CS 681: Computational Number Theory and Algebra	**Lecture 20**
Pollard's p-1 algorithm for factoring integers	
Lecturer: Manindra Agrawal	**Scribe: Chandan Saha**
	Septembor 22, 2005

In the previous lecture we have proven the following theorem:

Theorem 0.1 *If* $\psi(x,y) = \mid \{m \leq x \mid m \text{ is } y\text{-smooth}\} \mid$ *then, for* $y = \Omega(log^2 x)$, $\psi(x,y) \sim \frac{x}{u^u}$, *where* $u = \frac{ln\, x}{ln\, y}$.

Let $y = ln^2 x$ then $u = \frac{ln\, x}{2ln ln\, x}$. Therefore,

$$\begin{aligned}
\psi(x,y) &\sim \frac{x}{\left(\frac{ln\, x}{2lnln\, x}\right)^{\frac{ln\, x}{2lnln\, x}}} \\
&\sim \frac{x}{e^{\frac{1}{2}ln\, x}} \cdot e^{\frac{ln\, x \cdot lnlnln\, x}{2lnln\, x}} \\
&\sim x^{\frac{1}{2}} \cdot x^{\frac{lnlnln\, x}{2lnln\, x}} \\
&\sim x^{\frac{1}{2}+o(1)}
\end{aligned}$$

Problem: Find the smallest value of y such that $\psi(x,y) = \Omega(x)$.

1 Pollard's p-1 method for factoring

Let $n = pq$ be the number to be factored. Suppose $p-1$ be a k-smooth number. Let $K = (k!)^{lg\, p}$. By Fermat's Little Theorem, $a^K = 1\ (mod\, p)$. Suppose that, $q-1$ is not k-smooth. Then, the claim is that $a^K = 1\ (mod\, q)$ for 'few' a's. This is because, if $a^K = 1\ (mod\, q)$ then, $a^{gcd(K,q-1)} = 1\ (mod\, q)$. At most $gcd(K,q-1)$ of a's can satisfy the equation $a^{gcd(K,q-1)} = 1\ (mod\, q)$ and $gcd(K,q-1) \leq \frac{q-1}{2}$. This yields the following algorithm:

1.1 Algorithm

Input: Positive integer n.
Output: Either a proper divisor of n or 'failure'.
For $k = 2,3,4,\ldots$ do

1. Randomly select $a \in Z_n$.
2. $K \leftarrow (k!)^{(lg\, n)}$.
3. $b \leftarrow a^K\ (mod\, n)$.

4. $d \leftarrow gcd(b-1, n)$.

5. if $1 < d < n$ then return d else return 'failure'.

For the correct choice of k the above algorithm returns a proper divisor of n with probability greater than $\frac{1}{2}$. Since $(k!)^{log\, n} = ((k-1)!)^{log\, n} \cdot k^{log\, n}$, Step 2 requires $\tilde{O}(klog\, n \cdot log\, k)$ bit operations per iteration. Step 3 requires $\tilde{O}(klog^2 n \cdot log\, k)$ bit operations per iteration and Step 4 requires $\tilde{O}(log\, n)$ operations per iteration. Therefore time complexity of the above algorithm is $\tilde{O}(k^2 log^2 n \cdot log\, k)$ bit operations.