Homework

1. (from last two weeks) Prove that for all natural numbers n there exists a list $[n_1, \ldots, n_k]$ of natural numbers that constitutes a prime factorisation of n, i.e. such that each n_i is prime and such that $n = n_1 \cdot \ldots \cdot n_k$, and extract a program implementing this specification. (For simplicity you may assume that 0 and 1 are 'prime'.)

   You may proceed as follows:

   - First formalise the statement, partitioning it as above (in prime and factorisation).
   - Show that (it is decidable whether) a natural number n is prime or has a non-trivial divisor d, i.e. such that exists n' with $n = d \cdot n'$ and $1 < d, n' < n$.
   - Show that a divisor of a divisor of n is a divisor of n.
   - Show that if d is a non-trivial divisor of n, then $n/d < n$.

   and follow the pattern of the above development for division. (You may freely use that development, and facts about natural numbers in the library, but try to develop the proof yourself).

   Additional hints (to enable you to adapt your own partial solutions):

   - *Have a look at Patrik's partial solution of last week.*
   - *In my own development I have followed the steps suggested above. In particular, as discussed in the PS I have first constructed a 'greatest divisor of n below a bound $1 + b$'-function* `gdb`*:*[1]

```
Definition divisor n d : Prop :=
  exists e, e * d = n.
Definition bdivisor n b d :=
  divisor n d /\ d <= S b.
Definition gbd n b d :=
  bdivisor n b d /\ forall e, bdivisor n b e -> e <= d.
Definition gdb (n b:nat) : { d | gbd n b d }.
```

   and then shown that this function can be used to characterise primality of numbers larger than or equal to 2 :

```
Definition prime (n:nat) := forall d, divisor n d -> (d = 1 \/ d = n).
Lemma prime_gdb : forall n,
  prime (S (S n))  <-> proj1_sig (gdb (S (S n)) n) = 1.
```

   To prove the lemma i've made use of trichotomoy (that $n < m$ or $n = m$ or $n > m$), to split divisibility of two numbers into three cases (a number always divides itself, and never divides a smaller number, so the only interesting case, dealt with by `gdb`*, is when it's smaller.)*

   The `gdb` *function can then be used in the construction of the* `primefactorise` *function, which itself is defined by well-founded recursion using the factor(s) found by* `gdb`*:*

```
Definition listproduct := fold_right mult 1.
Definition factorisation l n := listproduct l = n.
Definition primelist := Forall prime.
Definition primelist0 l := (l = cons 0 nil) \/ primelist l.
Definition primefactorisation l n := primelist0 l /\ factorisation l n.
Definition primefactorise n : {l | primefactorisation l n}.
```

[1] The $1 + b$ serves to let the bound not go lower than 1.

For the second part, `primefactorise`*, some auxiliary lemma's allowing to express properties of appended lists, in terms of the lists being appended came in handy, e.g. that* $l1 +\!\!+ l2$ *is a list of prime numbers if and only if both* $l1$ *and* $l2$ *are.*[2] *In the proofs I've found apart from the tactics already discussed, the tactics below useful.*

(case_eq) Same as case, but keeping an explicit equality for the relevant case, which can subsequently (after intro) be used to 'update' (rewrite) assumptions with that case.

(subst) Unfold defining equalities among the assumptions. (Good companion to case_eq, since it may be used to do the mentioned update.)

(inversion) Useful to discard impossible cases, having assumptions that fail for structural/inductive reasons.

[2] I've also proven an auxiliary lemma showing that the only way to get the singleton list containing 0, is by factorising 0. But such a lemma would be superfluous for an a bit more permissive definition of `primelist0`.