Exercise 1

Some more mathematical background

a) Compute $7^{-1} \bmod 11$ by hand.

b) Given two numbers a, b of n bit each. What is the worst case number of arithmetic operations of the (Extended) Euclidean Algorithm? Which input yields this worst case?

c) Why do we need a, n coprime to compute the modular inverse $a^{-1} \bmod n$? E.g. what happens for $\gcd(15, 39)$?

Exercise 2

Given a set of modular equations

$$a_i \equiv x \mod n_i \qquad\qquad i = 1, \dots, k$$

the solution to the Chinese remainder theorem can be computed via

$$\begin{aligned} b_i &:= \prod_{j \neq i} n_j & i &= 1, \dots, k \\ b_i' &:= b_i^{-1} \bmod n_i & i &= 1, \dots, k \\ x &:= \sum_{i=1}^{k} a_i b_i b_i' \bmod \prod_j n_j \end{aligned}$$

a) Show that the above method is correct.

b) Implement both solutions for the Chinese Remainder Theorem.

Input: List of moduli $[n_1, \dots, n_k]$ and list of remainders $[a_1, \dots, a_k,]$; or list of pairs $[(a_1, n_1), \dots, (a_k, n_k)]$

Output: solution x, (and $\prod n_i$)

c) Compare their theoretical and practical running time.

d) We demanded coprime n_i.

- What happens if this is not the case?
- Why was this not mentioned in the lecture?

Exercise 3

Write a function to compute $\varphi(n)$

a) via the definition

b) via factorisation

Up to which size (roughly) can you compute this within a few seconds?
In IPython, you can get the time of a function call via

```
%time foo(bar)
```

or if you want to run multiple iterations

```
%timeit foo(bar)
```