Exercise sheet 12

Exercise 1 (Binary search trees)**.**

Consider the following type of binary trees:

```
Inductive tree A := empty | node (x : A) (l r : tree A).
```

A binary tree of type `tree A` is a *binary search tree* (with respect to a fixed decidable total order) if for every internal node `node x l r` of the tree, `l` and `r` are binary search trees, every element in `l` is less or equal to `x`, and every element in `r` is strictly greater than `x`.

1. Define an inductive type `BST {A} {dto : DecTotalOrder A} : tree A -> Prop` such that `BST t` holds iff `t` is a binary search tree.
2. Define a function `elements {A} (t : tree A) : list A` which returns the list of elements in `t` in the in-order traversal order.

*3. Prove `forall t, BST t -> Sorted (elements t)`.

Exercise 2 (Dependent pattern matching)**.**

In this exercise we implement a function which accesses the i-th element of a vector. Dependent types ensure that it is not possible to use the function with an "out-of-bounds" index.

1. Define a function `vnth {A n} (v : vector A n) (i : nat) (p : i < n) : A` which returns the `i`-th element of `v`. Do not use the `Program` command.

   Hint. You need to generalise the matches by appropriate equalities. You will need both the `in` and `as` annotations.
2. Define the previous function using the `Program` command.

Be careful to define the functions in such a way that they work with Coq's computation mechanism.

***Exercise 3** (Termination proofs)**.**

Without using the `Function` command or the `Program` package, implement a function

```
merge : forall {A} {dto : DecTotalOrder A}, list A -> list A -> list A
```

which merges two sorted lists into a single sorted list (in linear time).