Topics in Algebra: Cryptography - Complexity

http://www.mat.univie.ac.at/~gagt/crypto2019

Martin Finn-Sell
martin.finn-sell@univie.ac.at

1 Exponentiation by squares

Let a be a n-bit number and x an k-bit number. We'll estimate how complex the operation

$$f(x, a) : a \mapsto a^x$$

is using basic mathematical operations.

We'll use the **Square-and-Multiply** method, which runs (in pseudocode) like:

```
exp(a: int, k: int):
   z = exp(n*n, k/2)
   return a*z if k is odd else z
```

Remark 1.

- This reduces the number of bits in the exponent by 1. We will apply it many times (until the base has no exponent - this is rewriting the power in its binary expansion as in the exercise class only so we can count easily)

- For every bit in the base we have have to perform at most one squaring and one multiplication. The squaring will be on a number that's in-bits large, if we are looking at the i^{th} least significant bit, and we are multiplying by a number that's at most $(k - i)$ times at large.

So if we assume we can do multiplication in $\Omega(n \log n)$ time, then the two remarks above mean that the square and multiply process belongs to the complexity class $\sum_{i=0}^{k} O(in \log in)$. We now estimate this as follows:

$$\sum_{i=0}^{k} O(in \log in) \leq O(\sum_{i=0}^{k} in(\log n + \log k)) = O(n(\log n + \log k)k^2)$$

Where the first estimate works as everything is using the same constants - however one should really do that estimate carefully and not lazily as I've done here.

This means in the special case that $k \leq n$, we get that the exponentiation process takes $O(n^3 \log n)$.

It's also worth pointing out that the current best asymptotic multiplication algorithm has complexity worse than we used here - this is known as Fürer's Algorithm.