Homework 7: Problem 2

Jake Levin (jklevin)

Lachlan Kermode

1.

Our first algorithm to compute $a^x b^y (mod N)$, using at most $4logN+C$ multiplications is as follows. To solve for $a^x b^y (mod N)$, we will first break this term up into the following product:

$$(a^x (mod N) \cdot b^y (mod N))(mod N)$$

This we can do by the multiplicative laws of modular arithmetic that we saw in class, precept, and the textbook. Next, we will consider the individual terms $a^x (mod N)$ and $b^y (mod N)$. In order to solve for either of these modular exponents, we can use the following algorithm:

First, take $a^1 (mod N)$ and solve for that value. Let's call it c_0.
Now, we will look at $a^2 (mod N)$. Now we know $a^2 (mod N)$ is equivalent to $a^1 \cdot a^1 (mod N)$ (again this holds due to the laws of modular arithmetic regarding multiplication that we have previously discussed). Thus, we can solve for a^2 by solving for $c^2 (mod N)$. Let's call that value c_1.
Now, let's look at $a^4 (mod N)$, and apply the same process we used in the previous step to derive c_2. We can continue using this process $logx$ times until we have reached the greatest exponent of 2 that still divides our original exponent x, all the while keeping track of our values of c.
To finally solve for $a^x (mod N)$, we must simply break x down into it's binary representation, and look at the exponents of 2 that when summed together equal x. Since we have already solved for the value of a raised by all of these exponents modulo N, we can multiply together these corresponding values of c to find $a^x (mod N)$.

A brief example of this algorithm (in case it was confusing as written strictly in general terms), given $a = 5$, $x = 20$, $N = 7$ is as follows:

$$5^1 (mod 7) = 5$$

$$5^2 (mod 7) = 5^2 (mod 7) = 4$$

$$5^4 (mod 7) = 4^2 (mod 7) = 2$$

$$5^8 (mod 7) = 2^2 (mod 7) = 4$$

$$5^{16} (mod 7) = 4^2 (mod 7) = 2$$

Since 20 is equal to $16 + 4$, we can say that:

$$5^{20} (mod 7) = 5^{16} \cdot 5^4 (mod 7) = 2 \cdot 2 (mod 7) = 4$$

Since this algorithm uses at most $2logx + C$ total multiplications to finish (up to 2 multiplications per step over log N steps, and where C accounts for the final multiplicative steps), and $x < N$, this algorithm can run in at most $2logN + C$ time. Since we must use this algorithm to solve for both $a^x(modN)$, and then $b^y(modN)$, and then find the product of these two values modulo N to finally solve for $a^x b^y(modN)$, we know that this algorithm will take at most $4k+C$ multiplications, where $k = logN$ and C is a positive constant.

2.
If we consider our algorithm described above, we can rewrite this whole process much more succinctly as a recursive function in terms of $f(a, x)$ (or $f(b, y)$ – the point is its an algorithm to solve for a single exponential term). The key idea is that we are breaking down our exponent by using squaring. With this in mind, for the following problem we will use the well known recursive algorithm for solving for exponential values by squaring, which looks as follows:

$f(a, x) = f(a, \frac{x}{2})^2$ when x is even
$f(a, x) = a \cdot f(a, \frac{x-1}{2})^2$ when x is odd

Now using this algorithm on both terms a^x, b^y separately (as we did in part 1) takes 4logN + C multiplications because each term requires 2logN + C multiplications, and thus we were required to run this algorithm twice. Therefore, what we will aim to do here is to adjust our recursive algorithm so that it is in terms of $f(a, b, x, y)$, while retaining the condition that each step does not require more than 2 multiplications. To do so, we will derive a similar recursive algorithm, but one that takes into consideration all 4 conditions, where x,y are even, x,y are odd, x even y odd, y odd x even. We will write our algorithm as follows, where $f(a, b, x, y) =$:

$f(a, b, \frac{x}{2}, \frac{y}{2})^2$ where x and y are both even
$b \cdot f(a, b, \frac{x}{2}, \frac{y-1}{2})^2$ where x is even and y is odd
$a \cdot f(a, b, \frac{x-1}{2}, \frac{y}{2})^2$ where x is odd and y is even
$ab \cdot f(a, b, \frac{x-1}{2}, \frac{y-1}{2})^2$ where x and y are both odd

However, we have here a slight issue, as the case where x and y are both odd has us doing 3 multiplications. This can easily be solved however by recognizing that a and b are constants, and therefore we can pre compute our value ab, and call it c. We will then replace our fourth recursive condition we derived with:

$c \cdot f(a, b, \frac{x-1}{2}, \frac{y-1}{2})^2$ where x and y are both odd

With this final condition, we have now successfully derived an algorithm that will compute $a^x b^y(modN)$ in $logN$ steps, using no more than 2 multiplications at each step. Thus, this algorithm uses at most $2k + C$ multiplications, where $C = logN$.