CS-136L-03
Lab 5 - RSA keys
Lucina de la Rosa & Richard McCormick
DLD297 || RLM443
October 18th, 2019

1. **Problem Statement**

   For this lab, our main challenge was to write a program that could successfully generate an RSA key pair (public and private), encrypt a given key, as well as decrypt a given RSA key. We will implement the BigInteger class to perform the mathematical operations necessary. The final result should pass the tests in the RSA tester class.
   Some important things we needed include:
   - Use of BigInteger
   - Generation of RSA keys
   - Methods to encrypt and decrypt

2. **Planning**

   To begin our planning, we researched RSA encryption on both Wikipedia and the Internet. This allows us to get a general idea of what our end goal is going to look like, as well as view other peoples' implementations to see what a working example might look like. Our lab TA's also gave us an electronic handout that contained the necessary variables and formulas to implement our class.

3. **Implementation and Testing**

   The first part of our code is the most substantive and the most intensive mathematically. In order to generate the variables needed for secure encryption, we needed to use random numbers generated within certain parameters. This was easy enough to do with the java .random utility. However, to perform some of the operations required (such as inverse modulus, finding the totient, etc.) we needed to implement the BigInteger class for most of our mathematical variables. This increased the complexity of the problem, as now all operations needed to be carried out as method calls, instead of with operative statements such as with primitive data types and data type wrappers.

   Once each required variable was generated via the GenerateKeys method, and the keys were created, we moved on to the Encrypt and Decrypt methods. Because all the variables were already created and assigned, these methods were composed of only a few lines for checking the null conditions, and performing the single mathematical operation.

```
The encrypted text: 190691395589339766796088645874328637252919216108019482346798576705503535390611780450529938278567299138563199877504178116067984254582545324884419830766088405026 8
The decrypted text: 136
Test #1 passed!

The encrypted text: 175621325966876610754326935088828166139417019175332006904891315262666345403473896108572604782516093009268418531327456493439992052930148837757621159706277532124
The decrypted text: Programming is fun :)
Test #2 passed!
```

4. **Reflection**

   One of the largest problems we encountered in this lab was using the BigInteger class for our algorithms. Because each part of the variables had to be calculated with method calls, these statements often became quite elongated and difficult to read. However, while this did cause some difficulties with readability we made do by using extensive parentheses and double-checking our nested statements. In the future, we likely would use temporary variables in order to make things less difficult.