1 List Manipulation

1.0.1 Swap

Due to the differences in how the odd-length and even-length lists are handled the midpoint of the list has to be pulled out before the result can be constructed. there are many sub-definitions for clarity that should speak for themselves based on their names.

```
swap :: [] a -> [] a
swap ls = reverse $ take half result ++ mid ++ drop half result
  where (mid, elems) = if odd (length ls)
    then (take 1 rest, fst ++ (drop 1 rest))
    else ([], ls)
    fst = take half ls
    rest = drop half ls
    result = swaps elems
    swaps [] = []
    swaps [x] = [x]
    swaps (x : y : ls) = y : x : swaps ls
    half = length ls `div` 2
```

1.1 Sets

1.1.1 Set Union

I defined the union of two sets to be the result of appending the two lists together then deleting the duplicates of the list using *nub*. I composed *nub* and $++$ together using the double-compose operator $(\circ) \circ (\circ)$ to keep the function point-free.

```
import Data.List (nub)
setUnion :: Eq a => [a] -> [a] -> [a]
setUnion = ((.) . (.)) nub (++)
```

1.1.2 Set Intersection

To do set intersection, I define it as the list of all elements that come from the union of the two sets but belong to both set 1 and set 2.

```
setIntersection :: Eq a => [a] -> [a] -> [a]
setIntersection s1 s2 = [a | a <- setUnion s1 s2, (elem a s2) && (elem a s1)]
```

1.1.3 Set Difference

To define set difference, we need the concept of *xor*. I can define *xor* using Haskell's pattern matching:

```
xor True True = False
xor False False = False
xor _ _ = True
```

I now compute the set difference as

$$S1 \setminus S2$$

.

$setDifference :: Eq\ a \Rightarrow [\,]\ a \rightarrow [\,]\ a \rightarrow [\,]\ a$
$setDifference\ s1\ s2 = [x \mid x \leftarrow s1, \neg\ (elem\ x\ s2)]$

1.1.4 Set Equal

Set equal is defined as true if and only if all the elements in the union of the two sets are in both of the individual sets.

$setEqual :: Eq\ a \Rightarrow [\,]\ a \rightarrow [\,]\ a \rightarrow Bool$
$setEqual\ s1\ s2 = and\ [elem\ x\ s1 \wedge elem\ x\ s2 \mid x \leftarrow setUnion\ s1\ s2]$

1.1.5 Powerset

Traditional definition of a powerset:

$powerSet :: Eq\ a \Rightarrow [\,]\ a \rightarrow [\,]\ ([\,]\ a)$
$powerset\ [\,] = [[\,]]$
$powerSet\ (x : xs) = rest +\!\!+\ (map\ (x:)\ rest)$
  where $rest = powerset\ xs$