



1. Consider the definition of `nat`:

```
Inductive nat : Set := 0 : nat | S : nat->nat.
```

- (a) What are the constructors of `nat`?
- (b) Describe the elements of `nat`.
- (c) Give the type of `nat_ind`.

2. Consider the following definition:

```
Inductive A : Set :=  
| a : A -> A  
| b : A -> A -> A.
```

How many elements does the set `A` have?

3. (a) Consider the definition of `natlist` for lists of natural numbers:

```
Inductive natlist : Set :=  
| nil : natlist  
| cons : nat -> natlist -> natlist.
```

Give the type of `natlist_ind`, which is used to give proofs by induction.

- (b) Give the definition of an inductive predicate `last_element` such that `(last_element n l)` means that `n` is the last element of `l`.

4. (a) Give the inductive definition of the datatype `natbintree` of binary trees with unlabeled nodes and natural numbers at the leaves.

- (b) The Coq function for appending two lists is defined as follows:

```
Fixpoint append (l k : natlist) {struct l} : natlist :=  
  match l with  
  | nil => k  
  | cons n l' => cons n (append l' k)  
  end.
```

In what argument is the recursion? Why is the recursive call (intuitively) safe?

(c) Give the definition of a recursive function `flatten : natbintree -> natlist` which flattens a tree into a list that contains the nodes from left to right.

You may use `append`.

(d) Give a recursive definition of a function `count` that takes as input a `natbintree` and gives as output the number of nodes of the tree.

5. Consider the definition of an inductive predicate for even:

```
Inductive even : nat -> Prop :=
| even_zero      : even 0
| even_greater   : forall n:nat, even n -> even (S (S n)).
```

(a) What is the type of `even 0`?

(b) Give an inhabitant of `even 0`.

(c) Give an inhabitant of `even 2`.

6. (about program extraction) What is the type of the function that can be extracted from the proof of the following theorem:

```
forall l : natlist,
{l' : natlist | Permutation l l' /\ Sorted l'}.
```