Lab Session Software Testing 2013, Week 5 With each deliverable, indicate the time spent.

```
module Lab5

where
import Data.List
import Week5
```

1. The function `merge` from the course notes can be used as follows, to create a function for list sorting:

   ```
   mergeSrt ::  Ord a => [a] -> [a]
   mergeSrt [] = []
   mergeSrt (x:xs) = merge [x] (mergeSrt xs)
   ```

   Find a suitable assertion, and write an assertive version of this.

   Deliverables: Assertion, Haskell program that uses this assertion, indication of time spent.

2. Another approach to merge sort is to start by splitting the list to be sorted in equal parts, recursively sort the parts, next merge.

   Implement this, using the following split function.

   ```
   split :: [a] -> ([a],[a])
   split xs = let
      n = (length xs) `div` 2
     in
      (take n xs, drop n xs)
   ```

   Next, find a suitable assertion, and write an assertive version.

   Deliverables: Haskell program, assertion, assertive version of Haskell program that uses this assertion, indication of time spent.

3. The goal of this exercise is to extend the sudoku program from the course notes with functions that can also handle sudokus of a special kind: the sudokus that appear in NRC-Handelsblad each week (designed by Peter Ritmeester, from Oct 8, 2005 onward). These NRC sudokus are special in that they have to satisfy a few extra constraints: in addition to the usual sudoku constraints, each of the 3×3 subgrids with left-top corner (2,2), (2,6), (6,2), and (6,6) should also yield a surjective function.

   Here is an example (the sudoku exercise of Saturday Nov 26, 2005):

```
+---------+---------+---------+
|         | 3       |         |
|   +-----|--+   +--|-----+   |
|   |     | 7|   |  | 3   |   |
| 2 |     |  |   |  |     | 8 |
+---------+---------+---------+
|   |   6 |  |   |5 |     |   |
|   +-----|--+   +--|-----+   |
|    9  1 | 6       |         |
|   +-----|--+   +--|-----+   |
| 3 |     |  | 7 |1 | 2   |   |
+---------+---------+---------+
|   |     |  |   |  |    3| 1 |
|   |8    |  | 4 |  |     |   |
|   +-----|--+   +--|-----+   |
|       2 |         |         |
+---------+---------+---------+
```

   Your task is to formalize this extra constraint, and to use your formalization in a program that can solve this sudoku.

   Deliverables: formal statement of new constraint, modified Haskell program, sudoku solution for the above NRC-Handelsblad sudoku, indication of time spent.

4. The course notes of this week contain a sudoku solver. A sudoku generator written in Haskell is available on the course web page, as `RandomSudoku.hs`. Use your program from the previous exercise and this program to create a program that generates NRC-Handelsblad sudoku problems.

   Deliverables: NRC-Handelsblad sudoku generator, indication of time spent.

5. Test your programs from the previous two exercises, and document the test process. One important property to test is whether the generated sudoku problems are *minimal*. How can you test this?

   Deliverables: testing code, test report, indication of time spent.