Evan Heaton
CS315
Homework 3

All of the test results are in bold on the last page!

Note: To input a vector from a file, type bits in reverse order separated by spaces and delimited by a non-integer character. 256 characters will be ignored after the non-integer character, or until the next newline. For example,

0 1 0 1 1 0 0 0 1 1 1 1 # = x1
1 1 0 1 1 1 1 1 1 # = y1

1 0 1 0 1 1 0 0 0 1 1 1 1 0 1 0 1 0 1 1 1 0 1 # = x2
1 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 # = y2

1 1 1 1 1 0 0 0 1 1 1 1 0 1 # = N3
1 1 0 1 1 1 1 1 1 # = x3
0 1 0 1 1 0 0 0 1 1 1 1 # = y3

1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 # = N4
1 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 # = x4
1 0 1 0 1 1 0 0 0 1 1 1 1 0 1 0 1 0 1 1 1 0 1 # = y4

1. For this subtraction algorithm, I chose to take a two's complement approach. I negated and added one to the second argument in the subtraction call, and then simply called my addition algorithm on the first argument and the two's complemented second argument. Simple!
2. For multiplication, I implemented the iterative "repeated addition" function that was outlined in the notes. For division I also implemented the iterative version from the notes. However because it is impossible to return multiple values in c++ I sent two vectors which represented the quotient and remainder by reference.
3. Modular Exponentiation was perhaps the most difficult algorithm to implement. One bug that I ran into that was hard to solve was the fact that I had forgotten to reset my quotient and remainder before each division, which led to absurd results that were very obviously incorrect (like answers that were larger than N). Once that was solved, the debugging process went relatively without problems.
4. Multiplication is orders faster than modular exponentiation, and that makes sense because modular exponentiation uses both multiplication and division y times. It seems like the times for multiplication are about tripling for each double of the input size, while the modular exponentiation times are being multiplied by a factor of about 8 each time the input length doubles.

x1 = [0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1]
y1 = [1, 1, 0, 1, 1, 1, 1, 1, 1]
x2 = [1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1]
y2 = [1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1]
x1 - y1 = [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1]
x2 - y2 = [0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1]

x1 * y1 = [0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1]
x2 * y2 = [1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1]
x1 / y1 = [1, 1, 1]
with remainder: [1, 0, 1, 1, 1, 1, 0, 0, 1]
x2 / y2 = [0, 0, 1, 1, 1, 0, 1]
with remainder: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1]

N3 = [1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1]
x3 = [1, 1, 0, 1, 1, 1, 1, 1, 1]
y3 = [0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1]
N4 = [0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1]
x4 = [0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1]
y4 = [0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1]
x3^y3(modN3) = [1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1]
x4^y4(modN4) = [1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1]
Time elapsed (multiplication): 0 seconds for n = 100
Time elapsed (modular exponentiation): 4 seconds for n = 100
Time elapsed (multiplication): 0 seconds for n = 200
Time elapsed (modular exponentiation): 30 seconds for n = 200
Time elapsed (multiplication): 0 seconds for n = 400
Time elapsed (modular exponentiation): 231 seconds for n = 400
Time elapsed (multiplication): 0 seconds for n = 800
Time elapsed (modular exponentiation): >600 seconds for n = 800
Time elapsed (multiplication): 1 seconds for n = 1600
Time elapsed (modular exponentiation): >600 seconds for n = 1600
Time elapsed (multiplication): 4 seconds for n = 3200
Time elapsed (modular exponentiation): >600 seconds for n = 3200
Time elapsed (multiplication): 13 seconds for n = 6400
Time elapsed (modular exponentiation): >600 seconds for n = 6400
Time elapsed (multiplication): 52 seconds for n = 12800
Time elapsed (modular exponentiation): >600 seconds for n = 12800
Program ended with exit code: 0