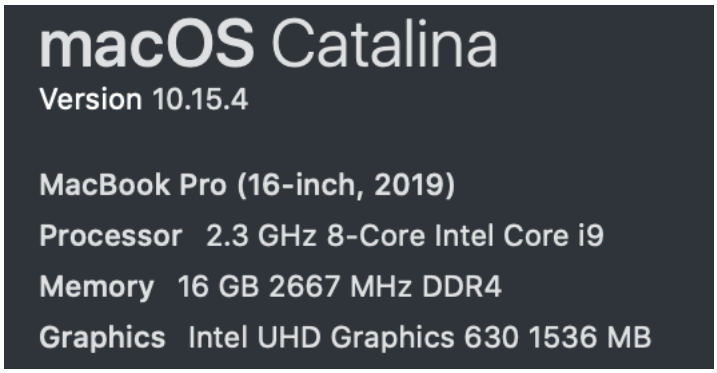
Build instruction for Integer Factorization

For CMSC441 SP2020 Project Part2

Implemented in **python**

Test environment:



Python 3.7.7

Library used:

```
##############################
# Libaray used in this program
# gmpy2 can be used as multiprecision lib stated in the piazza @56
##############################
from gmpy2 import *
from math import log, ceil
import sys
from random import *
import time
```

'gmpy2' must be installed before running the program

Run with **python3 ./intFactor_441.py**

Input any positive integer you want to factor

Exit with '**exit**' or '**quit**'

*** some codes are adapted from my previous project out of class

Overview:

Will try

- Trial Division
  - At very first for every input
  - Actually,it is useless in this project
- Pollard's p – 1
  - For tricky3 it does work for one time but it's stable
- Pollard's rho

Pollard rho:

1. Input: composite num n, and it is not a power of prime numbers.
2. Output: a nontrivial factor of n

– Random select $x \in \{0,1,\dots,n-1\}$, Polynomial $f(x) = x^2 + c \bmod n$, $c \neq 0, -2$

– For i : $g := \gcd\left(f^{2i}(x) - f^{i}(x), n\right)$

  - If $g = 1$, select next i
  - If $1 < g < n$, return g and terminate
  - If $g = n$,return step one selects x and c

Since under most condition the gcd = 1 multiply m $x_{2i} - x_i = f^{2i}(x) - f^{i}(x)$ such that $Q_k = \prod_{i=mk-m+1}^{mk}(x_{2i} - x_i) \bmod n$

In separate test I found that Brent's method is actually slower than pollard rho, maybe the reason is I implemented in a wrong way. So I remove Brent's method from the project

Brent's method:

1. Input: composite num n, and it is not a power of prime numbers.
2. Output: a nontrivial factor of n

3. Initialization: variable x and x^* store $x_m \bmod n$ and $x_{\ell(m)-1} \bmod n$, $l = \ell(m)$, $k = 2l - m$

- Euclidean algorithm: $g := \gcd(x - x^*, n)$
    - If $g = 1$, jump to next step ;
    - If $1 < g < n$, return g ;
    - If $g = n$, failed and terminated.
- $k := k - 1$
    - If $k = 0$, $x^* := x,\ l := 2l,\ k := l$
    - $x := x^2 + 1$, back to last step.

ANALYSIS:

The **time complexity** of the Pollard ρ method depends on the smaller prime factor p of the composite number n to be factored.

Generally, the non-trivial factor can be found after $\mathcal{O}(\sqrt{p}) = \mathcal{O}(n^{1/4})$ iterations (but possible **Not** a prime factor)

Pollard ρ **asymptotic time complexity** of the method is about $\mathcal{O}(\sqrt{p})$, for every prime factor p have 2 more digit, will increase by 10 times or more the average.

The Brent's method requires less than 10 ^ 6 iterations to find the decimal 10-bit and 11-bit factors, and when it is greater than 11, the algorithm's performance is unstable.

Pollard p – 1 method:

If the composite number n to be factored has a certain prime factor p, and p-1 has only a small prime factor, the algorithm can find this prime factor p with a high efficiency.

The mathematical principle on which the Pollard p-1 method relies is the Fermat 's Little theorem. From the perspective of algebra, the Pollard p-1 method is the natural result of

the LaGrange theorem on the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$ We suppose the composite number to be factored is N, the prime number p is a prime factor of N, and p> 2.

According to Fermat 's theorem: $a^{p-1} \equiv 1 \pmod{p}$， $\gcd(a,p) = 1$

Then we have $a^{K\cdot(p-1)} \equiv 1 \pmod{p}$, where K is any positive integer.

1. Input: composite num n, and boundary B.
2. Output: a nontrivial factor of n

- Construct a prime table $\mathbb{P} = \{2,3,5,7, \dots\}, \max_{p\in\mathbb{P}} p \le B$
- choose randomly $a_0 (2 \le a_0 < n)$, $a := a_0$
- For k = 1, 2, 3,…
    - $q := p_k \in \mathbb{P}$， p_k represent kth prime
    - $l := \lceil \log_q n \rceil$
    - $a := a^{q^l} \bmod n$
    - $g := \gcd(a-1, n)$

  If $g = 1$， next k

  If $g > 1$， return g;

  If $g = n$， return step 1 and select another a_0

The algorithm will fail in some cases.

The most common case is that p-1 has a prime factor much greater than B, then the Pollard p-1 method is not very applicable. There is also a very rare case, which is All prime factors of number n minus 1 have the same B, the algorithm is likely to get the trivial factor n itself, at this time you can modify the initial base **a.** The worst case is $n = (2p+1)(2q+1)$ where p, q, 2p + 1,2q + 1 are prime numbers and the scale of p, q is approximately the same. The time complexity of the algorithm is $\mathcal{O}\left(n^{\frac{1}{2}+\varepsilon}\right)$. In practice, it may even be less efficient than trial division

Reference:

Pollard J M. Theorems on factorization and primality testing

Can't finish p+1 implementation here are my thoughts

Williams p + 1 method:

Lucas sequence algorithm:

Input: initial P, the number of terms n

Output: nth item of sequence $V_n(P)$

1. n represent as $\sum_{i=0}^{t} b_i 2^{t-i}$ $(b_i = 0,1)$
2. if $n = 0$, return V_0 =2
3. if $n = 1$, return $V_1 = P$
4. for $k = 1,2, \dots t$

   If $b_k = 0$, then

   $$V_{f_{k+1}} := V_{f_k}^2 - 2 \bmod n$$

   $$V_{f_{k+1}-1} := V_{f_k} V_{f_k-1} - P \bmod n$$

   Else if $b_k = 1$, then

   $$V_{f_{k+1}} := P V_{f_k}^2 - V_{f_k} V_{f_k-1} - P \bmod n$$

   $$V_{f_{k+1}-1} := V_{f_k}^2 - 2 \bmod n$$

5. return V_{f_t} $(= V_n)$

p + 1 method:

input：composite number n, boundary B

output：a non-trivial factor of n

1. construct prime num table $\mathbb{P} = \{2,3,5,7, \dots\}, \max_{p \in \mathbb{P}} p \leq B$

2. randomly select $P_0 (3 \leq P_0 < n)$ let $\gcd(P_0^2 - 4, n) = 1$, $P := P_0$

3. for $k = 1,2,3, \ldots$

$q := p_k \in \mathbb{P}$, p_k represent nth prime number

$l := \lceil \log_q n \rceil$, $r_k := q^l$ $(r_k = p_k^{\lceil \log_{p_k} n \rceil})$

$P := V_{r_k}(P) \bmod n$ (where $V_{r_k}(P) = V_{r_1 r_2 \ldots r_k}(P_0)$)

$g := \gcd(P - 2, n)$

If $g = 1$, next k

If $g > 1$, return g and terminated

If $g = n$, back to step 1 and select another P_0

Analysis:

The efficiency does not depend on the composite number n itself being factored, but on the minimum prime factor p of the composite number n. Compared with the Pollard p-1 method, the p + 1 method takes more time to calculate the Lucas sequence compared with the modular exponentiation operation in Pollard p -1 method is doubled. Besides, when the initial value P_0 is replaced (at least 3 times, there is a probability of 7/8 so that $\left(\frac{D}{p}\right) = -1$), it takes much more time.

Although the efficiency of the Williams p+1 method is not as efficient as that of the Pollard p-1 method, the p+1 method has a wider range of application. The number that the pollard p-1 method can factor, theoretically the p+1 method can also do. As long as $p \pm 1$ id smooth, the Williams p+1 method will work.

Reference:

Williams, H. (1982). A $p + 1$ Method of Factoring. *Mathematics of Computation, 39*(159), 225-234. doi:10.2307/2007633

Lehmer, D. (1930). An Extended Theory of Lucas' Functions. *Annals of Mathematics, 31*(3), second series, 419-448. doi:10.2307/1968235

ECM:

ECM is a probabilistic integer factorization algorithm, which is a typical special integer factorization method. Its running time is only related to the prime factor p, and has nothing to do with the composite number n to be factored. The asymptotic time for ECM method is $\mathcal{O}\left(e^{c\sqrt{\log p \log\log p}}\right) = \mathcal{O}\left(p^{c\sqrt{\log\log p/\log p}}\right)$, where c is a constant related to the implementation details of the algorithm. Similar to the QS method, it is a sub-exponential algorithm, but when $\log p/\log n \ll 1/2$, ECM can find the prime factor p faster.

Input: positive int n, k int $x_1, x_2, \dots, x_k$ which is not divisible by n

Output: inverse mod (array of y…), or a non-trivial factor of n

1. set $t_1 := x_1$, for $i = 2, \dots, k$, set $t_i := t_{i-1} \cdot x_i \bmod n$
2. using extended Euclidean algorithm calc for u, v, g which satisfy $ut_k + vn = g$
   If $g = 1$ process to step 3;
   else if: $g = n$, calc $g = \gcd(n, x_i)$, for $i = 1, \dots, k$,
   until $g > 1$ or $i = k$, return non-trivial factor of n: g and terminate the alg.
3. For $i = k, k-1, \dots, 2$, calculate $y_i = ut_{i-1} \bmod n$, and set $u := ux_i \bmod n$; let $y_1 = u$ and terminate.

Input: odd composite number n wait to be factored (n is not a power of a prime number, gcd(n,6) = 1), bound B

Output: a non-trivial factor of n

1. randomly select $a \in \{1,2, \dots, n-1\}$, and determine the elliptic curve $y^2 = x^3 + ax + 1 (\bmod n)$, $4a^3 + 27 \not\equiv 0 (\bmod n)$, point $P := (0,1)$

2. construct a prime table $\mathbb{P} = \{2,3,5,7, \dots\}, \max_{p \in \mathbb{P}} p \leq B$

3. for each $q \in \mathbb{P}$

$l = \lceil \log_q B \rceil$

$k = q^l$

Calculate $kP \pmod n$,

If failed:

Find the irreversible element t, return $g = \gcd(t, n)$, terminate;

Else set $P := kP$, if $P = \infty$, failed, back to step 1 and take another elliptic curve; otherwise, take next q, continue the loop.

Each iteration of the algorithm calculates the point addition and number multiplication on multiple elliptic curves, but through the parallel modular inverse algorithm, each time the point addition of multiple elliptic curves needs only use one extended Euclidean algorithm for one time, which greatly improves effectiveness.

This program can easily find the prime factors of decimal 10 to 20 digits. The ECM method is more sensitive to the two parameters of the smooth boundary B and the number of elliptic curves C. The parameters B and C are related to the scale of the prime factor. The parameters are as following table:

$\log_{10} p$	# trials	B_1
12	50	125
13	53	250
14	57	500
15	62	830
16	68	1500
17	75	2500
18	85	4200
19	100	6500
20	120	10000
21	145	15000
22	175	22000
23	210	32000
24	250	45000
25	300	65000
26	400	85000
27	500	115000
28	650	155000
29	750	205000
30	950	275000

Table Reference:

Computational Algebra and Number Theory page 133.

Can't complete SIQS …. Here are my thoughts

SIQS:

Input: positive int n， boundary B， interval $[a,b] \subseteq \left[\sqrt{n}, \sqrt{2n}\right]$

Output: all B-smooth numbers in $Q(x) = x^2 - n, x \in [a,b]$

1. prime table $\mathbb{P} = \{2,3,5,7,\dots\}, \max_{p \in \mathbb{P}} p \leq B$，

2. initialize the array $q[a \dots b]$, satisfy $q[i] = Q(i), a \leq i \leq b$, set $p := 1$

3. if $p = 1$, then set $p := 2$; otherwise, variable p takes next prime number. If $p \notin \mathbb{P}$, jump to step 8; otherwise, $p \in \mathbb{P}$, jump to step 4.

4. solve all the solution of the quadratic congruence equation $x^2 \equiv n \pmod{p}$, set s so that $s^2 \equiv n \pmod{p}$

5. set $x := s + \lceil (a-s)/p \rceil \cdot p$

6. set $q[x] := q[x]/p$， until p does not divide $q[x]$

7. set $x := x + p$， if $x \leq b$, jump to step 5; else, jump to step 3

8. For the array q, x takes values sequentially from a to b. If q[x] = 1, then Q(x) is B-smooth; otherwise Q(x) is not a smooth number

In step 2, initialize the array $q[i] = \log Q(i), a \leq i \leq b$, here does not require accurate logarithm operation, rounding will work here. So that the trial division in step 6 will be $q[x] := q[x] - \log p$. In the last step 8, if $q[x]$ is very close to "0" within a certain error, use trial division to verify whether $Q(x)$ is B-smooth or not. Although this may miss a small part of the smooth number, since only a small number of $Q(x)$ will use trial division, the efficiency of the secondary sieve method can be significantly improved in this way.

When looking for small factors, the ECM method still has considerable advantages. The SIQS method is suitable for factoring large integers without "small" prime factors. The quadratic sieve method is very sensitive to the size of the factor base. The factor base is small, then less smooth number needs to be found, but the speed of searching for smooth numbers is very slow; the factor base is larger, and the speed of finding smooth

numbers is very fast, but there are many numbers to find, and the constructed matrix will be very large.

Reference:
https://pdfs.semanticscholar.org/5c52/8a975c1405bd35c65993abf5a4edb667c1db.pdf