Exercise 4.1 Informally, a multiset is a set that can count how often an element occurs in it. Formally, a multiset (over some type $T$) corresponds to a function\(^1\) from $T$ to $\text{nat}$, and we say that $x$ is an element of a multiset $S$ iff $S(x) > 0$. Below, you find a PVS theory \texttt{multiset_defs} which defines multisets and some operations on them analogous to the prelude theory \texttt{sets}. Go through the prelude theory \texttt{sets_lemmas} and generalize as many of the lemmata as possible to multisets.

\texttt{multiset_defs}[T: \text{TYPE}]: \text{THEORY}
\texttt{BEGIN}
  \text{multiset}: \text{TYPE} = [T \to \text{nat}]

  \text{x, y, z: VAR T}
  \text{s, t: VAR \text{multiset}}

  \text{count(x,s): \text{nat} = s(x)}
  \text{member(x,s): \text{bool} = s(x) > 0}

  \text{submultiset?(s,t): \text{bool} = \text{FORALL x: s(x) \leq t(x)}}

  \text{empty: \text{multiset} = \text{LAMBDA x: 0}}
  \text{union(s,t): \text{multiset} = \text{LAMBDA x: s(x) + t(x)}}
  \text{intersection(s,t): \text{multiset} = \text{LAMBDA x: min(s(x), t(x))}}
  \text{difference(s,t): \text{multiset} = \text{LAMBDA x: max(s(x) - t(x), 0)}}

  \text{singleton(x): \text{multiset} =}
    \text{\quad \text{LAMBDA y: IF x = y THEN 1 ELSE 0 ENDIF}}
  \text{add(x,s): \text{multiset} =}
    \text{\quad \text{LAMBDA y: IF x = y THEN s(y)+1 ELSE s(y) ENDIF}}
  \text{remove(x,s): \text{multiset} =}
    \text{\quad \text{LAMBDA y: IF x = y THEN max(s(y)-1, 0) ELSE s(y) ENDIF}}
\text{END \text{multiset_defs}}

Hints:
\begin{itemize}
  \item If a lemma generalizes to multisets then its proof is easy.
  \item Prove (and use) the following version of extensionality:
    \((\text{FORALL x: count(x,s) = count(x,t)}) \implies s = t)\)
\end{itemize}

\(^1\)Recall that a subset of some set $X$ corresponds to its \textit{characteristic function} from $X$ to \{0,1\}.
Exercise 4.2  Sorting lists.

1. Prove that the function `mergesort` below is well-defined, i.e., prove the TCCs from its recursive definition. See exercises 1.3 and 3.3 for definitions of the functions `split` and `merge`.

   ```plaintext
   mergesort(xs:list[t]): RECURSIVE list[t] =
   LET n = length(xs), (ys,zs) = split(xs, floor(n/2)) IN
      IF n <= 1
          THEN xs
          ELSE merge(mergesort(ys),mergesort(zs))
      ENDIF
   MEASURE length(xs)
   ```

   **Hint:** Instead of working with the recursive definition of `split` directly, you should probably derive its specification in terms of `length` first, i.e., prove the lemmata

   ```plaintext
   split_length1: LEMMA
      n <= length(xs) IMPLIES length(split(xs,n)`1) = n
   
   split_length2: LEMMA
      n <= length(xs) IMPLIES length(split(xs,n)`2) = length(xs) - n
   ```

2. Prove that `mergesort` applied to any list yields a sorted list, i.e., prove the theorem `mergesort_sorted` below. See exercise 3.3 for a definition of the predicate `sorted?`.

   ```plaintext
   mergesort_sorted: THEOREM sorted?(mergesort(xs))
   ```

   **Hints:**
   - Use `measure-induct`.
   - It might be useful to derive the specification of `split` in terms of `append`, i.e., prove the lemma
     
     ```plaintext
     split_append: LEMMA n <= length(xs) IMPLIES append(split(xs,n)) = xs
     ```