Number Theory Assignment #2

21011712 박준영

1) gcd(a,b)

```
def gcd(a, b):
    while(a != 0):
        b, a = a, b % a

    return b
```

the source code

```
==========
>>> gcd(45, 75)
15
>>> gcd(666, 1414)
2
>>> gcd(102, 222)
6
>>> gcd(2**101+16, 2**202+8)
24
```

results of given examples

I've first defined the function **gcd(a, b)**, having two integers *a* and *b* as its factors. The gcd function is based on the Euclidean algorithm. In the while loop continues to put the value of *a* in variable *b*, and the value of *b % a* (the remainder of *b* divided by *a*) in variable *a*, which shows the implementation of the Euclidean algorithm. When the value of *a* becomes 0 the while loop ends, returning the former remainder *b*.

The picture below shows the results of the given examples, and it seems the code is working properly.

2) extended_gcd(a, b)

```python
def extended_gcd(a, b):
    if a == 0 :
        return b,0,1

    gcd,s1,t1 = extended_gcd(b%a, a)

    s = t1 - (b//a) * s1
    t = s1

    return gcd,s,t
```

the source code

```
>>> extended_gcd(45, 75)
(15, 2, -1)
>>> extended_gcd(666, 1414)
(2, -138, 65)
>>> extended_gcd(102, 222)
(6, -13, 6)
>>> extended_gcd(2**101+16, 2**202+8)
(24, 121737730625681081480451673661978806142549639637818238455189, -480170681904632349051781517l9)
>>>
```

results of given examples

The idea behind this function is to make it a recursive function. Unlike **gcd(a, b)**, we need to define what to do when *a* becomes 0. This is mainly because unlike the gcd function that had a condition in its while loop, this particular function is recursive, meaning that it does not automatically shut down when the *a* becomes 0. So I inserted an if function that returns the known value when the *a* is 0.

The recursion updates all three variables; *gcd*, *s1*, and *t1* which are used to update the result of this function. Let's see why both variables *s* and *t* can be defined like above.

s and *t* are the results of this function, meaning that gcd can be defined as

- s*a + t*b'. -

Another set of results are *s1* and *t1*, meaning that gcd can be defined as

- s1*(b % a) + t1*a -

If we put define *(b % a)* as *(b – [b / a] * a)* and put it in the above,

- gcd = s1 * (b – [b / a] * a) + t1 * a -

If we write the equation differently it becomes

- gcd = b * s1 + a * (t1 – [b / a] * s1) -

If we compare this equation with what we had previously,

s = t1 – [b / a] * s1

t = s1

Thus we can safely presume and define *s* and *b* as we see in the source code.

The code then returns the gcd value, value of *s*, and value of *t*.

The picture below shows the results of the given examples, and it seems the code is working properly.