This assignment has several questions that warrant proofs. Try to emulate the proof style that we have used in class and the proof style that appears in the text. That is, make sure your reasoning flows logically from one statement to another. You should edit your proofs to make sure they read well. Abstraction is crucial. That is, identify and isolate common ideas. Write clearly and concisely or use LaTeX to typeset your solutions.

On homework assignments throughout the semester, hints will be provided in QR codes in the margin, numbered by their corresponding question. Some questions may have multiple hints; generally the hints are in order of hintiness. If you do not have a device capable of reading QR codes, you can find the hints in the `.tex` source for this document, linked from the course webpage.

Start early!

Please do come ask for help when you get stuck!

Have fun!

The Euclidean Algorithm, Again

In class we considered the problem of finding the *greatest common divisor* of two positive integers. We explored two ways to compute the GCD:

Method 1 Factor the two numbers into their unique prime factorizations, and find the biggest subset of primes contained in both; this is the factorization of the GCD.

Method 2 Run the Euclidean Algorithm.

In this first question, you will consider the difference between these methods.

Question 1.

(a) Use your brain, a calculator, Wolfram Alpha[1], and/or some other appropriate computational system to factor 170520 into its prime factors.

(b) Now factor 522522, and use the results to find $\gcd(170520, 522522)$ by Method 1.

(c) Trace the execution of the Euclidean Algorithm on $(170520, 522522)$. Compare and contrast the two methods of computing the GCD of these two numbers.

(d) Now try to factor the number m shown below, using whatever methods you like (for example, try using Wolfram Alpha). What happens?

[1] `http://www.wolframalpha.com/`; try typing `factor 170520`. You may find Wolfram Alpha to be a helpful resource for this class; just remember to cite it when you use its results.

$$m = 308903627938612635213051732991863520976852479109400884338832430641564115236537.$$

I would be willing to bet some money that you did not succeed in factoring m—though not a large amount, since there do exist algorithms that can factor numbers of this size in a matter of hours or even minutes[2]; but in any case factoring a number with, say, four times as many digits as this would require more like centuries. Factoring is widely believed to be a rather difficult computational problem.[3]

[2] On my computer, `https://www.alpertron.com.ar/ECM.HTM` was able to factor m in 12m13s using a self-initializing quadratic sieve (SIQS) algorithm.

[3] The security of your bank account probably depends on it!

(e) Now, suppose I tell you that there are in fact three prime numbers p, q, and r, such that

$$m = p \cdot q = 308903627938612635213051732991863520976852479109400884338832430641564115236537$$
$$n = p \cdot r = 235051426595377535232357646935740625119338466495681619002059053022377525089111$$

(this is the same number m from part (a)). Find p, q, and r.

1(e)

(f) What does this suggest about the relative efficiency of the two methods for computing the GCD?

Analyzing the Euclidean Algorithm

From now on, when referring to the Euclidean Algorithm, we will specifically work with the recursive implementation, GCDR, reproduced for convenience in Figure 1.

```
GCDR(a,b) =
  if b = 0
    then a
    else GCDR(b, a mod b)
```

Figure 1: The Euclidean Algorithm

In addition, when discussing $\gcd(a,b)$ from now on we will assume that $a > b$. This is not a real restriction, since $\gcd(a,b) = \gcd(b,a)$, so we can always switch the arguments if they are in the wrong order, and if the arguments are the same, $\gcd(a,a) = a$.

Recall that the *Fibonacci numbers* F_n are defined as follows:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_{n+2} = F_{n+1} + F_n$$

You might also occasionally see a definition with $F_0 = F_1 = 1$, but there are very good reasons for preferring the definition with $F_0 = 0$. For example, with this definition we have nice properties such as $m \mid n$ iff $F_m \mid F_n$, and, if we extend to negative Fibonacci numbers in the obvious way, $F_{-n} = (-1)^{n+1} F_n$.

That is, the first two Fibonacci numbers are $0, 1$, and then each subsequent Fibonacci number is the sum of the previous two, so the first few are $0, 1, 1, 2, 3, 5, 8, 13, \ldots$

Question 2.

(a) Compute F_9 and F_{10}, and trace the execution of the Euclidean Algorithm to compute $\gcd(F_{10}, F_9)$. What happens?

(b) Prove by induction on n that $\gcd(F_{n+1}, F_n) = 1$ for all $n \geq 0$.

(c) Explain why your proof also shows that the Euclidean Algorithm requires n recursive steps to compute $\gcd(F_{n+1}, F_n) = 1$.

In fact, it turns out that something much stronger is true: $\gcd(F_m, F_n) = F_{\gcd(m,n)}$! Proving this would be well outside the scope of this assignment, but you might be interested to explore it later.

Question 3. In fact, more is true: consecutive Fibonacci numbers (F_{n+1}, F_n) are in some sense a *worst-case* input for the Euclidean Algorithm: they are the *smallest* numbers for which the Euclidean Algorithm needs n steps.

3(a)

(a) Prove by induction on n: if $a > b$ and the Euclidean Algorithm requires n steps to compute $\gcd(a,b)$, then $a \geq F_{n+1}$ and $b \geq F_n$.

(b) Conclude that if $a \leq F_{n+1}$ then the Euclidean Algorithm requires at most n steps.

Question 4. How big are Fibonacci numbers? Let's find out:

(a) Solve $x^2 = x + 1$ for x and call the positive solution φ (this is often known as the *golden ratio*) and the negative solution $\hat{\varphi}$.

(b) Prove by induction on n that $F_n = \frac{1}{\sqrt{5}}(\varphi^n - \hat{\varphi}^n)$.

(c) Conclude that $F_n \approx \varphi^n / \sqrt{5}$.

Hints for (b): don't forget that you need two base cases; show that $\varphi - \hat{\varphi} = \sqrt{5}$, and remember that $\varphi^2 = \varphi + 1$ and similarly for $\hat{\varphi}$.

Hint for (c): what can you say about $\hat{\varphi}^n$ as n gets large?

Question 5. Suppose we want to run the Euclidean Algorithm to compute $\gcd(a,b)$, again assuming $a > b$. Let F_{n+1} be the smallest Fibonacci number which is greater than or equal to a. Then we know from question 3 that the Euclidean Algorithm takes at most n steps to run, with the worst case being when $a = F_{n+1}$. We want to figure out how n relates to a.

Starting from $a = F_{n+1}$ (since this is the *worst* case), and using the result from the previous question, solve (approximately) for n in terms of a. Your final answer should be of the form

$$n \approx k_1 \log_{10} a + k_2$$

for suitable constants k_1 and k_2; you should give k_1 and k_2 in approximate, decimal form. Conclude that the Euclidean Algorithm requires, *in the worst case*, a number of steps proportional to the number of digits in the base-ten representation of a.

One (not-so-secret) purpose of this question is to serve as a review of logarithms, which will feature prominently in this course. Please come ask if you need help remembering how to work with logarithms to solve this question!

This is (essentially) known as Lamé's Theorem, and was first proved by Gabriel Lamé in 1844. It was significant in being one of the first "practical" applications of the Fibonacci numbers, and one of the first results in what we would now call the theory of algorithms.

Question 6 (Optional Extra Credit). The jellybean game is a game for two players. There is a row of n jars, numbered 1 to n from left to right, each of which starts out containing some number of jellybeans (possibly zero). We assume that these are magical jars which can hold an *unlimited number* of jellybeans, so we don't have to worry about the jars getting too full. (Note, however, that the jars cannot hold *infinitely* many beans—each jar must always have some finite number of jellybeans in it, but that number can be as big as we want.) We also assume that there is an unlimited supply of extra jellybeans (such as a jellybean factory or a magical jellybean-pooping unicorn).

The players alternate turns. On a player's turn, she must:

1. Pick a nonempty jar, call it jar k.
2. *Remove* one jellybean from jar k.
3. *Add* as many jellybeans as she wants (possibly zero) to each of the jars $1 \ldots (k-1)$ to the left of her chosen jar.

For example, suppose there are five jars which currently hold

$$1 \quad 3 \quad 0 \quad 2 \quad 6.$$

One possible valid turn is to pick jar 4 and remove one jellybean from it, then add 6, 29, and one billion jellybeans to jars 1, 2, and 3 respectively, resulting in

$$7 \quad 32 \quad 10^9 \quad 1 \quad 6.$$

The winner is the player who removes the last jellybean. Put another way, the loser is the first player who is unable to move because all the jars are empty.

(a) Prove that the jellybean game always ends eventually. That is, even if the players conspire to try to make the game last forever, they cannot (as long as they follow the rules).

Of course, they certainly *can* make the game take a very, *very* long time…

(b) Describe a winning strategy for the jellybean game.

Question 7. On a scale of 1 to 10, with 1 being "my pet goldfish could do it in its sleep" and 10 being "who do you think I am, Einstein?", how difficult was this assignment? How many hours would you estimate that you spent on it?