CSE4102: Programming Languages **Spring 2017**

Homework 2: ML And Recursive Data

Out Date: 01/31/2017
Due Date: 02/07/2017

Objectives

Now that we have some basic container capabilities, you are to build several classic container data structures. Feel free to use the course web page to find additional information and tutorials if needed. Handin a single text file (`.sml` extension) with your answers. Do respect the name of the functions and calling style to ease grading. If you need helper (auxiliary) functions, feel free to add them. Do comment your code. Feel free to refer to or convenience functions to manipulate strings.

1 Question 1

1. Write an ML function `qsort` which, given a list of objects l and a comparator function f (given in curry style, f has type $'a \rightarrow' a \rightarrow bool$) outputs a sorted version of l in ascending order according to f using the classic quicksort algorithm. Note that we are well aware of the Wikipedia implementation or the other dozen implementations floating on the web. I expect you to write this on your own! Do not use any basis (library) functions and write every helper function from first principles. Namely, write

```
fun qsort l f = ...
```

2. Revisit the previous question, but this time, you cannot define the partition from first principles (as a recursive function) nor can you use the List.partition basis function. The only thing you can use is the higher-order function `foldr`.

```
fun qsHigh l f = ...
```

3. Write an ML function cross which, given two lists, computes the list of their cross-products. Namely, given the inputs `[1,2,3]` and `[4,5,6]`, it computes `[(1,4), (1,5), (1,6), (2,4), (2,5), (2,6), (3,4), (3,5), (3,6)]`.

```
fun cross a b = ...
```

4. Revisit the previous question and write an ML function `crossHigh` that computes the cross-product in $\Theta(n * m)$ (where n and m are the lengths of the input lists) while using exclusively higher order functions (i.e., nothing **you** write can be recursive). Feel free to use the `append` list function:`op@`.

```
fun crossHigh a b = ...
```

5. Finally, revisit the question one last time and produce an implementation that does not use `op@` and only relies on folding.

```
fun crossFold a b = ...
```

Have fun!