Homework #3

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

   a. Decide if all logical values in a list are true:

```
and :: [Bool] -> Bool

and [] = False

and [n] = n

and(n:ns)

        | tempTail == True = True

        | otherwise = False

        where tempTail = and ns
```

   b. Concatenate a list of lists:

```
concat :: [a] -> [a] -> [a]

concat [] [] = []

concat ns [] = ns

concat [] ps = ps

concat (n:ns) (p:ps) = n: concat ns  (p:ps)
```

   c. Produce a list with n identical elements:

```
replicate :: Int -> a -> [a]

replicate n p = if n>0 then p: replicate (n-1) p else []
```

   d. Select the nth element of a list:

```
(!!) :: [a] -> Int -> a

(!!) (n:ns) p = if p>1 then (!!) ns (p-1) else n
```

e. Decide if a value is an element of a list:

```
elem :: Eq a => a -> [a] -> Bool
elem _ [] = False
elem n (p:ps)
        | n == p = True
        | otherwise = elem n ps
```

2. Define a recursive function

```
merge :: Ord a => [a] -> [a] -> [a]
```

That merges two sorted lists of values to give a single sorted list.

For example:

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

```
merge :: Ord a => [a] -> [a] -> [a]
merge [] ns = ns
merge ns [] = ns
merge (n:ns) (p:ps)
        | p < n = p: merge (n:ns) ps
        | otherwise = n: merge ns (p:ps)
```

3. Define a recursive function

```
msort :: Ord a => [a] -> [a]
```

that implements merge sort, which can be specified by the following two rules:

i. Lists of length <= 1 are already sorted;

ii. Other lists can be sorted by sorting the two halves and merging the resulting lists.

```
msort :: Ord a => [a] -> [a]
msort [] = []
msort ns
   | (length ns) > 1 = merge (msort ls) (msort rs)
   | otherwise = ns
   where (ls, rs) = splitinhalf ns

splitinhalf :: [a] -> ([a], [a])
splitinhalf ns = (take n ns, drop n ns)
   where n = length ns `div` 2

merge :: Ord a => [a] -> [a] -> [a]
merge [] ns = ns
merge ns [] = ns
merge (n:ns) (p:ps)
   | p < n = p: merge (n:ns) ps
   | otherwise = n: merge ns (p:ps)
```

We call msort on the list and it calls the rest.