Factorization and Primality Testing Chapter 7 Ad Hoc Methods

Robert C. Vaughan

October 23, 2023

Pollard rho

- John Pollard, in the 1970s, created a number of different techniques for factoring large integers.

Pollard rho

- John Pollard, in the 1970s, created a number of different techniques for factoring large integers.
- The Pollard rho is named for a way of representing the iterative process which looks like the Greek lower case rho, ρ.

Pollard rho

- John Pollard, in the 1970s, created a number of different techniques for factoring large integers.
- The Pollard rho is named for a way of representing the iterative process which looks like the Greek lower case rho, ρ.
- Suppose you start from some object P_0, and successively compute $P_1, P_2, P_3, \ldots$ and that sooner or later you find some pair $j < k$ so that $P_j = P_k$.

Pollard rho

- John Pollard, in the 1970s, created a number of different techniques for factoring large integers.
- The Pollard rho is named for a way of representing the iterative process which looks like the Greek lower case rho, ρ.
- Suppose you start from some object P_0, and successively compute $P_1, P_2, P_3, \ldots$ and that sooner or later you find some pair $j < k$ so that $P_j = P_k$.
- Then $P_{j+1} = P_{k+1}$ and so on.

Pollard rho

- John Pollard, in the 1970s, created a number of different techniques for factoring large integers.
- The Pollard rho is named for a way of representing the iterative process which looks like the Greek lower case rho, ρ.
- Suppose you start from some object P_0, and successively compute $P_1, P_2, P_3, \ldots$ and that sooner or later you find some pair $j < k$ so that $P_j = P_k$.
- Then $P_{j+1} = P_{k+1}$ and so on.
- That is the sequence just repeats itself with period $k - j$.

Pollard rho

- John Pollard, in the 1970s, created a number of different techniques for factoring large integers.
- The Pollard rho is named for a way of representing the iterative process which looks like the Greek lower case rho, ρ.
- Suppose you start from some object P_0, and successively compute $P_1, P_2, P_3, \ldots$ and that sooner or later you find some pair $j < k$ so that $P_j = P_k$.
- Then $P_{j+1} = P_{k+1}$ and so on.
- That is the sequence just repeats itself with period $k - j$.
- We can represent this as a ρ, where P_0 is at the base of the tail, and P_j is where the tail meets the loop.

- How this works to factorize n in the case of Pollard rho is that one chooses some polynomial, normally irreducible over $\mathbb{Q}$, like
$$f(x) = x^2 + 1,$$

- How this works to factorize n in the case of Pollard rho is that one chooses some polynomial, normally irreducible over $\mathbb{Q}$, like
$$f(x) = x^2 + 1,$$
- pick an x_0 at random and successively compute

$$\begin{aligned} x_1 &= f(x_0) \pmod{n}, \\ x_2 &= f(x_1) \pmod{n}, \\ x_3 &= f(x_2) \pmod{n}, \\ \vdots &\quad \vdots \qquad\quad \vdots \end{aligned}$$

- How this works to factorize n in the case of Pollard rho is that one chooses some polynomial, normally irreducible over $\mathbb{Q}$, like
$$f(x) = x^2 + 1,$$
- pick an x_0 at random and successively compute
$$\begin{aligned} x_1 &= f(x_0) \pmod{n}, \\ x_2 &= f(x_1) \pmod{n}, \\ x_3 &= f(x_2) \pmod{n}, \\ \vdots & \quad \vdots \qquad \vdots \end{aligned}$$
- Since there are only n residue classes, sooner or later there has to be a repetition. We then check $GCD(x_i - x_j, n)$ for each pair i, j and hope to find a non-trivial factor of n.

- How this works to factorize n in the case of Pollard rho is that one chooses some polynomial, normally irreducible over $\mathbb{Q}$, like

$$f(x) = x^2 + 1,$$

- pick an x_0 at random and successively compute

$$\begin{aligned} x_1 &= f(x_0) \pmod{n}, \\ x_2 &= f(x_1) \pmod{n}, \\ x_3 &= f(x_2) \pmod{n}, \\ &\vdots \end{aligned}$$

- Since there are only n residue classes, sooner or later there has to be a repetition. We then check $GCD(x_i - x_j, n)$ for each pair i, j and hope to find a non-trivial factor of n.
- There is no guarantee of finding one quickly, but sometimes one is found.

- How this works to factorize n in the case of Pollard rho is that one chooses some polynomial, normally irreducible over $\mathbb{Q}$, like
$$f(x) = x^2 + 1,$$
- pick an x_0 at random and successively compute
$$\begin{aligned} x_1 &= f(x_0) \pmod{n}, \\ x_2 &= f(x_1) \pmod{n}, \\ x_3 &= f(x_2) \pmod{n}, \\ &\vdots \quad \vdots \quad \vdots \end{aligned}$$
- Since there are only n residue classes, sooner or later there has to be a repetition. We then check $GCD(x_i - x_j, n)$ for each pair i, j and hope to find a non-trivial factor of n.
- There is no guarantee of finding one quickly, but sometimes one is found.
- The usual procedure is to stop after a certain amount of time and try a different polynomial f.

- What is the theory?

- What is the theory?
- Suppose d is a proper divisor of n.

- What is the theory?
- Suppose d is a proper divisor of n.
- For every i let $y_i \equiv x_i \pmod{d}$.

- What is the theory?
- Suppose d is a proper divisor of n.
- For every i let $y_i \equiv x_i \pmod{d}$.
- Then $y_j \equiv x_j \equiv f(x_{j-1}) \equiv f(y_{j-1}) \pmod{d}$.

- What is the theory?
- Suppose d is a proper divisor of n.
- For every i let $y_i \equiv x_i \pmod{d}$.
- Then $y_j \equiv x_j \equiv f(x_{j-1}) \equiv f(y_{j-1}) \pmod{d}$.
- Thus sooner or later $y_j = y_k$ for some j, k with $j \neq k$.

- What is the theory?
- Suppose d is a proper divisor of n.
- For every i let $y_i \equiv x_i \pmod{d}$.
- Then $y_j \equiv x_j \equiv f(x_{j-1}) \equiv f(y_{j-1}) \pmod{d}$.
- Thus sooner or later $y_j = y_k$ for some j, k with $j \neq k$.
- Then $x_j \equiv y_j \equiv y_k \equiv x_k \pmod{d}$. Probably, and hopefully, $x_j \neq x_k$ so $d | GCD(x_j - x_k, n)$ and the GCD will differ from n.

- How far should we expect to go before finding a solution?

- How far should we expect to go before finding a solution?
- Given a prime p with $p|n$ we are seeking different numbers in the same residue class modulo p.

- How far should we expect to go before finding a solution?
- Given a prime p with $p|n$ we are seeking different numbers in the same residue class modulo p.
- If we have $x_1, x_2, \ldots, x_s$ created at random, this is akin to the birthday paradox with a year that has p days and a class size of s.

- How far should we expect to go before finding a solution?
- Given a prime p with $p|n$ we are seeking different numbers in the same residue class modulo p.
- If we have $x_1, x_2, \ldots, x_s$ created at random, this is akin to the birthday paradox with a year that has p days and a class size of s.
- Thus we can expect that with s not much bigger than $\sqrt{p}$ we will find a solution.

Example 1

Let $n = 1133$ and $f(x) = x^2 + 1$. Of course $11|1133$.
Take $x_0 = 2$. Then $x_1 = 5$, $x_2 = 26$, $x_3 = 677$, $x_4 = 598$. Now

$$
\begin{aligned}
(x_1 - x_0, n) &= (3, 1133) = 1,\\
(x_2 - x_0, n) &= (24, 1133) = 1,\\
(x_3 - x_0, n) &= (675, 1133) = 1,\\
(x_4 - x_0, n) &= (596, 1133) = 1,\\
(x_2 - x_1, n) &= (21, 1133) = 1,\\
(x_3 - x_1, n) &= (672, 1133) = 1,\\
(x_4 - x_1, n) &= (593, 1133) = 1,\\
(x_3 - x_2, n) &= (651, 1133) = 1,\\
(x_4 - x_2, n) &= (572, 1133) = 11.
\end{aligned}
$$

Not very efficient, but it illustrates the idea.

- The method can be speeded up as follows by an idea due to Floyd.

- The method can be speeded up as follows by an idea due to Floyd.
- We want to know when we have reached the loop.

- The method can be speeded up as follows by an idea due to Floyd.
- We want to know when we have reached the loop.
- Think of this as a race with two runners.

- The method can be speeded up as follows by an idea due to Floyd.
- We want to know when we have reached the loop.
- Think of this as a race with two runners.
- If one is running twice as fast as the other, the point at which the faster one comes round the loop to overtake the slower one is the place where the tail meets the loop.

- The method can be speeded up as follows by an idea due to Floyd.
- We want to know when we have reached the loop.
- Think of this as a race with two runners.
- If one is running twice as fast as the other, the point at which the faster one comes round the loop to overtake the slower one is the place where the tail meets the loop.
- With this in mind, let $z_0 = x_0$ and then at the j-th step compute x_j as above and $z_{j+1} \equiv f(f(z_j)) \pmod{n}$.

- The method can be speeded up as follows by an idea due to Floyd.
- We want to know when we have reached the loop.
- Think of this as a race with two runners.
- If one is running twice as fast as the other, the point at which the faster one comes round the loop to overtake the slower one is the place where the tail meets the loop.
- With this in mind, let $z_0 = x_0$ and then at the j-th step compute x_j as above and $z_{j+1} \equiv f(f(z_j)) \pmod{n}$.
- Then $z_j = x_{2j}$, so we are computing x_j and x_{2j} simultaneously.

- The method can be speeded up as follows by an idea due to Floyd.
- We want to know when we have reached the loop.
- Think of this as a race with two runners.
- If one is running twice as fast as the other, the point at which the faster one comes round the loop to overtake the slower one is the place where the tail meets the loop.
- With this in mind, let $z_0 = x_0$ and then at the j-th step compute x_j as above and $z_{j+1} \equiv f(f(z_j)) \pmod{n}$.
- Then $z_j = x_{2j}$, so we are computing x_j and x_{2j} simultaneously.
- If x_j and x_k with $j < k$ are the smallest pair with $x_j \equiv x_k \pmod{d}$, let $l = k - j$. Then $x_i \equiv x_{i+rl} \pmod{d}$ for every $i \geq j$ and every $r \geq 0$.

- The method can be speeded up as follows by an idea due to Floyd.
- We want to know when we have reached the loop.
- Think of this as a race with two runners.
- If one is running twice as fast as the other, the point at which the faster one comes round the loop to overtake the slower one is the place where the tail meets the loop.
- With this in mind, let $z_0 = x_0$ and then at the j-th step compute x_j as above and $z_{j+1} \equiv f(f(z_j)) \pmod{n}$.
- Then $z_j = x_{2j}$, so we are computing x_j and x_{2j} simultaneously.
- If x_j and x_k with $j < k$ are the smallest pair with $x_j \equiv x_k \pmod{d}$, let $l = k - j$. Then $x_i \equiv x_{i+rl} \pmod{d}$ for every $i \geq j$ and every $r \geq 0$.
- Take $i = l\lceil j/l \rceil$ so that $i \geq j$ and $r = \lceil j/l \rceil$.

- The method can be speeded up as follows by an idea due to Floyd.
- We want to know when we have reached the loop.
- Think of this as a race with two runners.
- If one is running twice as fast as the other, the point at which the faster one comes round the loop to overtake the slower one is the place where the tail meets the loop.
- With this in mind, let $z_0 = x_0$ and then at the j-th step compute x_j as above and $z_{j+1} \equiv f(f(z_j)) \pmod{n}$.
- Then $z_j = x_{2j}$, so we are computing x_j and x_{2j} simultaneously.
- If x_j and x_k with $j < k$ are the smallest pair with $x_j \equiv x_k \pmod{d}$, let $l = k - j$. Then $x_i \equiv x_{i+rl} \pmod{d}$ for every $i \geq j$ and every $r \geq 0$.
- Take $i = l\lceil j/l \rceil$ so that $i \geq j$ and $r = \lceil j/l \rceil$.
- Then $rl = i$ and so $x_i \equiv x_{2i} \pmod{d}$. Thus we only need check $GCD(x_{2i} - x_i, n)$ and this really speeds up the computations. In the previous example.

- Thus we only need check $GCD(x_{2i} - x_i, n)$s.

- Thus we only need check $GCD(x_{2i} - x_i, n)$s.
- In the previous example.

Example 2

Let $n = 1133$, $f(x) = x^2 + 1$ and $x_0 = 2$.
Then we compute $x_1 = 5$, $x_2 = 26$, $x_3 = 677$, $x_4 = 598$.

$$x_1 = 5, x_2 = 26, \; (x_2 - x_1, n) = (21, 1133) = 1,$$
$$x_2 = 26, x_4 = 598, \; (x_4 - x_2, n) = (572, 1133) = 11.$$

That is more like it!

- A less obvious example

Example 3

Let $n = 713$, $f(x) = x^2 + 1$ and $x_0 = 2$.
Then we compute $x_1 = 5$, $x_2 = 26$, $x_3 = 677$, $x_4 = 584$.

$$x_1 = 5, x_2 = 26, \ (x_2 - x_1, n) = (21, 713) = 1,$$
$$x_2 = 26, x_4 = 584 \, (x_4 - x_2, n) = (558, 713) = 31.$$

- There are a number of more sophisticated variants of this which are designed to speed the algorithm up.

- There are a number of more sophisticated variants of this which are designed to speed the algorithm up.
- Generally there is no rigorous proof but it is believed that the run time is normally proportional to $\sqrt{p}$ where p is the smallest prime factor of n and so in the worst case, for a composite number the run time is proportional to $n^{1/4}$.

- There are a number of more sophisticated variants of this which are designed to speed the algorithm up.
- Generally there is no rigorous proof but it is believed that the run time is normally proportional to $\sqrt{p}$ where p is the smallest prime factor of n and so in the worst case, for a composite number the run time is proportional to $n^{1/4}$.
- One way to see that this might give the correct runtime in most cases is to look at it the following way.

- There are a number of more sophisticated variants of this which are designed to speed the algorithm up.
- Generally there is no rigorous proof but it is believed that the run time is normally proportional to $\sqrt{p}$ where p is the smallest prime factor of n and so in the worst case, for a composite number the run time is proportional to $n^{1/4}$.
- One way to see that this might give the correct runtime in most cases is to look at it the following way.
- Let p be the smallest prime factor of n, and suppose we have computed the first s values of x_j.

- There are a number of more sophisticated variants of this which are designed to speed the algorithm up.
- Generally there is no rigorous proof but it is believed that the run time is normally proportional to $\sqrt{p}$ where p is the smallest prime factor of n and so in the worst case, for a composite number the run time is proportional to $n^{1/4}$.
- One way to see that this might give the correct runtime in most cases is to look at it the following way.
- Let p be the smallest prime factor of n, and suppose we have computed the first s values of x_j.
- Then what is the chance that two of them will be in the same residue class modulo p?

- There are a number of more sophisticated variants of this which are designed to speed the algorithm up.
- Generally there is no rigorous proof but it is believed that the run time is normally proportional to $\sqrt{p}$ where p is the smallest prime factor of n and so in the worst case, for a composite number the run time is proportional to $n^{1/4}$.
- One way to see that this might give the correct runtime in most cases is to look at it the following way.
- Let p be the smallest prime factor of n, and suppose we have computed the first s values of x_j.
- Then what is the chance that two of them will be in the same residue class modulo p?
- Well there are only p residue classes modulo p.

- There are a number of more sophisticated variants of this which are designed to speed the algorithm up.
- Generally there is no rigorous proof but it is believed that the run time is normally proportional to $\sqrt{p}$ where p is the smallest prime factor of n and so in the worst case, for a composite number the run time is proportional to $n^{1/4}$.
- One way to see that this might give the correct runtime in most cases is to look at it the following way.
- Let p be the smallest prime factor of n, and suppose we have computed the first s values of x_j.
- Then what is the chance that two of them will be in the same residue class modulo p?
- Well there are only p residue classes modulo p.
- Wait a minute.

- There are a number of more sophisticated variants of this which are designed to speed the algorithm up.
- Generally there is no rigorous proof but it is believed that the run time is normally proportional to $\sqrt{p}$ where p is the smallest prime factor of n and so in the worst case, for a composite number the run time is proportional to $n^{1/4}$.
- One way to see that this might give the correct runtime in most cases is to look at it the following way.
- Let p be the smallest prime factor of n, and suppose we have computed the first s values of x_j.
- Then what is the chance that two of them will be in the same residue class modulo p?
- Well there are only p residue classes modulo p.
- Wait a minute.
- This is just an example of the generalised birthday paradox.

- There are a number of more sophisticated variants of this which are designed to speed the algorithm up.
- Generally there is no rigorous proof but it is believed that the run time is normally proportional to $\sqrt{p}$ where p is the smallest prime factor of n and so in the worst case, for a composite number the run time is proportional to $n^{1/4}$.
- One way to see that this might give the correct runtime in most cases is to look at it the following way.
- Let p be the smallest prime factor of n, and suppose we have computed the first s values of x_j.
- Then what is the chance that two of them will be in the same residue class modulo p?
- Well there are only p residue classes modulo p.
- Wait a minute.
- This is just an example of the generalised birthday paradox.
- So we can expect that if s is not much bigger than $\sqrt{p}$, then there will be a coincidence!

- Here we take a fairly large number K and hope that n has a prime factor p such that none of the prime factors of $p-1$ exceed K.

- Here we take a fairly large number K and hope that n has a prime factor p such that none of the prime factors of $p-1$ exceed K.
- To explain the method we will assume a little more, namely that $p-1|K!$

- Here we take a fairly large number K and hope that n has a prime factor p such that none of the prime factors of $p-1$ exceed K.
- To explain the method we will assume a little more, namely that $p-1|K!$
- Obviously we do not want to compute and store $K!$, which will be huge.

- Here we take a fairly large number K and hope that n has a prime factor p such that none of the prime factors of $p-1$ exceed K.
- To explain the method we will assume a little more, namely that $p-1|K!$
- Obviously we do not want to compute and store $K!$, which will be huge.
- Thus for some a coprime with n we define $x_1 = a$ and successively compute

$$x_k \equiv x_{k-1}^k \pmod{n} \;\&\; GCD(x_k - 1, n) \quad (k = 2, 3, \ldots, K),$$

stopping if the GCD reveals a proper factor of n.

- Here we take a fairly large number K and hope that n has a prime factor p such that none of the prime factors of $p-1$ exceed K.
- To explain the method we will assume a little more, namely that $p-1|K!$
- Obviously we do not want to compute and store $K!$, which will be huge.
- Thus for some a coprime with n we define $x_1 = a$ and successively compute

$$x_k \equiv x_{k-1}^k \pmod{n} \;\&\; GCD(x_k - 1, n) \quad (k = 2, 3, \ldots, K),$$

stopping if the GCD reveals a proper factor of n.
- Since n is large we can expect that $x_k \not\equiv 1 \pmod{n}$, but if $p|n$ and $p-1|k!$, so that $k! = m(p-1)$ for some m, then we have

$$x_k \equiv a^{k!} = (a^{p-1})^m \equiv 1 \pmod{p}.$$

- Consider our old friend 1133.

Example 4

Let $a = 2$. Thus $x_1 = 2, x_2 = 2^2 = 4, x_3 = 4^3 = 64,$

$$x_4 = 64^4 = 16777216 \equiv 719 \pmod{1133}, \quad (718, 1133) = 1,$$

$$x_5 = 719^5 = 192,151,797,699,599 \equiv 1101 \pmod{1133},$$

$$(1100, 1133) = 11.$$

- Consider our old friend 1133.

Example 4

Let $a = 2$. Thus $x_1 = 2, x_2 = 2^2 = 4, x_3 = 4^3 = 64$,

$$x_4 = 64^4 = 16777216 \equiv 719 \pmod{1133}, \ (718, 1133) = 1,$$

$$x_5 = 719^5 = 192, 151, 797, 699, 599 \equiv 1101 \pmod{1133},$$

$$(1100, 1133) = 11.$$

- Now look at the less obvious example we considered above

Example 5

Let $n = 713$, & $a = 2$. Thus $x_1 = 2, x_2 = 2^2 = 4, x_3 = 4^3 = 64$,

$x_4 = 64^4 = 16777216 \equiv 326 \pmod{713}$, $(325, 713) = 1$, $x_5 =$

$326^5 = 3, 682, 035, 745, 376 \equiv 311 \pmod{713}, (310, 713) = 31$

- In practice for large numbers the elliptic curve method is faster and the Pollard $p - 1$ has largely disappeared.

- In practice for large numbers the elliptic curve method is faster and the Pollard $p-1$ has largely disappeared.
- It uses the group structure of the powers of a modulo n.

- In practice for large numbers the elliptic curve method is faster and the Pollard $p-1$ has largely disappeared.
- It uses the group structure of the powers of a modulo n.
- The elliptic curve method is based on a similar basic idea but takes advantage of the richer underlying group structure of elliptic curves.