COMP 302 Programming Languages and Paradigms Assignment 3

Prakash Panangaden
McGill University: School of Computer Science

Due Date: 31^{st} **January 2020 3pm**

There are 4 questions to be solved on the LearnOCaml system. There is a spiritual growth question that you could think about if you are interested but it has nothing to do with the class material; it will not help you do well in the class.

We will test to see that you made two recursive calls in Q4, so you cannot just implement insertion sort. You cannot use any built-in sorting function.

Q1. [24 points]

Write a function `common` that takes a pair of lists and forms a new list containing a *unique* copy of each element that occurs in both lists. Here is the type, as echoed by the interpreter, and an example.

```
val common : 'a list * 'a list -> 'a list = <fun>

# let l1 = [1;3;2;4;1;5;6;3];;
val l1 : int list = [1; 3; 2; 4; 1; 5; 6; 3]
# let l2 = [3;9;8;2;11;21;3];;
val l2 : int list = [3; 9; 8; 2; 11; 21; 3]
# common (l1,l2);;
- : int list = [3; 2]
```

It does not matter if the final list is sorted or not. The input lists are not necessarily sorted. Indeed it should work on lists even when there is no notion of ordering the elements in the lists. You may find `List.mem` and `List.filter` useful.

The following three questions are about merge sort. The mergesort algorithm is a recursive algorithm for sorting lists which runs in time $O(n \log n)$. The items in the list must have an order relation defined on them, otherwise sorting does not make sense, of course.

The idea is as follows: the given list l is split into two equal (if the length of l is odd then one of the "halves" is one item longer than the other) lists l_1 and l_2. These lists are sorted recursively and then the results are merged back to give a single sorted list. Code this in OCaml. Your algorithm can use $<$ as a comparison operator. Your code *must* have a function `split` that produces a pair of lists, a function `merge` that merges *sorted* lists and a function `mergesort` that implements the overall algorithm. For testing only use integer lists.

Q2. [26 points]

In this question you will implement `split` with the following type:

```
val split : 'a list -> 'a list * 'a list = <fun>

# split [1;3;2;4;5;6;9;11;17;13;12];;
- : int list * int list = ([1; 2; 5; 9; 17; 12], [3; 4; 6; 11; 13])
```

Q3. [24 points] In this question you will implement the `merge` algorithm. The inputs are *sorted* lists.

```
val merge : 'a list * 'a list -> 'a list = <fun>

# let l3 = [1; 3; 5; 7; 9];;
val l3 : int list = [1; 3; 5; 7; 9]
# let l4 = [2; 4; 6; 8];;
val l4 : int list = [2; 4; 6; 8]
# merge (l3,l4);;
- : int list = [1; 2; 3; 4; 5; 6; 7; 8; 9]
```

Q4.[26 points] Finally we complete the mergesort algorithm. Here is the type and an example.

```
val mergesort : 'a list -> 'a list = <fun>

# mergesort [10;2;8;5;1;4;3;9;7;6];;
- : int list = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10]
# mergesort [1;3;2;4;1;2;5;3];;
- : int list = [1; 1; 2; 2; 3; 3; 4; 5]
```

Q5. [0 points] This question is for your spiritual growth only. Do not think it will give you extra credit or help you learn the material better. It will however stretch your brain in other directions. Do not attempt it if you have not yet finished the required homework. Do not submit a solution; please talk to me if you have solved it. Do not worry about it if you don't understand the question.

How many comparisons are *required* to find the largest member of a list of n elements? [Easy] How many are required to find the largest *and* the smallest. [Not so easy] Here "required"

means that in the worst case you will *have to do* that many comparisons. Can you find the smallest and the largest with fewer than $2n - 3$ comparisons? Note that we are interested in exact counts; not just in $O(\cdot)$ estimates. Can you *prove* your claims?