Starred exercises are optional.

1. For the prime numbers $p = 195931$ and $q = 2106945901$ and $e = 65537$ do, using RSA as on slide 10 of week 9, the following:
   a) Compute an inverse d of e modulo $n = (p-1) \cdot (q-1)$.
   b) Give the RSA-codes resulting from (publicly) encrypting the messages "cat" and "mouse", and check that (privately) decrypting their codes yields the original messages again.

      Here strings of letters over the English alphabet are encoded as natural numbers by viewing them as base-26 numbers (with $a = 0$ and $z = 25$) with a leading 1. For instance, the string `"abc"` is viewed as the integer $((1 \cdot 26 + 0) \cdot 26 + 1) \cdot 26 + 2 = 17604$.
   c) How long can messages we want to encrypt be (roughly), with this encryption scheme (for these p, q)? What goes wrong for longer messages?
   d∗) Replace encryption and decryption in the Haskell program with more efficient versions based on the Chinese remainder theorem.

   For all items, you may use/supplement the Haskell program on the flip side.

2. Let $f(n) = n \log n$ and $g(n) = n^2 + 2n + 1$. Compute $\limsup_{n\to\infty} \frac{f(n)}{g(n)}$ and $\limsup_{n\to\infty} \frac{g(n)}{f(n)}$ and use this to determine whether $f \in O(g), \Omega(g), \Theta(g), o(g)$ and/or vice versa.
   (f and g can e.g. be thought of as complexities of merge sort and bubble sort respectively.)

3. Suppose $f, g \in O(h)$, for f, g, h functions from $\mathbb{N}$ to $[0, \infty)$. Determine for both of the following functions whether or not it is in $O(h)$. If so, prove it. If not, give a counterexample.
   a) $f + g$, i.e. the function that maps n to $f(n) + g(n)$;
   b) $f \cdot g$, i.e. the function that maps n to $f(n) \cdot g(n)$.

4∗) Give functions f and g such that neither $f \in O(g)$ nor $g \in O(f)$, and argue why this is the case.

5∗) Alice and Bob, far removed from each other, want to construct a number they both know but no one else knows, based on a prime number p and primitive root r of p already known to them. A *primitive* root r of p is such that $\{r^1 \bmod p, r^2 \bmod p, \ldots, r^{p-1} \bmod p\}$ comprises the numbers 1 to $p-1$ (up to order). For instance, 3 is a primitive root of 7 since $\{3, 2, 6, 4, 5, 1\}$ comprises all numbers 1–6, but since $\{2, 4, 1, 2, 4, 1\}$ does not, 2 is not a primitive root of 7.

   They go about as follows:
   a) Alice selects some number $0 < a < p$ and sends $r^a \bmod p$ to Bob;
   b) Bob selects some number $0 < b < p$ and sends $r^b \bmod p$ to Alice;

   Argue that the following are true for the number $n = r^{a \cdot b} \bmod p$:
   - Both Alice and Bob can compute n easily;
   - For people other than Alice and Bob it is hard to find the number n, even if they have eavesdropped on the two messages sent between Alice and Bob.

```
expmod :: Integer -> Integer -> Integer -> Integer
expmod a n m = if n > 1 then (high * low) `mod` m else low where
  high = ((expmod a (n `div` 2) m)^2) `mod` m
  low = if (n `mod` 2) == 1 then a else 1

bezout :: Integer -> Integer -> (Integer,Integer,Integer)
bezout a b = if a > b then swap (bezout b a) else aux a b 1 0 0 1 where
  swap (g,u,v) = (g,v,u)
  aux a b ua va ub vb = if m == 0 then (a,ua,va) else aux m a um vm ua va where
    dm = divMod b a
    d = fst dm
    m = snd dm
    um = ub - ua*d
    vm = vb - va*d

invmod :: Integer -> Integer -> Integer
invmod a n = let (g,u,_) = bezout a n in
  if g == 1 then u `mod` n else error "no inverse"

primep = 195931
primeq = 2106945901
numbere = 65537
publickey :: (Integer,Integer)
publickey = (0,0) -- to be determined as on slide 10 week 9
privatekey :: Integer
privatekey = 0 -- to be determined as on slide 10 week 9

encode :: String -> Integer
encode = foldl (\y x -> 26 * y + toInteger (x - fromEnum 'a')) 1 . map fromEnum

decode :: Integer -> String
decode x = if x<=1 then "" else let (d,m) = divMod x 26 in
  decode d ++ [toEnum (fromEnum 'a' + fromInteger m)]

encrypt :: String -> Integer
encrypt m = expmod (encode m) (fst publickey )(snd publickey)

decrypt :: Integer -> String
decrypt c = decode $ expmod c privatekey (snd publickey)
```