Pollard's Rho algorithm for prime factorization:

If a certainly large odd integer n is composite, then the method of trial division of primes $\leq \sqrt{n}$ can take more than $O(\sqrt{n})$ bit operations.

The simplest algorithm which is substantially faster than this is JM Pollard's "rho method".

1) The first step in the method is to choose an map from $\mathbb{Z}/n\mathbb{Z}$ to itself, i.e a fairly simple polynomial with integer coefficients, such as $f(x) = x^2 + 1$.

2) Choose some particular value $x = x_0$ and compute the successive iterates of f:

$$x_1 = f(x_0)$$
$$x_2 = f(f(x_0)).$$

i.e We define:

$$x_{j+1} = f(x_j), \quad j = 0, 1, 2, \ldots$$

3) Then we make comparisons b/w different x_j's hoping to find two which are in different residue classes mod n.

Once we find such x_j, x_k we have $\gcd(x_j - x_k, n)$ equal to proper divisor of n.

Algorithm :-

Step-1 Start with random x and c. Take y equal to x and $f(x) = x^2 + c$.

Step-2 While a divisor isn't obtained.

1.) Update x to f(x) (modulo n) (tortoise move)
2.) Update y to f(f(y)) mod n (Hare move)
3.) Calculate GCD of |x-y| and n.
4.) if GCD is not unity.
   (a) if GCD is n, Repeat from step 2 with another set of x, y and c
   (b) else GCD is our answer.

Ex. Let $n = 187$.

Case I. An example of random values such that the above algorithm finds results is.
$y = x = 2$ and $c = 1$
then $f(x) = x^2 + 1$.

$x_{i+1} = f(x_i)$ $\quad y_{i+1} = f(f(y_i))$ $\quad$ c $\quad d = gcd(|x-y|, n)$

1.) ~~x=2~~
putting $x_1 = 2$, we get $x_2 = 5$ | putting $x_1 = 2$ $y_2 = ((2)^2+1)^2+1$ $y_2 = 26$ | 1 | $gcd(21, 187) = 1$

2.) putting $x = 5$, we get = 26. | putting $x = 26$, we get 180. | 1 | $gcd(154, 187) = 11$

here, 11 is our one factor of 187.

Case-2 An example of random values where that algorithm doesn't find the result.

$x = y = 147$ and $c = 67$.

$f(x) = x^2 + 67$ — ①

$f(f(x)) = (x^2+67)^2 + 67$ — ②

$x_{i+1} = f(x_i)$ $\quad y_{i+1} = f(f(y_i))$ & $d = gcd(|x-y|, n)$

1.)	put $x = 147$ in eqn ① we get = 32	put $x = 147$ in eqn ② ; = 156.	67	1
2.)	put $x = 32$, ⇒ 156	put $x = 156$, 114	67	1.
3.)	put $x = 156$, ⇒ 93	put $x = 114$ ⇒ 48	67	1
4.)	put $x = 93$. ⇒ 114	put $x = 48$ ⇒ 114.	67	187.

NOTE:-

1.) Algorithm will run indefinitely for prime numbers.

2.) The algorithm may ~~fail~~ not find the factors and return a failure for composite n. In that case we use different set of x, y and c and try again.

3.) This algorithm only finds a divisor. To find a prime factor, we may recursively factorize divisor d.

Time Complexity of the Algorithm:-

* The algorithm offers a trade off between its running time and the probability that it finds a factor.

A prime divisor can be achieved with a probability around 0.5, in $O(\sqrt{d}) \leq O(n^{1/4})$ iterations.