Esta primera parte de la primera práctica consiste en realizar, como ejercicio, lo que queda pendiente de la ordenación por inserción para enteros que se ha visto en clase y también definir una función *sort* del siguiente tipo:

```
sort
     : forall A : Set,
       (forall leA eqA : relation A,
        (forall x y : A, {leA x y} + {leA y x})
        -> (forall eqA_dec : forall x y : A, {eqA x y} + {not (eqA x y)},
           (forall l : list A,
            {l' : list A | equiv eqA eqA_dec l l' /\ sorted leA l'})))
```

Se trata de implementar la ordenación por inserción en listas sobre un tipo A en el cual se tiene una relación de igualdad *eqA* decidible (*eqA_dec*); un orden total *leA* ($\forall\ x\ y{:}A\ \{leA\ x\ y\}+\{leA\ y\ x\}$) y una equivalencia de listas que nos dice cuando dos listas tienen los mismos elementos *equiv*.

Como indicación pueden inspirarse en la ordenación de listas de enteros que hemos visto en clase y en la librería *Sorting*. Se puede utilizar el siguiente ecabezamiento:

```
Require Export List.
Set Implicit Arguments.
Section defs.

 Variable A : Set.
 Variable leA : A→ A→Prop.
 Variable eqA : A→A→ Prop.

 Hypothesis leA_dec : ∀ x y:A, {leA x y} + {leA y x}.
 Hypothesis eqA_dec : ∀ x y:A, {eqA x y} + {~ eqA x y}.
 Hypothesis leA_refl : ∀ x y:A, eqA x y → leA x y.
 Hypothesis leA_trans : ∀ x y z:A, leA x y → leA y z → leA x z.
 Hypothesis leA_antisym : ∀ x y:A, leA x y → leA y x → eqA x y.

 Hint Resolve leA_refl.
 Hint Immediate eqA_dec leA_dec leA_antisym.

Inductive sorted : list A → Prop :=
  | sorted0 : sorted nil
  | sorted1 : ∀ z:A, sorted (z :: nil)
  | sorted2 :
      ∀ (z1 z2:A) (l:list A),
        leA z1 z2 →
        sorted (z2 :: l) → sorted (z1 :: z2 :: l).
```

Hint *Resolve sorted0 sorted1 sorted2 : sort.*