```
Theorem app_assoc4 : forall l1 l2 l3 l4 : natlist,
  l1 ++ (l2 ++ (l3 ++ l4)) = ((l1 ++ l2) ++ l3) ++ l4.
Proof.
 intros l1 l2 l3 l4. induction l1 as [| h t HL].
  -simpl. rewrite app_assoc. reflexivity.
  -simpl. rewrite HL. reflexivity.
Qed.

(** An exercise about your implementation of [nonzeros]: *)

Lemma nonzeros_app : forall l1 l2 : natlist,
  nonzeros (l1 ++ l2) = (nonzeros l1) ++ (nonzeros l2).
Proof.
 intros l1 l2. induction l1 as [| n HL].
  -simpl. reflexivity.
  - destruct n.
    + simpl. rewrite -> IHHL. reflexivity.
    + simpl. rewrite -> IHHL. reflexivity.
Qed.
```

```
Theorem remove_does_not_increase_count: forall (s : bag),
  (count 0 (remove_one 0 s)) <=? (count 0 s) = true.
Proof.
  intros s. induction s as [| n HS].
  -simpl. reflexivity.
  -destruct n.
    +simpl. rewrite leb_n_Sn. reflexivity.
    +simpl. rewrite IHHS. reflexivity.
Qed.
(** [] *)

(** **** Exercise: 3 stars, standard, optional (bag_count_sum)

    Write down an interesting theorem [bag_count_sum] about bags
    involving the functions [count] and [sum], and prove it using
    Coq.  (You may find that the difficulty of the proof depends on
    how you defined [count]!) *)
(* FILL IN HERE

    [] *)

(** **** Exercise: 4 stars, advanced (rev_injective)

    Prove that the [rev] function is injective. There is a hard way
    and an easy way to do this. *)

Search rev.

Theorem rev_injective : forall (l1 l2 : natlist),
    rev l1 = rev l2 -> l1 = l2.
Proof.
  intros. rewrite <- (rev_involutive l1). rewrite H. apply rev_involutive.
Qed.
```

```
Theorem update_eq :
  forall (d : partial_map) (x : id) (v: nat),
    find x (update d x v) = Some v.
Proof.
 intros. simpl. rewrite<-eqb_id_refl. reflexivity.
Qed.
```