FP Lab 4

1. Without looking at the standard prelude, define the following library functions using recursion:

   - Concatenate a list of lists: `concat :: [[a]] -> [a]`
   - Produce a list with n identical elements: `replicate :: Int -> a -> [a]`
   - Decide if a value is an element of a list: `elem :: Eq a => a -> [a] -> Bool`

2. Define a recursive function `merge :: [Int] -> [Int] -> [Int]` that merges two sorted lists of integers to give a single sorted list. For example,

   ```
   > merge [2,5,6] [1,3,4]
   [1,2,3,4,5,6]
   ```

3. Define a recursive function `msort :: [Int] -> [Int]` that implements merge sort, which can be specified by the following two rules:

   (a) Lists of length less than or equal to 1 are already sorted;

   (b) Other lists can be sorted by sorting the two halves and merging the resulting lists.

   Hint: You can use the following function to split a list into a pair of lists whose length differ by at most one:

   ```
   halve :: [a] -> ([a], [a])
   halve xs = splitAt (length xs `div` 2) xs
   ```

4. Consider the following function:

   ```
   some_function :: (a -> b) -> (a -> Bool) -> [a] -> [b]
   some_function f p xs = [f x | x <- xs, p x]
   ```

   Define an equivalent function with the name `map_and_filter` that uses the functions map and filter rather than list-comprehension. (c.f. slides VIII for relevant information).

5. Redefine `map f` and `filter p` using `foldr` as explained in VIII.