Homework

1. 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).

   Partial solution of Patrik:

```
Require Import Utf8 Nat Bool Arith Coq.Arith.PeanoNat Peano_dec List
Program Coq.Program.Wf Ring Omega Coq.Numbers.NatInt.NZDiv.

Definition divides (n d : nat) : bool := if eq_nat_decide (n mod d) 0 then true else false.

Program Fixpoint factor_bigger_i (n i: nat) (v : i<>0)
  { measure (id (n - i)) } :
  { t : bool * (nat * nat) |
  ( snd(snd t) * fst(snd t) = n /\
    fst(t) = true ) \/
  ( fst(t) = false /\
    forallb (fun x => if divides n x then false else true) (seq i (n - i)) = true )
  } :=

  if le_lt_dec n i
  then (false, (0,0)) (* if n <= i then no factor found *)
  else
    if eq_nat_decide (n mod i) 0
    then (true, (n / i, i)) (* if i is factor then return else recurse*)
    else factor_bigger_i n (S i) v
  .

Next Obligation.
  right.
  split.
    trivial.

    assert (n-i = 0) as len_eq_zero. omega.
    rewrite len_eq_zero.
    unfold seq, forallb; trivial.
Qed.
```

```
Next Obligation.
  apply (eq_nat_eq _ _) in H0.
  left.
  split.
    assert (i * (n / i) = i * (n / i) + 0) as H1. omega.
    rewrite H1.
    rewrite <- H0.
    rewrite <- Nat.div_mod.
    trivial.

    exact v.

    trivial.
Qed.

Next Obligation.
  unfold id.
  omega.
Qed.

Next Obligation.
  destruct_call factor_bigger_i. simpl in *.
  case o.
    intros ol.
    left.
    exact ol.

    intros or.
    right.
    split.
      exact (proj1 or).

      pose (proj2 or) as IH.
      assert (S (n- S i) = n-i ) as H1. omega.
      rewrite <- H1.
      simpl.
      apply andb_true_iff, conj.
        unfold divides.
        case eq_nat_decide.
          intros H0_contradiction.
          contradict H0_contradiction.
            exact H0.

          intros H0_duplicate.
          trivial.

        exact IH.
Qed.

Lemma twoNotOne: 2<>0.
Proof. omega. Qed.

Definition factor (n:nat)  := factor_bigger_i n 2 twoNotOne.

Eval compute in proj1_sig(factor 1071).
Eval compute in proj1_sig(factor 357).
Eval compute in proj1_sig(factor 119).
Eval compute in proj1_sig(factor 197).
```

```
Definition is_prime (n:nat) := negb(fst(proj1_sig(factor n))).

Eval compute in is_prime 197.

(* TODO: Factorization *)
```

- Bonus (may be not so easy): Prove that the prime factorisation is unique up to the order of the elements (and repetitions of 0,1).