

**Pollard's Factoring Algorithm**  
**Exposition by William Gasarch**

## 1 Introduction

There is a trivial algorithm that factors  $N$  in time  $O(N^{1/2})$ . We will present Pollard's algorithm for factoring which is believed to have complexity  $O(N^{1/4})$  though this has not been proven. It works well in practice.

We take *factoring* to mean just finding a non-trivial factor. In practice we would use such an algorithm recursively.

## 2 We Seek $x, y$ such that $x \equiv y \pmod{p}$

We want to factor  $N$ . Let  $p$  be the smallest prime factor of  $N$ . Note that  $p \leq N^{1/2}$ . We do not know  $p$ . Lets say we somehow find  $x, y$  such that  $x \equiv y \pmod{p}$ . Then  $GCD(x-y, N)$  will likely yield a nontrivial factor of  $N$ . We look at several approaches to finding such an  $x, y$  that do not work before presenting the approach that does work.

## 3 Use Randomization!

Given  $N$  we generate a sequence of random numbers  $x_1, x_2, \dots \in [0, N-1]$ . Thought experiment: look at

$$x_1 \bmod p, x_2 \bmod p, \dots$$

This is a sequence of random elements in  $[0, p-1]$ . By the birthday paradox, with high probability there exists  $i, j \leq p^{1/2} \leq N^{1/4}$  such that  $x_i \pmod{p} = x_j \pmod{p}$ , or  $x_i \equiv x_j \pmod{p}$ .

So we could have an algorithm that generates this sequence and looks for repeats. NO WE CAN'T- we don't know  $p$ . But we can pretend that  $x_i \equiv x_j \pmod{p}$  and try  $GCD(x_i - x_j, N)$ . Which  $x_i, x_j$  do we do this for? ALL of them which is why this algorithm is too slow. Even so, here is the algorithm.

```

x_1 = RAND(0,N-1)
i=2
FOUND = FALSE
while NOT FOUND
{
  x_i := RAND(0,N-1)
  for j=1 to i-1
  {
    d=GCD(x_i-x_j,N)
    if (d NE 1) and (d NE N) then FOUND=TRUE
  }
  i=i+1
}
output(d)

```

Assume If  $x_i \equiv x_j \pmod{p}$  and  $x_i \neq x_j$ . Then  $x_i - x_j \equiv 0 \pmod{p}$ . Hence  $p$  divides  $d = GCD(x_i - x_j, N)$ . Therefore  $d \neq 1$ . Since  $x_i, x_j \in [0, N - 1]$ ,  $d \neq N$ . Hence if  $x_i \equiv x_j \pmod{p}$  then the algorithm will terminate.

Look at the sequence  $x_1 \bmod p, x_2 \bmod p, \dots$ . By the birthday paradox this sequence will almost surely have a repeat before  $O(p^{1/2})$  iterations. Hence the run time is almost surely bounded by

$$\sum_{i=1}^{p^{1/2}} \sum_{j=1}^{i-1} \log N \leq \log N \sum_{i=1}^{p^{1/2}} i = O(p) = O(N^{1/2}).$$

That's not better than the trivial algorithm. Oh well.

Also, the algorithm is a space hog.

## 4 Don't Use Randomization

The reason the last algorithm was a space hog is that it generated random numbers and had to store all of them. Instead we use a deterministic sequence that looks random.

The sequence that begins with a random  $x_1$  and  $c$ , and then does  $x_i := x_{i-1}^2 + c \pmod{N}$  appears random. This has not been proven (I am not even sure how you would state it); however, it does seem to have the property of repeating within  $O(p^{1/2})$  steps.

With this in mind we can write the algorithm which is no longer a space hog but still takes too much time.

```

x_1 = RAND(0,N-1)
c = RAND(0,N-1)
i=2
FOUND = FALSE
while NOT FOUND
  {
    x_i := x_{i-1}^2 + c mod N
    for j=1 to i-1
      {
        for k=2 to j
          x_k = x_{k-1}^2 + c.
        d=GCD(x_i-x_j,N)
        if (d NE 1) and (d NE N) then FOUND=TRUE
      }
    i=i+1
  }
output(d)

```

## 5 Using Cycle Detection

We plan to generate  $x_1, x_2, \dots$  deterministically. We need to find  $x_i, x_j$  such that  $x_i \equiv x_j \pmod{p}$  without storing too much or spending too much time.

We prove a lemma due to Floyd that is interesting in its own right.

**Lemma 5.1** *Let  $z_1, z_2, z_3, \dots$  be an infinite sequence. Let  $m$  be such that there is some  $i \leq m$  such that the sequence  $z_i, z_{i+1}, \dots$  is periodic with period  $\rho \leq m$ . Then there exists  $a \leq 2m$  such that  $z_a = z_{2a}$ .*

**Proof:**

Let  $a$  be such that  $(a-1)\rho \leq i < a\rho$ . Note that the sequence is  $a\rho$ -periodic.

Since the sequence is  $a\rho$ -periodic after  $z_i$  we have that, for all  $\Delta \geq 0$ ,  $z_{i+\Delta} = z_{i+a\rho+\Delta}$ . Plug in  $\Delta = a\rho - i$  (note that  $a\rho - i \geq 0$  by the case that we are in) to obtain.  $z_{a\rho} = z_{2a\rho}$ .

How big is  $a\rho$ ? We know that

$$a\rho/2 \leq (a-1)\rho \leq i \leq m, \text{ so } a\rho \leq 2m. \quad \blacksquare$$

We will form two sequences. One will be  $x_1, x_2, \dots$ . The other will be  $x_2, x_4, \dots$ . Given  $c$  we let  $f_c$  be the function  $f_c(x) \equiv x^2 + c \pmod{p}$ .

```

x = RAND(0,N-1)
c = RAND(0,N-1)
y = f_c(x)
FOUND = FALSE
while NOT FOUND
  {
    x := f_c(x)
    y := f_c(f_c(y))
    d=GCD(x-y,N)
    if (d NE 1) and (d NE N) then FOUND=TRUE
  }
output(d)

```

Consider the sequence  $x_1 = x$ ,  $x_i = f_c(x_{i-1})$ . Note that the  $x$ -sequence is  $x_1, x_2, x_3, \dots$  while the  $y$ -sequence is  $x_2, x_4, \dots$ . We assume that the sequence has the same properties as a random sequence. Let  $z_i = x_i \pmod{p}$ . This is also random. By the Birthday paradox it is highly likely that there is a repeat before  $O(p^{1/2})$  iterations. By Lemma 5.1 there exists  $a \leq p^{1/2}$  such that  $z_a = z_{2a}$ . When this occurs we have  $x - y \equiv 0 \pmod{p}$ , and hence  $d \neq 1$  and  $d \neq N$ .

With high prob this algorithm takes  $O(p^{1/2}) = O(N^{1/4})$  iterations. Each iteration only takes  $\log N$  steps. Hence the algorithm takes  $O(N^{1/4} \log N)$  steps.