Exercise 1 (Typeclasses)**.**

Recall the typeclass of decidable total orders:

```
Class DecTotalOrder (A : Type) := {
  leb : A -> A -> bool;
  leb_total : forall x y, leb x y \/ leb y x;
  leb_antisym : forall x y, leb x y -> leb y x -> x = y;
  leb_trans : forall x y z, leb x y -> leb y z -> leb x z }.
```

1. Define:

   ```
   leb_total_dec {A} {dto : DecTotalOrder A} : forall x y, {leb x y} + {leb y x}
   ```

   Hint. Using the `sdestruct` tactic from CoqHammer is often the easiest way to reason by cases on boolean expressions.

2. Prove that any type in the `DecTotalOrder` typeclass has decidable equality.

3. Define a function

   ```
   lexb {A} {dto : DecTotalOrder A} (l1 l2 : list A) : bool
   ```

   which implements the lifting of the order on `A` to the lexicographic order on `list A`.

*4. Use `lexb` to define an instance of `DecTotalOrder (list A)` for any `A` in `DecTotalOrder`, i.e., finish the following definition:

   ```
   Instance dto_list {A} {dto_a : DecTotalOrder A} : DecTotalOrder (list A).
   ```

   Hint. Look up the `functional induction` tactic in Coq's reference manual.

Exercise 2 (Induction principles)**.**

1. Prove `forall x y : nat, x + y = y + x` using the induction principle `nat_ind` directly. Do not use `induction`, `lia`, `omega` or `auto with arith`.

2. Prove

   ```
   forall l1 l2 : list A, List.length (l1 ++ l2) =
                          List.length l1 + List.length l2
   ```

   using the induction principle `list_ind` directly. Do not use `induction`.

Exercise 3 (Ackermann's function)**.**

Implement the Ackermann's function using Coq's `nat_rec`.