This week we are learning about decidability and automation in Idris programming.

A decision procedure for a predicate is an algorithm that for each index either produces a proof that the predicate holds or else a refutation proving that it does not. In Idris the type constructor for decidability is called Dec with constructors Yes and No. Additionally, there is an interface for types with decidable equality called DecEq in the standard library module Decidable.Equality.

An auto-implicit argument is one that is intended to be found by Idris’s term search mechanism. By default this consists of using constructors and recursion in order to find a term of a given type, but you may specify additional terms for it to try using the %hint directive. We typically use auto-implicit arguments as constraints to guarantee that some validity condition is satisfied.

Task 1
Write a function that returns the head element of a nonempty list:

```idris
list_head : (xs : List a) -> {auto nonempty : Not (xs = [])} -> a
```

Make sure that it is able to satisfy the nonempty constraint for list literals formed with the (::) constructor:

```idris
dl : list_head [1, 2, 3] 1
dl : list_head [1] 1
```

Error: Can't find an implementation for Void.

Task 2
Write a function that returns the list element at a valid index:

```idris
list_index : (i : Nat) -> (xs : List a) -> {auto inbounds : LTE (S i) (length xs)} -> a
```

Make sure that it is able to satisfy the inbounds constraint for appropriate Nat and List literals:

```idris
> list_index 0 [True, False] True
> list_index 1 [True, False] False
> list_index 2 [True, False] Error: Can't find an implementation for LTE 3 2.
```

Task 3
Write a decision procedure for the \(\leq\) relation on natural numbers:

```idris
decide_LTE : (m, n : Nat) -> Dec (LTE m n)
```
**Task 4**
Use the following definition of the “between” relation on natural numbers:

\[
\text{Between} : \mathbb{Nat} \rightarrow \mathbb{Nat} \rightarrow \mathbb{Nat} \rightarrow \text{Type} \\
\text{Between lower upper n} = \text{LTE lower n} \ ` \text{And} ` \text{LTE n upper}
\]

in order to write a decision procedure for betweenness:

\[
\text{decide_between} : (\text{lower}, \text{upper}, \text{n} : \mathbb{Nat}) \rightarrow \text{Dec (Between lower upper n)}
\]

**Task 5**
Recall the type of (node-labeled binary) trees:

\[
\text{data Tree} : \text{Type} \rightarrow \text{Type} \ where \\
\text{Leaf} : \text{Tree a} \\
\text{Node} : (l : \text{Tree a}) \rightarrow (x : a) \rightarrow (r : \text{Tree a}) \rightarrow \text{Tree a}
\]

In this task you will write a decision procedure for \text{Tree} equality by implementing the \text{DecEq} interface for \text{Trees} whose element types themselves have decidable equality:

\[
\text{implementation DecEq a => DecEq (Tree a) where}
\]

Here are some hints to help you:

- A \text{Tree} is like a \text{List} with two tails, so the decision procedure for \text{List} equality that we wrote in lecture, \text{decide_list_eq} will be a useful guide.
- Recall the \text{heads_differ} and \text{tails_differ} functions for \text{Lists} that you wrote in lab last week. You most likely implemented these using contrapositive. In this task you will need to do something analogous for \text{Trees}.
- When you case analyze a term \text{decEq x y}, remember that in the \text{Yes} branch, if you further case analyze the equality proof you will discover that the only constructor is \text{Refl}, and in this case the equality indices are unified.