- In the factoring algorithms, the time spent depends on the bound y & the probability bound.

- A time complexity of $L_n(\alpha, c)$, $\alpha < 1$, is termed <u>subexponential</u>, in contrast to the exponential $L_n(1, c)$.

  Eg. Eratosthenes sieve takes $L_n(1, \frac{1}{2})$ time.

Factoring algorithms

- We will start with algorithms that are better than the brute-force on certain n.

Pollard's rho method (1975)

- Idea is to exploit the presence of a "small" prime factor $p \mid n$.

Input: odd $n>1$ & a pseudorandom function $f(x)$ (say, $f(x) = x^2+1 \bmod n$).

Output: Factors n heuristically in $\tilde{O}(\sqrt{p}\cdot \lg n)$ time.

1) Randomly pick x; $y = x$; $d = 1$.
2) While $d = 1$
   - $x = f(x)$; $y = f\circ f(y)$.
   - $d = \gcd(x-y, n)$.
3) If $d \neq n$ then OUTPUT d, else FAIL.

Assumption: p is the smallest prime factor of n & $\{f^i(x) \mid i \geq 0\}$ is a random sequence.

Lemma: Whp $p \mid (x-y)$ after $O(\sqrt{p})$ iterations.

Pf:

- Consider the sequence $\{f^i(x) \bmod p \mid 0 \leq i \leq j\}$.
- The probability of them being distinct $\leq \frac{p}{p}\cdot\frac{p-1}{p}\cdots\frac{p-j}{p} \approx e^{-(\frac{1}{p}+\frac{2}{p}+\cdots+\frac{j}{p})} \approx e^{-j^2/p}$.

$\Rightarrow$ For $j = O(\sqrt{p})$, the probability of a repetition is good. (Birthday paradox)

• Thus, $\exists\ 0 \le i_1 < i_2$, $i_2 - i_1 = O(\sqrt{p})$ s.t. $f^{i_1}(x) \equiv f^{i_2}(x) \pmod{p}$.

$\Rightarrow$ After the i_1-th iteration the period $r \le i_2 - i_1 = O(\sqrt{p})$.

$\Rightarrow$ At the (i_1+t)-th iteration we get a <u>collision</u> if $(i_1+t) \equiv 2(i_1+t) \bmod r$,

$\Rightarrow$ $t = r - i_1$ works

$\Rightarrow$ $p \mid x-y$ at the $O(\sqrt{p})$-th iteration whp. □

– Thus, whp $p \mid d$ in $\tilde{O}(\sqrt{p} \cdot \lg n)$ time.

– Heuristically, we can say that this d will factor n.

<u>Success</u>: Brent & Pollard (1980) factored Fermat number $F_8 = 2^{2^8}+1$ into primes of

16 & 62 digits in 2 hours on a UNIVAC.

Pollard's $p-1$ method (1974)

- It exploits the smoothness of $(p-1)$, for a prime factor $p \mid n$.

Input: odd $n > 1$ not a perfect power.
Output: Factors n.

1) For $r = 2, 3, 4, \ldots\ldots$
   - Randomly pick $a \in (\mathbb{Z}/n\mathbb{Z})^*$
   - $d = (a^k - 1, n)$, $k = (r!)^{\lceil \lg n \rceil}$.
   - If $d \notin \{1, n\}$, then OUTPUT d.

Assumption: $\exists$ distinct primes $p, q \mid n$ s.t. $(p-1)$ is R-smooth but $(q-1)$ is not.

Lemma: Whp n is factored in $\tilde{O}(R \cdot \lg^2 n)$ time.

Proof: • When r reaches R: $\forall a \in (\mathbb{Z}/n\mathbb{Z})^*$,
$a^k \equiv 1 \pmod p$ $\quad [\because (p-1) \mid k]$.

• But, for $< \frac{1}{2}$ of the $a \in (\mathbb{Z}/n\mathbb{Z})^*$,
$a^k \not\equiv 1 \pmod q$.

$\Rightarrow$ Whp $p \mid (a^k - 1)$ & $q \nmid (a^k - 1)$.

• Time to compute $a^k - 1 \pmod n$ is:
$\tilde{O}(\lg k \cdot \lg n) = \tilde{O}(\lg n \cdot r \cdot \lg n)$.

$\Rightarrow$ Time overall (doing binary search $\leq R$)
$= \tilde{O}(R \cdot \lg^2 n)$. □

▷ If we run this for a single (large enough) r, then the complexity is $\tilde{O}(r \cdot \lg^2 n)$.

Success: In GIMPS (Great Internet Mersenne Prime Search), this is used to eliminate composites.

▷ In RSA, $(p-1)$ & $(q-1)$ should not be smooth!

Fermat's method

- Tries to write $n = a^2 - b^2$.

  Works well when a factor of n is very close to $\sqrt{n}$.

▷ $n = cd = \left(\frac{c+d}{2}\right)^2 - \left(\frac{c-d}{2}\right)^2$.

↑ both are odd.

▷ $c \approx \sqrt{n} \Leftrightarrow d \approx \sqrt{n} \Leftrightarrow (c-d)$ is "small".

- So, we can find $\frac{c-d}{2}$ by brute-force.

Algo: 1) For $x = 1, 2, 3, \ldots$

- If $(n+x^2)$ is square, then compute $y = \sqrt{n+x^2}$ & OUTPUT $(y-x)$.

▷ It has time complexity $\tilde{O}(m \cdot \lg n)$, where $m := \min\{c-d \mid n = c \cdot d\}$.

- (Lehman '74) made this general purpose with

time complexity $\tilde{O}(n^{1/3})$.

Success: The fundamental idea of finding two squares congruent modulo n, appears in many advanced algorithms.

Also, known as Kraitchik's family of algorithms. Starting idea from 1920s:

- Consider $Q(x) := X^2 - n$.
- Find $x_1, \dots, x_k$ s.t. $Q(x_1) \cdots Q(x_k)$ is a square v^2.

$\Rightarrow (x_1 \cdots x_k)^2 \equiv v^2 \pmod{n}$.

$\Rightarrow \gcd(x_1 \cdots x_k - v, n)$ might factor n!

Modification by Lehmer & Powers (1931).

- Compute the continued fraction (say, for k terms): $\sqrt{n} = a_0 + \frac{1}{a_1 +} \frac{1}{a_2 +} \frac{1}{a_3 +} \cdots$