Applied Modern Algebra (Spring 2020)
Homework 11: Primes and factoring (Due: Wed Apr 22 midnight)

Instructions: This homework has 12 problems, 2 of which are solely Python problems, and some others involve you using Python. You are allowed to use any of your code (or my helper code) from previous labs, but no other external code or code from pre-written Python modules (e.g., the `math` module). For #2 and #3, just submit the functions in a `.py` file attachment. For other problems which involve Python, you should handwrite or type your answers, and you do not need to include output or snippets of your Python session, but you are welcome to if it helps you explain your answer.

Note: Some of the problems are harder than others, and many of them involve previous problems. If you can't figure out some of the problems, that's okay. You can still move on and try to do later problems assuming the results of the previous problems if you need to.

1. Let $n > 1$ and $a \in (\mathbb{Z}/n\mathbb{Z})^\times$. Show that $a^{n-1} \equiv 1 \mod n$, i.e., n is a Fermat pseudo-prime (possibly prime) for base a, if and only if the order of a divides $n-1$.

2. Write a function in python `order(a,n)` to compute the order of a mod n. Return an error or -1 if a is not coprime to n. (You do not have to make this too efficient—e.g., you can compute the order of an element by brute force, but I suggest you use your earlier gcd function to test if a is coprime to n.)

3. Write a python function `avgorder(n)` that computes the average order of an element in $(\mathbb{Z}/n\mathbb{Z})^\times$.

4. Using your above code on a wide range of n, what can you say about how `avgorder(n)` tends to grow with n? Does there appear to be a difference if you restrict to n which is prime or a product of 2 primes, say?

5. Both the security of RSA and the practicality of Fermat's pseudo-prime test rely on the notion that most a's coprime to n have large order. Explain why. Based on what you found in the previous problem, does this suggest anything about when RSA and Fermat's pseudo-prime test work well?

6. Let $n = pq$ where p, q are distinct primes. Let $a \in \mathbb{Z}$. Prove that $a \equiv 1 \mod n$ if and only if $a \equiv 1 \mod p$ and $a \equiv 1 \mod q$. This is a special case of the *Chinese Remainder Theorem.*[1]

7. Let $n = pq$, with p, q distinct primes. Show that $a^{n-1} \equiv 1 \mod n$ for all $a \in (\mathbb{Z}/n\mathbb{Z})^\times$ if and only if $(p-1)|(n-1)$ and $(q-1)|(n-1)$. In other words, n is a Fermat pseudo-prime for all bases a if and only if $(p-1)|(n-1)$ and $(q-1)|(n-1)$. In this case, we call n a *Carmichael number* (see Section 11.4 of the text).

[1] I think this term is only mildly racist. It probably should be called something like *the remainder theorem* or *Sun Tzu's remainder theorem* (no, not that Sun Tzu). After all, you wouldn't call the Pythagorean theorem the "Greek right triangle theorem" (which you shouldn't anyway, because it was discovered at least by the Babylonians and possibly independently by Indians long before Pythagoras). Still, the Chinese also call it the Chinese remainder theorem.

8. Continuing with the previous exercise, show that $n = pq$ (with p, q distinct) is never a Carmichael number.

   Remark: The takeaway from these exercises is that for a non-prime n to be a Fermat pseudo-prime for many a, one essentially needs certain conditions about how the factors of $n-1$ relate to the factors of $p-1$ for primes $p|n$, and it's only in rare instances that all of these conditions align. (There's also no clean way to say when the Fermat pseudo-prime test will give you the wrong answer.) This is why the Fermat pseudo-prime test for a random base a rules out non-primes with very high probability (and it works even better if you also check n has no small prime factors, and/or you check several bases a).

9. Use Fermat's factorization method as presented in Section 11.3 to completely factor $n = 7538164$. You may use your python and any of your computer lab code. You do not need to include all of your input/output for this, but you should clearly explain your process, and include key details.

10. Use the Pollard rho factorization method to completely factor $n = 76663826$. (Same instructions as previous problem.)

11. The tortoise and the hare algorithm presented in the notes on Pollard rho, together with repeated squaring, can also be used to help compute the order of an element a mod n, without knowing anything about the factorization of n. Explain how this might work. Do you think this should give you a polynomial time algorithm to compute the order of a mod n? Explain.

12. Assume you can quickly compute the order of a mod n for any a, n which are coprime. Will this help you crack and RSA message or solve the discrete log problem? (Possibly under certain assumptions on n) Explain.