JUNE 14TH COMPUTER LAB - FINDING GCDS, BEZOUT NUMBERS AND INVERSES

Number theorists are like lotus-eaters – having tasted this food they can never give it up. – Leopold Kronecker

We want to write our variant of the `GCD` function that returns the Bezout numbers (the s and t so that $sa + tb = 1$). I structured mine again as a recursive function. However, at each step, I didn't just have my function return the gcd. I had it return a list of three numbers.

`[g,s,t]`

where `g` is the the gcd and `s` and `t` are such that `sa + tb = g`. Thus before we can write our function, we have to think.

Say we are computing the gcd of a and b and we write $a = qb + r$. We then call `fancygcd` on b, r. `fancygcd` will return a list of integers $[g, s_1, t_1]$ where $s_1 b + t_1 r = g$. We plug in $a - qb$ for r and get that $s_1 b + t_1(a - qb) = g$. If we manipulate the left side, we obtain:

$$t_1 a + (s_1 - t_1 q)b = g$$

In other words, we should pick $s = t_1$ and $t = (s_1 - t_1 q)$ and then return $[g, s, t]$. Remember, to just take the integer part of division, you can always run `a // b` in Sage/Python. In other words `q = a//b` is what we are going to want to call.

My function looks like this (you'll need to fill in the ...):

```
def fancyGCD(a,b):
    if (a == b):
        ...
        ...
    if (b > a):
        r = b%a
        q = b//a
        ll = fancyGCD(a,r)
        g = ll[0]
        t = ll[2]
        s = ll[1]-ll[2]*q
        return [g,s,t]
```

Your job. Write your own function. Test it out on examples.

Finally, we want to make a function which finds the inverse of a modulo n, if it exists. To do this, we simply compute the `fancyGCD`. We should check to make sure the gcd is 1, and if it is, we now know that $sa + tn = 1$. So the inverse of a is ... :-)

Next... Write your own `inverseMod` function. In other words:

```
def inverseMod(a,n):
    ...
    return ...
```