COMPUTER EXPERIMENTATION #7 – MATH 5405 SPRING 2016

DUE: THURSDAY APRIL 14TH

We will implement adding (and doubling) points on an elliptic curve. This assignment is worth 10 points, but if you do this in an object oriented way (see the end of the assignment) you can get up to 5 bonus points.

We always assume our elliptic curve is written as

$$y^2 = x^3 + ax^2 + bx + c$$

(1) First let's create a function that checks whether a point `[x,y]` lies on the elliptic curve specified by the values `[a,b,c]`. By the way, I put all my elliptic curve functions in a file I created by running `gedit EC.py &`. Here's the start of my first function which does this check.

```
def isPtOnC(P, Elist, char):
    x = P[0]
    y = P[1]
    a = Elist[0]
    b = Elist[1]
    c = Elist[2]
```

If `char > 0`, then you should mod out by the characteristic. Otherwise, you should just check if $y^2 = x^3 + ax^2 + bx + c$. Note, in char 0, x and y should really be elements of the `Fractions` class. ie,

```
>>> from fractions import Fraction
>>> Fraction(1,2) + Fraction(1,3)
```

We do this because floating point decimal numbers will behave badly (rounding!).

(2) Next we need a function which doubles a point, ie computes $P + P = 2P$. You also need to decide how to return the point at infinity. One reasonable option is to write the point at infinity as `[float("inf"), float("inf")]`. This is what I decided to do. I also went back and adjust my point checking function to handle this point as well (manually, since modding out infinity by a number is badly behaved). You'll need your function which computes the inverse of a mod n as well (you can copy and paste it to the new file, or just import an old file).

I started my function like this

```
def doublePt(P, Elist, char):
    x = P[0]
    y = P[1]
    a = Elist[0]
    b = Elist[1]
    c = Elist[2]
    inf = float("inf")
    if (y == inf): #first handle the case if we are doubling the point at infinity
        return [float("inf"), float("inf")]
    if (y%char == 0): #next handle the case if our tangle line is vertical
```

```
        return [float("inf"), float("inf")]
    #now do the real work, remember
    #A = (3x^2 + 2ax + b)/(2y), so we need to invert 2y
    if (char > 0):  #first do the positive characteristic case
        twoyinv = invMod(2*y, char)
        ...
    else:
        ...
```

Make sure to *test* your function on some easy examples that you already know (from the book).

(3) Next we need to create a function that adds two arbitrary points on the elliptic curve. This should handle adding your point at infinity to other points.

```
def addPts(P, Q, Elist, char):
    def addPts(P, Q, Elist, char):
    xP = P[0]
    yP = P[1]
    xQ = Q[0]
    yQ = Q[1]
    a = Elist[0]
    b = Elist[1]
    c = Elist[2]
    inf = float("inf")
    #first we handle all the easy cases
    if (yP == inf):
        return Q
    if (yQ == inf):
        return P
    if (xP == xQ):
        if (yP == yQ):#if we are doubling a point...
            return doublePoint(P, Elist, char)
        else: #if we are not doubling a point, but the x-coords
              #agree, then we are going to add to infinity.
            return [float("inf"), float("inf")]
    #next do the real work.
    if (char > 0):  #do the positive characteristic case
        ...
    else:
        ...
```

Make sure to *test* this function also on some easy examples that you already know (from the book).

(4) Create a function that computes nP as efficiently as you can make it. In particular, do a bunch of doubling, then add things together appropriately.

(5) Write a naive discrete log function for elliptic curves. In other words, given $Q = nP$, write a loop that finds the value n.

Extra Credit: Make a class that is points on elliptic curves. Presumably this class should know the elliptic curve it is on (so it should know a, b, c, and the characteristic p). You could also make an elliptic curve class if you want. You should overload the operators `+,*` so that writing things like `P + Q` and `n * Q` in python will actually return the correct points.