Part II Computational Projects 2020

C3331E

Attach to front of report

Project number:

15.1

CATAM Part II - 15.1 - Primality Tests

Introduction

Throughout this project I code in python, using some inbuilt functions from the math package. The only non trivial function used is gcd, in accordance with the programming note at the end of the project.

Question 1

The function *trial_division(n)* takes an integer n as input and returns a boolean True or False. We make a small optimisation by only testing 2 and odd divisors. The function *q1()* runs the tests on the given ranges, and returns the following:

Primes in range: $[1900, 1999]$ are $\{1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999\}$
Primes in range: $[1294268500, 1294268700]$ are $\{\}$

Question 2

First consider the problem of computing $a^b \mod N$. We do this by considering the binary representation of $b = \sum_{i=0}^{N} 2^i b_i$ where each b_i is 0 or 1. So then we may write a^b as $a^{b_0}(a^2)^{b_1} \dots (a^{2^N})^{b_N}$ and each individual power can be calculated iteratively $a^{2^k} = (a^{2^{k-1}})^2 \mod N$. We can then multiply the terms $(a^{2^k})^{b_k}$, taking the result (mod N) at each stage, hence the largest number ever stored is order N^2. This however is not good enough, so we must devise a cleverer way of multiplying mod N.

Consider multiplying numbers x and y. We write for y even, $xy = (2x)(\frac{y}{2})$ and for y odd $xy = x + (2x)(\frac{y-1}{2})$. Initialise result to 0. In the y odd case, add $x \pmod N$ to the result. Now iterate the process with $x' = 2x$ and $y' = \frac{y}{2}$ or $\frac{y-1}{2}$ accordingly. The process terminates when y becomes 0, which must occur as y becomes strictly smaller each iteration. In doing so we take mod N after every step and are just doing addition so worst case we store at most $2N < 2 \times 10^{10} << 10^{15}$.

These are implemented in $modular\_multiply(a, b, mod)$, and
$modular\_exponent(base, exponent, modulo)$. The precise algorithms we use are listed below, to help us analyse complexity:

```
modular_multiply(a, b, N):
    result = 0
    a = a mod N
    while b>0:
        if b % 2 == 1:
            result = (result + a) mod N
        a = (2 * a) mod N
        b = floor(b/2)
    return result


modular_exponent(base, exponent, N):
    result = 1
    base = base mod N
    while exponent > 0:
        if exponent mod 2 == 1:
            result = modular_multiply(result, base, modulo)
        exponent = floor(exponent / 2)
        base = modular_multiply(base, base, modulo)
    return result
```

The function $fermat_test(a, N)$ performs the test and the wrapper $q2()$ finds the numbers passing to base a in the designated ranges:

```
2: {1901, 1905, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993,
    1997, 1999}
```

```
3: {1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997,
    1999}

4: {1901, 1905, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993,
    1997, 1999}

5: {1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997,
    1999}

6: {1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997,
    1999}

7: {1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997,
    1999}

8: {1901, 1905, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993,
    1997, 1999},

9: {1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997,
    1999}

10: {1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997,
    1999}

11: {1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997,
    1999}

12: {1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997,
    1999}

13: {1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997,
    1999}
```

These results are consistent with question 1; we expect primes to pass the Fermat test to each base, hence be included in each of the above lists. The only pseudoprime in the above lists is 1905, to bases 2,4 and 8.

To analyse complexity, fix an a and N. To compute $a^{N-1} \mod N$, we have one loop, with the exponent halving each time, so $O(\log(N))$ iterations. Each iteration has 2 modular multiplications, each of which is multiplying 2 numbers of any size $< N$. Our modular multiplication algorithm also runs in $O(\log N)$ as $y = O(N)$ is halved at each step. So our algorithm is $O(\log^2 N)$ or $O(n^2)$ for n the input size or number of bits $n = \log N$.

Question 3

First, we compute the so called Sarrus numbers, by applying the Fermat test to each integer, and checking for and ignoring primes using trial division. We get the following list of length 245 using $q3i()$:

```
{341, 561, 645, 1105, 1387, 1729, 1905, 2047, 2465, 2701, 2821, 3277, 4033,
    4369, 4371, 4681, 5461, 6601, 7957, 8321, 8481, 8911, 10261, 10585, 11305,
    12801, 13741, 13747, 13981, 14491, 15709, 15841, 16705, 18705, 18721, 19951,
    23001, 23377, 25761, 29341, 30121, 30889, 31417, 31609, 31621, 33153, 34945,
    35333, 39865, 41041, 41665, 42799, 46657, 49141, 49981, 52633, 55245, 57421,
    60701, 60787, 62745, 63973, 65077, 65281, 68101, 72885, 74665, 75361, 80581,
    83333, 83665, 85489, 87249, 88357, 88561, 90751, 91001, 93961, 101101,
    104653, 107185, 113201, 115921, 121465, 123251, 126217, 129889, 129921,
    130561, 137149, 149281, 150851, 154101, 157641, 158369, 162193, 162401,
    164737, 172081, 176149, 181901, 188057, 188461, 194221, 196021, 196093,
    204001, 206601, 208465, 212421, 215265, 215749, 219781, 220729, 223345,
    226801, 228241, 233017, 241001, 249841, 252601, 253241, 256999, 258511,
    264773, 266305, 271951, 272251, 275887, 276013, 278545, 280601, 282133,
    284581, 285541, 289941, 294271, 294409, 314821, 318361, 323713, 332949,
    334153, 340561, 341497, 348161, 357761, 367081, 387731, 390937, 396271,
```

```
399001, 401401, 410041, 422659, 423793, 427233, 435671, 443719, 448921,
449065, 451905, 452051, 458989, 464185, 476971, 481573, 486737, 488881,
489997, 493697, 493885, 512461, 513629, 514447, 526593, 530881, 534061,
552721, 556169, 563473, 574561, 574861, 580337, 582289, 587861, 588745,
604117, 611701, 617093, 622909, 625921, 635401, 642001, 647089, 653333,
656601, 657901, 658801, 665281, 665333, 665401, 670033, 672487, 679729,
680627, 683761, 688213, 710533, 711361, 721801, 722201, 722261, 729061,
738541, 741751, 742813, 743665, 745889, 748657, 757945, 769567, 769757,
786961, 800605, 818201, 825265, 831405, 838201, 838861, 841681, 847261,
852481, 852841, 873181, 875161, 877099, 898705, 915981, 916327, 934021,
950797, 976873, 983401, 997633}
```

Secondly, we compute the Carmichael numbers, $n < 10^6$. It suffices to check which base 2 pseudoprimes are psuedoprimes for all larger bases. We get the following list of 43 integers:

```
{ 561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041, 46657,
  52633, 62745, 63973, 75361, 101101, 115921, 126217, 162401, 172081, 188461,
  252601, 278545, 294409, 314821, 334153, 340561, 399001, 410041, 449065,
  488881, 512461, 530881, 552721, 656601, 658801, 670033, 748657, 825265,
  838201, 852841, 997633 }
```

We use the following simple observations to speed up our checking:

1) We can assume n is odd. To see this, suppose $n \geq 4$ is even. Then $(n-1)^{n-1} \equiv (-1)^{n-1} \equiv -1 \mod n$ so n fails the Fermat test base $n-1$.

2) We need only check bases up to $\lceil \frac{n}{2} \rceil$. This is because $(-a)^{n-1} \equiv (-1)^{n-1}a^{n-1} \equiv a^{n-1} \mod n$ since n is odd.

To determine how many bases we actually need to get this list, we'll start with the Sarrus numbers minus the Carmichael's. Increment the base a, and remove any numbers failing the fermat test to base a. Repeat this process until no numbers remain in our list. Running $q3ii()$, we find testing up to $a = 11$ suffices, with 721801 the only Sarrus number needing this large a base.

Question 4

First we calculate the Jacobi Symbol, using the following simple properties of it:

$$(\frac{a}{N}) = (\frac{a \mod N}{N}) \tag{1}$$

$$(\frac{ab}{N}) = (\frac{a}{N})(\frac{b}{N}) \tag{2}$$

$$(\frac{a}{N}) = \begin{cases} 0 & \text{if } \gcd(a,N) \neq 1, \\ \pm 1 & \text{if } \gcd(a,N) = 1. \end{cases} \tag{3}$$

$$(\frac{-1}{N}) = (-1)^{\frac{N-1}{2}} = \begin{cases} 1 & \text{if } n \equiv 1 \pmod 4, \\ -1 & \text{if } n \equiv 3 \pmod 4, \end{cases} \tag{4}$$

$$(\frac{2}{N}) = (-1)^{\frac{N^2-1}{8}} = \begin{cases} 1 & \text{if } n \equiv 1,7 \pmod 8, \\ -1 & \text{if } n \equiv 3,5 \pmod 8. \end{cases} \tag{5}$$

and for M, N odd coprime integers

$$(\frac{M}{N})(\frac{N}{M}) = (-1)^{\frac{(N-1)(M-1)}{4}} = \begin{cases} 1 & \text{if } N \equiv 1 \pmod 4 \text{ or } M \equiv 1 \pmod 4, \\ -1 & \text{if } N \equiv M \equiv 3 \pmod 4 \end{cases} \tag{*}$$

The algorithm used is as follows, identical to calculations performed on IIC Number Theory Example Sheets.

1) Reduce the numerator modulo the denominator using rule 1
2) Extract even numerators using rules 2 and 5
3) If the numerator is 1, we get 1. If the numerator and denominator are not coprime, rule 3 gives a result of 0.

4) Otherwise, have odd coprime integers, so flip the symbol using (*)

At each stage the denominator is strictly smaller, so the process must terminate. We implement this algorithm in $jacobi\_calculate(a, b)$ and also the Euler test in $euler\_test(a, N)$

The following results are found using $q4()$. Firstly, we compute the Euler pseudoprimes base 2, finding 114 of them:

```
{561, 1105, 1729, 1905, 2047, 2465, 3277, 4033, 4681, 6601, 8321, 8481, 10585,
    12801, 15841, 16705, 18705, 25761, 29341, 30121, 33153, 34945, 41041, 42799,
    46657, 49141, 52633, 62745, 65281, 74665, 75361, 80581, 85489, 87249, 88357,
    90751, 104653, 113201, 115921, 126217, 129921, 130561, 149281, 158369,
    162401, 164737, 172081, 188057, 196093, 208465, 215265, 220729, 223345,
    233017, 252601, 253241, 256999, 266305, 271951, 278545, 280601, 294409,
    314821, 323713, 334153, 340561, 348161, 357761, 390937, 399001, 410041,
    427233, 448921, 449065, 458989, 476971, 486737, 488881, 489997, 493697,
    514447, 526593, 530881, 552721, 580337, 588745, 625921, 635401, 647089,
    656601, 658801, 665281, 670033, 683761, 711361, 721801, 741751, 745889,
    748657, 800605, 818201, 825265, 838201, 838861, 841681, 852481, 852841,
    873181, 875161, 877099, 916327, 976873, 983401, 997633}
```

We find there are no absolute euler pseudoprimes in the range. In fact there are non at all, as shown in the IIC Number Theory Course.

We use a similar algorithm to question 3 to determine how many bases we need. Running $q4ii()$ we find we need up to base 13, with 399001 the only number needing this high a base.

Question 5

We implement the test in $strong\_test(a, N)$. Using $q5()$ we find the 46 strong pseudoprimes of base 2 to be:

```
{2047, 3277, 4033, 4681, 8321, 15841, 29341, 42799, 49141, 52633, 65281, 74665,
    80581, 85489, 88357, 90751, 104653, 130561, 196093, 220729, 233017, 252601,
    253241, 256999, 271951, 280601, 314821, 357761, 390937, 458989, 476971,
    486737, 489997, 514447, 580337, 635401, 647089, 741751, 800605, 818201,
    838861, 873181, 877099, 916327, 976873, 983401}
```

and again there are no absolute strong pseudoprimes.

Via $q5ii()$ we see that only bases 2 and 3 are needed to determine the primality of all integers in the range.

Question 6

k	Fermat	Euler	strong	primes
5	28	13	3	8392
6	16	9	4	7216
7	6	4	0	6241
8	1	0	0	5411
9	0	0	0	4832

Table 1: $a = 2$

k	Fermat	Euler	strong	primes
5	28	17	7	8392
6	13	9	3	7216
7	5	4	0	6241
8	1	1	1	5411
9	0	0	0	4832

Table 2: $a = 3$

k	fermat	euler	strong	primes
5	9	3	0	8392
6	4	2	0	7216
7	2	2	0	6241
8	1	0	0	5411
9	0	0	0	4832

Table 3: $a = 2\&3$

We have the relationship, Strong $\Rightarrow$ Euler $\Rightarrow$ Fermat, where A $\Rightarrow$ B means if n passes A for some base a then it will also pass B for base a. This is confirmed by the decreasing numbers moving right across the tables.

The Euler $\Rightarrow$ Fermat implication is clear, since if $a^{(N-1)/2} \equiv \pm 1 \mod N$ then $a^{N-1} \equiv 1 \mod N$ by squaring, while the other implication is a little more involved.

Question 7

It turns out trial division is one of the better strategies to run on numbers this small. The reason being at worst, we end up only trial dividing numbers up to about 100000, and can avoid most of them if we are clever about it. This really isn't that many operations, and checking divisibility is comparatively easy. On the other hand, consider the tests available to us. The fermat test is fairly useless, since it can't tell us with any certainty if a number is prime. Both the euler and fermat tests are ok at detecting compositeness, but to check primality with them is hard (unless we've done some work before to limit how many bases we need to check). They also carry a heavier computational cost, since modular exponentiation, and calculating the jacobi symbol are non trivial.

It's worth noting since there are about $\frac{x}{\ln x}$ primes up to x, so most of our random sample will be composite. This suggests initially using trial division will pick up most composites quickly, as shown in question 1. We can be clever about how we do this, and picking small primes to test will eliminate the most composites of any choice of divisors. Lets fix this as primes up to 250 for now. After this, we employ a test. The strong test being the stronger of the two (as shown in q6) seems like a better choice, despite being potentially more computationally demanding. Since our tests our so demanding, we are only going to perform a small number of them, and revert to trial division after. We test various numbers of fixed bases used for the strong test stage over 100000 randomly generated integers in the range, and time the total time to determine the primality of them all.

bases	2	2,3	2,3,5	2,3,5,7
trial division	17.783	17.487	18.6	17.1
my test	19.099	20.785	24.7	25.8

Table 4: Seconds to run the two tests with varying bases for the strong test

So we don't improve on trial division at all, in fact we make it worse every time we add a strong test base. This may be due to inefficient coding of the tests, but is also unsurprising in a way since we still are relying on trial division mostly, and simply using the strong test to reduce checking slightly. Thankfully, we can exploit some theory and come up with a much better test. C. Pomerance et al. [1] showed in 1980 that for $n < 25,000,000,000$ the only strong pseudo prime to all of bases 2,3,5 and 7 is 3,215,031,751 and so we can skip the trial division after. Doing so, we improve on our time massively. We also now test the impact of the small prime trial division at the start, and find checking primes up to 250 is quickest, giving about a 2x speedup on trial division

primes up to	0	50	100	150	200	250	300
running time	41.23	12.29	11.88	10.55	9.84	9.41	10.01

Table 5: Seconds to run my test with varying number of small primes trial divided at the start

The result in the article [1] is in a similar flavour to our thoughts before of how many bases would suffice to determine primality up to 10^6. Similar results exist for larger numbers with different/more bases, enabling this test to extend well to larger numbers. Trial division on the other hand becomes impossible when $n > 10^{20}$ or

[1] C. Pomerance, J. L. Selfridge and Wagstaff, Jr., S. S., "The pseudoprimes to $25 \cdot 109$," Math. Comp., 35:151 (1980) 1003-1026. Page 1023 in particular

so, and our prime tests, while detecting compositeness quickly, will struggle to verify primality without the aid of these stronger results.

Question 8

Unsurprisingly, we find the strong test is very good at detecting composites. My implementation may be found under $q8()$. We take a random sample of 10^6 N in the range. If N prime, we record it as having passed all rounds of the strong test. Otherwise, we pick a random a coprime to N, and conduct the strong test. We repeat the strong test with more bases going until the composite no longer passes the strong test.

We run trials for k ranging from 15 through 22, and got no composites surviving more than 3 rounds of the strong test.

k/t	**1**	**2**	**3**	**Primes**
22	43	4	0*	140336
21	31	2	0	73586
20	19	3	1	38645
19	22	2	1	20390
18	12	1	1	10749
17	15	1	0	5709
16	11	1	0	3030
15	9	0	0	1612

Table 6: Raw number of composites passing the strong test for random bases over 10^6 trials, through t rounds, and total number of primes in range

These probabilities agree but are all very small compared to theory, from which we get the following bound:

A composite passes t rounds of the strong test to co-prime bases with probability at most $\frac{1}{4^k}$ found in the IIC Number Theory course.

From this we'd expect the cell marked * for example to have magnitude at most 10^5, and it is clearly much lower.

Code

```
import random
import time
from collections import defaultdict
from math import floor, sqrt, gcd, ceil

second_trial_division_used = False


def trial_division(n):
    # implements naive trial division to decide whether n is prime
    if n % 2 == 0:
        return False

    for i in range(3, floor(sqrt(n)) + 1, 2):  # only do odds
        if n % i == 0:
            return False
    return True


def modular_multiply(a, b, mod):
    result = 0  # Initialize
    a = a % mod
    while b > 0:
        if b % 2 == 1:
            result = (result + a) % mod
        a = (2 * a) % mod
        b = b // 2  # floor b/2
    return result


def modular_exponent(base, exponent, modulo):
    # takes the exponent without overflow issues, by doing so in powers of 2 at a time

    result = 1
    base = base % modulo
    while exponent > 0:
        if exponent % 2 == 1:
            result = modular_multiply(result, base, modulo)
        exponent = floor(exponent / 2)
        base = modular_multiply(base, base, modulo)
    return result


def fermat_test(a, N):
    if modular_exponent(a, N - 1, N) == 1:
        return True
    else:
        return False


def jacobi_calculate(a, b):
    result = 1
    while True:
        a = int(a % b)  # step 1

        if a % 2 == 0:  # step 2
            a = floor(a / 2)
            if b % 8 == 3 or b % 8 == 5:
                result = -result
            continue

        if a == 1:
            break
```

```
        if gcd(a, b) != 1:
            result = 0
            break

        if a % 4 == 3 and b % 4 == 3:
            result = -result
        a, b = b, a

    return result


def euler_test(a, N):
    result = modular_exponent(a, int((N - 1) / 2), N)
    if result == 1 or result == N - 1:
        if result == jacobi_calculate(a, N) % N:
            return True
    return False


def strong_test(a, N):
    r = 0
    s = N - 1
    while True:
        if s % 2 == 0:
            r += 1
            s = s / 2
        else:
            break

    s = int(s)

    temp = modular_exponent(a, s, N)
    if temp == 1 or temp == N - 1:
        return True
    else:
        for k in range(1, r):  # k is the power of 2 in the test
            if modular_exponent(a, (2 ** k) * s, N) == N - 1:
                return True

    # if none of these are true return false
    return False


def q1():
    ranges = [range(1900, 2000), range(1294268500, 1294268701)]
    primes = defaultdict(list)

    for testrange in range(len(ranges)):
        primes[testrange] = []
        for n in ranges[testrange]:
            if trial_division(n):
                primes[testrange].append(n)

    return primes


def q2():
    ranges = [range(1900, 2000), range(1294268500, 1294268701)]
    fermat_passed_a = defaultdict()  # holds the numbers passing the fermat test for each a
    fermat_passed = defaultdict()  # holds the numbers passing the fermat test for all a
    for testrange in range(len(ranges)):
        fermat_passed_a[testrange] = defaultdict()
        for a in range(2, 14):
            print(a)
            fermat_passed_a[testrange][a] = []
```

```
            for n in ranges[testrange]:
                if fermat_test(a, n):
                    fermat_passed_a[testrange][a].append(n)

    return fermat_passed_a


def q3i():
    pseudoprimes_base_two = []
    for n in range(1, 10 ** 6 + 1):
        if fermat_test(2, n) and not trial_division(n):
            pseudoprimes_base_two.append(n)

    def isCarmichael(n):
        composite = False
        for a in range(3, ceil(n / 2) + 1):  # we've already checked 2
            if gcd(a, n) == 1:
                if not fermat_test(a, n):
                    return False
            else:
                composite = True
        return composite  # if it gets here it must have passed all the rounds of testing

    carmichaels = []
    for n in pseudoprimes_base_two:
        if isCarmichael(n):
            carmichaels.append(n)
            print(n)

    return carmichaels, pseudoprimes_base_two


def q3ii():
    sarrus = [341, 561, 645, 1105, 1387, 1729, 1905, 2047, 2465, 2701, 2821, 3277, 4033, 4369, 4371,
              4681, 5461, 6601,
              7957, 8321, 8481, 8911, 10261, 10585, 11305, 12801, 13741, 13747, 13981, 14491, 15709,
              15841, 16705,
              18705, 18721, 19951, 23001, 23377, 25761, 29341, 30121, 30889, 31417, 31609, 31621,
              33153, 34945, 35333,
              39865, 41041, 41665, 42799, 46657, 49141, 49981, 52633, 55245, 57421, 60701, 60787,
              62745, 63973, 65077,
              65281, 68101, 72885, 74665, 75361, 80581, 83333, 83665, 85489, 87249, 88357, 88561,
              90751, 91001, 93961,
              101101, 104653, 107185, 113201, 115921, 121465, 123251, 126217, 129889, 129921,
              130561, 137149, 149281,
              150851, 154101, 157641, 158369, 162193, 162401, 164737, 172081, 176149, 181901,
              188057, 188461, 194221,
              196021, 196093, 204001, 206601, 208465, 212421, 215265, 215749, 219781, 220729,
              223345, 226801, 228241,
              233017, 241001, 249841, 252601, 253241, 256999, 258511, 264773, 266305, 271951,
              272251, 275887, 276013,
              278545, 280601, 282133, 284581, 285541, 289941, 294271, 294409, 314821, 318361,
              323713, 332949, 334153,
              340561, 341497, 348161, 357761, 367081, 387731, 390937, 396271, 399001, 401401,
              410041, 422659, 423793,
              427233, 435671, 443719, 448921, 449065, 451905, 452051, 458989, 464185, 476971,
              481573, 486737, 488881,
              489997, 493697, 493885, 512461, 513629, 514447, 526593, 530881, 534061, 552721,
              556169, 563473, 574561,
              574861, 580337, 582289, 587861, 588745, 604117, 611701, 617093, 622909, 625921,
              635401, 642001, 647089,
              653333, 656601, 657901, 658801, 665281, 665333, 665401, 670033, 672487, 679729,
              680627, 683761, 688213,
              710533, 711361, 721801, 722201, 722261, 729061, 738541, 741751, 742813, 743665,
              745889, 748657, 757945,
              769567, 769757, 786961, 800605, 818201, 825265, 831405, 838201, 838861, 841681,
```

```
            847261, 852481, 852841,
            873181, 875161, 877099, 898705, 915981, 916327, 934021, 950797, 976873, 983401,
            997633]
    carmichael = [561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041, 46657, 52633,
                  62745, 63973,
                  75361, 101101, 115921, 126217, 162401, 172081, 188461, 252601, 278545, 294409,
                  314821, 334153, 340561,
                  399001, 410041, 449065, 488881, 512461, 530881, 552721, 656601, 658801, 670033,
                  748657, 825265,
                  838201, 852841, 997633]
    to_test = [x for x in sarrus if x not in carmichael]

    largestbase = 3
    culprits = []
    for n in to_test:
        a = 3
        while True:
            if fermat_test(a, n):
                a += 1
            else:
                if a > largestbase:
                    largestbase = a
                    culprits = [n]
                elif a == largestbase:
                    culprits.append(n)
                break

    return largestbase, culprits


def q4i():
    pseudoprimes_base_two = []
    for n in range(3, 10 ** 6 + 1, 2):
        if euler_test(2, n) and not trial_division(n):
            pseudoprimes_base_two.append(n)

    def isAbsoluteEuler(n):
        composite = False
        for a in range(2, n):
            if gcd(a, n) == 1:
                if not euler_test(a, n):
                    return False
            else:
                composite = True
        return composite  # if it gets here it must have passed all the rounds of testing

    absolute_euler_pseudoprimes = []
    for n in pseudoprimes_base_two:
        if isAbsoluteEuler(n):
            absolute_euler_pseudoprimes.append(n)

    return absolute_euler_pseudoprimes, pseudoprimes_base_two


def q4ii():
    to_test = [561, 1105, 1729, 1905, 2047, 2465, 3277, 4033, 4681, 6601, 8321, 8481, 10585, 12801,
               15841, 16705, 18705,
               25761, 29341, 30121, 33153, 34945, 41041, 42799, 46657, 49141, 52633, 62745, 65281,
               74665, 75361, 80581,
               85489, 87249, 88357, 90751, 104653, 113201, 115921, 126217, 129921, 130561, 149281,
               158369, 162401,
               164737, 172081, 188057, 196093, 208465, 215265, 220729, 223345, 233017, 252601,
               253241, 256999, 266305,
               271951, 278545, 280601, 294409, 314821, 323713, 334153, 340561, 348161, 357761,
               390937, 399001, 410041,
               427233, 448921, 449065, 458989, 476971, 486737, 488881, 489997, 493697, 514447,
```

```
            526593, 530881, 552721,
            580337, 588745, 625921, 635401, 647089, 656601, 658801, 665281, 670033, 683761,
            711361, 721801, 741751,
            745889, 748657, 800605, 818201, 825265, 838201, 838861, 841681, 852481, 852841,
            873181, 875161, 877099,
            916327, 976873, 983401, 997633]
    print(len(to_test))

    largestbase = 3
    culprits = []
    for n in to_test:
        a = 3
        while True:
            if euler_test(a, n):
                a += 1
            else:
                if a > largestbase:
                    largestbase = a
                    culprits = [n]
                elif a == largestbase:
                    culprits.append(n)
                break

    return largestbase, culprits


def q5i():
    pseudoprimes_base_two = []
    for n in range(2, 10 ** 6 + 1):
        if strong_test(2, n) and not trial_division(n):
            pseudoprimes_base_two.append(n)

    print(pseudoprimes_base_two)

    def isAbsoluteStrong(n):
        composite = False
        for a in range(2, n):
            if gcd(a, n) == 1:
                if not strong_test(a, n):
                    return False
            else:
                composite = True
        return composite  # if it gets here it must have passed all the rounds of testing

    absolute_strong_pseudoprimes = []
    for n in pseudoprimes_base_two:
        if isAbsoluteStrong(n):
            absolute_strong_pseudoprimes.append(n)

    return absolute_strong_pseudoprimes, pseudoprimes_base_two


def q5ii():
    to_test = [2047, 3277, 4033, 4681, 8321, 15841, 29341, 42799, 49141, 52633, 65281, 74665, 80581,
            85489, 88357,
            90751, 104653, 130561, 196093, 220729, 233017, 252601, 253241, 256999, 271951,
            280601, 314821, 357761,
            390937, 458989, 476971, 486737, 489997, 514447, 580337, 635401, 647089, 741751,
            800605, 818201, 838861,
            873181, 877099, 916327, 976873, 983401]

    largestbase = 3
    culprits = []
    for n in to_test:
        a = 3
        while True:
```

```
            if strong_test(a, n):
                a += 1
            else:
                if a > largestbase:
                    largestbase = a
                    culprits = [n]
                elif a == largestbase:
                    culprits.append(n)
                break

    return largestbase, culprits


def q6():
    fermat_pseudoprimes = [0, 0, 0, 0, 0]
    euler_pseudoprimes = [0, 0, 0, 0, 0]
    strong_pseudoprimes = [0, 0, 0, 0, 0]
    primes = [0, 0, 0, 0, 0]
    k_values = [5, 6, 7, 8, 9]
    bases = [2, 3]  # options [2], [3], [2,3]
    for k_index in range(5):
        k = k_values[k_index]
        for n in range(10 ** k, 10 ** k + 10 ** 5 + 1):
            if n % 1000 == 0:
                print(n)
            if not trial_division(n):  # if not prime
                if all([fermat_test(base, n) for base in bases]):
                    fermat_pseudoprimes[k_index] += 1
                if n % 2 == 1 and all([euler_test(base, n) for base in bases]):
                    euler_pseudoprimes[k_index] += 1
                if all([strong_test(base, n) for base in bases]):
                    strong_pseudoprimes[k_index] += 1
            else:  # if prime
                primes[k_index] += 1
        print(euler_pseudoprimes)

    return fermat_pseudoprimes, euler_pseudoprimes, strong_pseudoprimes, primes

    # ([28, 16, 6, 1, 0], [11, 8, 4, 0, 0], [3, 2, 0, 0, 0], [8392, 7216, 6241, 5411, 4832])


def q7():
    def rand():
        return random.randrange(3, 10 ** 10 + 1)

    def small_trial_division(n):
        for i in [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
                  79, 83, 89, 97, 101,
                  103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181,
                  191, 193, 197, 199,
                  211, 223, 227, 229, 233, 239, 241]:
            if n % i == 0:
                return False
        return True

    def my_test(n):
        if not small_trial_division(n):
            return False
        if n == 3215031751:
            return False
        for a in [2, 3, 5, 7]:
            if not strong_test(a, n):
                return False
        return True

    numbers = []
```

```
    prime1 = []
    prime2 = []

    for i in range(10000):
        numbers.append(rand())

    t0 = time.perf_counter()

    for n in numbers:
        if trial_division(n):
            prime1.append(True)
        else:
            prime1.append(False)

    t1 = time.perf_counter()

    trial_div_time = t1 - t0

    t0 = time.perf_counter()

    for n in numbers:
        if my_test(n):
            prime2.append(True)
        else:
            prime2.append(False)

    t1 = time.perf_counter()

    my_time = t1 - t0

    same = (prime1 == prime2)
    primes = prime1.count(True)
    return trial_div_time, my_time, same, primes


def q8():
    k = 15
    trials = 10 ** 6
    maxsuccesses = 10
    primes = 0

    def randodd():
        return random.randrange(2 ** (k - 1) + 1, 2 ** k, 2)

    def rand():
        return random.randrange(2 ** (k - 1), 2 ** k)

    results = []
    for i in range(maxsuccesses):
        results.append([0,
                        0])  # we'll store results in these lists, the first being the number of composites and the

    # print(results)
    for trial in range(trials):
        N = randodd()

        # for N in range(2 ** (k - 1) + 1, 2 ** k, 2):
        # if trial % 10 ** 5 == 0:
        # print(trial)
        if trial_division(N):  # don't care about primes
            primes += 1
            for success in range(maxsuccesses):
                results[success][1] += 1  # will pass the strong test for any base
            continue
        successes = 0  # how many rounds of the strong test N has passed
        while successes < maxsuccesses:
```

```
        while True:
            a = rand()
            if gcd(a, N) == 1:
                break
        if strong_test(a, N):
            successes += 1
            results[successes - 1][0] += 1
            results[successes - 1][1] += 1
        else:
            break

props = []
for i in results:
    props.append(i[0] / i[1])
return results, props, primes
```