Pollard's ρ heuristic for Integer Factorization

Pollard's algorithm is based on three concepts:

1. Birthday Paradox
2. Pseudo Random Number generation
3. Cycle detection Algorithm

(1) Birthday Paradox: Let $Z_n = \{0, 1, \cdots n-1\}$ = complete residue system mod n. We randomly select elements from Z_n. By Birthday Paradox, ~~the exp~~ after $\theta(\sqrt{n})$ steps, with good probability (at least $\frac{1}{2}$) we get a repeated number.

(2) Pseudo Random Number generation: Generating truely random sequence is very difficult. Instead of generating truely random sequence, first we generate a small random "seed" x_0. Then we use the seed in a recursive formula (for example, $x_i \equiv (x_{i-1}^2 - 1) \pmod{n}$) to generate a sequence $x_0, x_1, x_2 \ldots$ which "looks" random. The sequence $x_0, x_1, x_2, \ldots$ is called a Pseudo Random Number sequence

(3) Cycle detection Algorithm: The sequence generated in Pollard's algorithm looks like the greek letter ρ (rho) with a possible tail (the non-repeating portion of the sequence $x_0, x_1, \ldots$) and a cycle (the repeating portion of the sequence $x_0, x_1, \ldots$).

The pointer jumping algorithm for cycle detection in a sequence: We use two pointers p_1 and p_2 in the algorithm. The pointer p_1 is incremented by 1 in each iteration. The sequence for p_1 is:

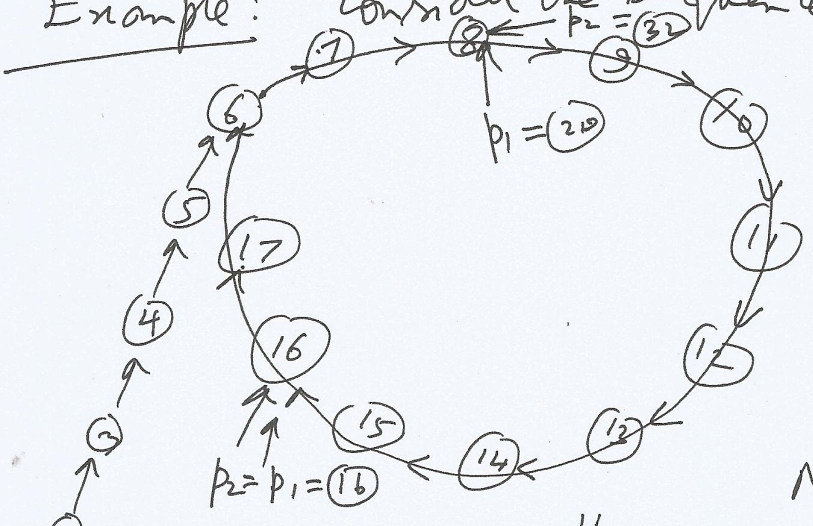
1, 2, 3, - - - .

The pointer p_2 is "jumped" in powers of 2. The sequence for p_2 is:

1, 2, 4, 8, 16, - - -

Initially, we take $p_1 = 1$ and $p_2 = 2$. We compare x_{p_1} with x_{p_2} untill $p_1 = p_2$. At this point, we jump p_2: $p_2 \leftarrow 2p_2$. We repeat the process. At some point in time, p_2 arrives in the cycle, and again, after some time, the value of p_2 is high enough in the cycle, so that we get $x_{p_1} = x_{p_2}$ before the condition of $p_1 = p_2$, and we have detected the cycle.

Example: Consider the sequence: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ____, - - -



Consider the point at which $p_1 = p_2 = 16$

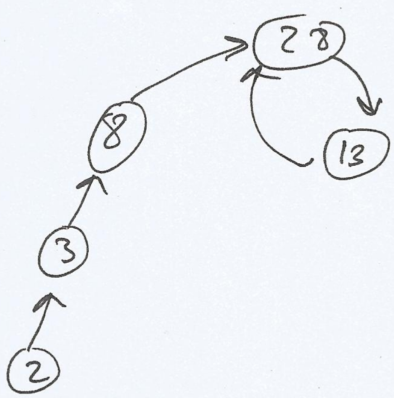
$p_2 \leftarrow 2p_2 = 32$

Now p_2 will point to 8

p_1 will "catch" p_2 at $p_1 = 20$ (just after 4 steps). Now we have detected a cycle. 8 is inside the cycle.

```
Pollard-Rho(n)
{ i ← 1;
  x_1 ← Random(0, n-1);
  y ← x_1;
  k ← 2;
  while(True)
  {  i ← i+1;
     x_i ← (x_{i-1}^2 - 1)(mod n);
     d ← gcd(y - x_i, n);
     if (d ≠ 1 and d ≠ n)
     {     print d;  }
     if (i == k)
     {     y ← x_i;
           k ← 2k;
     }
  }
}
```

Let us run the above algorithm for $n = 35$, with $x_1 = 2$:
The sequence $x_1, x_2, \ldots$ looks like:

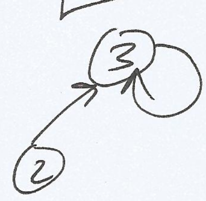


$x_i \pmod{35}$

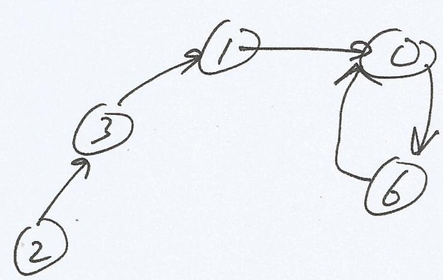
$i = 3$ gives

$d = \gcd(8-3, 35)$

$= \boxed{5}$



$x_i \pmod{5}$



$x_i \pmod{7}$

The sequence $x_i \pmod{p}$, where $p \mid n$, p is a prime, also makes a ρ. A cycle is detected in the sequence $x_i \pmod{p}$ if $x_k \equiv x_i \pmod{p} \iff x_k - x_i \equiv 0 \pmod{p}$
$\iff p \mid (x_k - x_i) \iff p \mid (y - x_i) \iff d = \gcd(y - x_i, n) > 1$.

After $O(\sqrt{p})$ steps, we can expect with good probability to find p as a factor (however, there is no gaurantee of this fact). Since for any prime $p \mid n$, $p \neq 1$, $p \neq n$, $p \leq \sqrt{n}$, we can expect in $O(n^{1/4})$ ~~time~~ steps to find a non-trivial factorization of n. This is much better than the brute-force method of checking all divisors upto $\sqrt{n}$ taking time $O(\sqrt{n})$ steps.