introduce Lenstra's elliptic curve factorization algorithm, which is one of the fastest known factoring algorithms, just after the quadratic sieve algorithm (which we will discuss next).

The algorithm is an improvement of a more basic algorithm called Pollard's $p-1$ factorization, which depends on Fermat's Little Theorem. Let n be the number we are trying to factor; if n is composite, we know that n will have a prime factor p satisfying $p < \sqrt{n}$. For this prime p, $a^{(p-1)} \equiv 1(\mod p)$ by Fermat's Little Theorem for some a relatively prime to n, so $a^{(p-1)m} \equiv 1(\mod p)$. Then, if we can find a k such that $k = (p-1)m$ for some value of m, then we know that $\gcd(n, a^k - 1)$ will produce a non-trivial factor of n.

The algorithm is:

1. Let n be the number we are factoring, and let $K < \sqrt{n}$, $k = (1, 2, \cdots K)$.
2. Choose an a between 1 and n exclusive that is relatively prime to n (since we can't use Fermat's Little Theorem if this is not true).
3. Calculate $x = \gcd(a^k - 1, n)$; if x is between 1 and n exclusive, then x is a non-trivial factor of n; if not, then either pick a different value of K or a.

This algorithm relies on $p-1$ being a product of many small primes with small factors; if $p-1$ is a product of large primes, then the algorithm is not very good. Thus, instead of considering the residue classes modulo p, we consider the group of points on an elliptic curve over $\mathbb{F}_p$. This algorithm works because of the following difficult theorem about elliptic curves, which is called the Hasse-Weil bound:

Theorem 3.3. *Let E be an elliptic curve over $\mathbb{F}_p$, and let N be the number of points in $E(\mathbb{F}_p)$. Then, $|N - (p+1)| \leq 2\sqrt{p}$.*

The steps for Lenstra's Elliptic Curve Factorization Algorithm are listed below:

1. Choose an elliptic curve S and a starting point P_1 on E.
2. Choose a smoothness bound B, and let $F = \{p_1, p_2 \cdots p_k\}$ be the factor base consisting of all primes less than B. For each $p \in F$, let $e_p = \lfloor \log_q n \rfloor$. Let $m = \prod_{i=1}^{k} p_i^{e_{p_i}}$.
3. For each i, let $P_{i+1} = p_i^{e_{p_i}} P_i$. If this multiplication doesn't work out, then we have successfully factored n. During the process, we computed d, the denominator of the slope between points P_i and P; if the multiplication fails, then $\gcd(d, n)$ is a non-trivial factor of n.

3.4 The Quadratic Sieve Algorithm

The quadratic sieve algorithm is one of the fastest known factoring algorithms; there is a faster variant of it known as the number field sieve algorithm, but we will not discuss it here.

Definition 3.2. *Let F be a nonempty set of positive primes, and let k be a positive integer. We call F a factor base, and we say that k is smooth over F if all the primes dividing k are in F.*

The sieve algorithms rest on the following idea:

Lemma 3.4. *Let x, y be integers such that $x+y$ and $x-y$ are not divisible by n, satisfying $n|(x+y)(x-y)$. Then, $\gcd(x+y, n)$ and $\gcd(x-y, n)$ are factors of n other than 1 and n.*

Let B be a positive integer, and $F = p_1, p_2, p_3, \cdots p_k$ be a factor base consisting of all primes less than B (the smoothness bound).

The quadratic sieve algorithm is as follows:

1. Pick a bound B, and let F be the set of primes less than B. Let the number of elements in F be f.
2. Find $f+1$ numbers r such that $r^2(\mod n)$ is smooth over F.

3. Factor each of these numbers (this shouldn't be too hard, since they are smooth over F), and put the exponents mod 2 into a matrix.

4. We use a process called row reduction to find a subset of the numbers that have a product that is a perfect square.

Our goal in row reduction is to modify the matrix of exponents M such that we can easily identify a subset of rows whose entries are all even (and thus, whose product of corresponding values of r is a square mod n).

We first switch the rows and columns of M. Currently, the rows of M represent the values of r, and the columns of M represent the primes, so the rows of M^T represent the primes and the columns of M^T represent the r values.

We want to modify M so that it satisfies the following conditions:

1. The first 1 in row i has to come to the left of the first 1 in row $i+1$.

2. Whatever column the first 1 in row i comes in must have all other entries be 0.

We allow the following operations on M^T to be done:

1. Switch any two rows.

2. Replace a row i with the sum of row i and some other row, j.

Operation 1 will clearly preserve the property that the sum of columns has all even entries, since it just rearranges the order of the entries in the columns. Operation 2 will also preserve this property: say we have a row i with entries $i_1, i_2, \cdots i_k$ and a row j with entries $j_1, j_2, \cdots j_m$. If a subset of the columns have a sum with only even entries, then a subsequence of $i_1, i_2, \cdots i_k$, say $a_1, a_2, \cdots a_I$, and a subsequence of $j_1, j_2, \cdots j_m$, say $b_1, b_2, \cdots c_I$, both have sum 0. If we replace row i with row $i+j$, then the nth entry of row i will be $i_n + j_n$. Thus, the sum of terms of the new sequence of a's will be $(a_1+b_1)+(a_2+b_2)+\cdots+(a_I+b_I) = (a_1+a_2+\cdots+a_I)+(b_1+b_2+\cdots+b_I) = 0$, while the sum of the b's does not change.

After we have the matrix in the proper form, we pick a subset of the columns with sum having only even entries. This is pretty simple when the matrix is in this form, as we will see in the example. We take the numbers that the columns correspond to, say $r_1, r_2, \cdots r_s$, and we form two numbers x and y:

$$x = r_1 r_2 \cdots r_s$$
$$y^2 = (r_1^2(\mod n)(r_2^2(\mod n)\cdots(r_s^2(\mod n)$$

We can see that $x^2 - y^2$ will be a multiple of n, so $\gcd(x-y, n)$ will be a non-trivial factor of n. Note that in a rare case, we might get 1, but then we can just try again.

To clarify what the algorithm does, we show an example of the algorithm. Let $n = 221$, and $B = 10$ (so $F = 2, 3, 5, 7$). To search for numbers r such that $r^2(\mod n)$ is smooth over F, we start at $\lfloor\sqrt{n}\rfloor = 11$. To do this, we can simply write a program in Sage; we find that the first 5 values of r are 23, 24, 27, 29, and 30. We also factor $r^2(\mod n)$ for each r:

$23^2(\mod n) = 2^2 \cdot 5^2$
$24^2(\mod n) = 2^2$
$27^2(\mod n) = 2 \cdot 7$
$29^2(\mod n) = 2 \cdot 3^2 \cdot 7$
$30^2(\mod n) = 2 \cdot 3 \cdot 7$

We then put the exponents mod 2 into a matrix M and transpose M:

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

$$M^T = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

After we perform row reduction on M, we get the following:

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Since columns 1 is all 0's, we take the first number, 23. We get $x = 23$, $y = 10$, which gives us the non-trivial factor 13.

4 Sources

1. Bruce, J. W. "A Really Trivial Proof of the Lucas-Lehmer Test." The American Mathematical Monthly, vol. 100, no. 4, Apr. 1993, pp. 370-71. JSTOR, DOI:10.2307/2324959. Accessed 3 Nov. 2018.
2. Conrad, Keith. The Miller-Rabin Test. University of Connecticut, www.math.uconn.edu/ kconrad/blurbs/ugradnumthy/millerrabin.pdf. Accessed 4 Nov. 2018.
3. Agrawal, Manindra, et al. "PRIMES is in P." Annals of Mathematics, vol. 160, no. 2, 2004, pp. 781-93.
4. http://www.cs.columbia.edu/ rjaiswal/factoring-survey.pdf
5. https://webcourse.cs.technion.ac.il/236500/Spring2016/ho/WCFiles/cryptanalysis-slides-02-factoring.1x1.pdf
6. http://web.mit.edu/16.070/www/lecture/big$_o$*.pdf*
7. http://math.uchicago.edu/ may/REU2014/REUPapers/Parker.pdf