**COMP 212 : Functional Programming, Spring 2022**

# Homework 08

Name: ______________________________

Wes Email: ______________________________

| Question | Points | Score |
|---|---|---|
| 1 | 12 | |
| 2 | 25 | |
| Total: | 37 | |

If possible, please type/write your answers on this sheet and upload a copy of the PDF to your google drive handin folder. Otherwise, please write the answers in some sort of word processor and upload a PDF. Please name the file `hw08-written.pdf`.

1. **Analysis**

   (a) The following append function was a task in lab (see the lab handout and the lecture notes for Lect 15-16 for an explanation of how tabulate works):

```
fun myAppend (s1 : 'a Seq.seq, s2 : 'a Seq.seq) : 'a Seq.seq =
    Seq.tabulate (fn i => case i < Seq.length s1 of
                      true => Seq.nth (i, s1)
                    | false => Seq.nth (i - (Seq.length s1), s2),
                  Seq.length s1 + Seq.length s2)
```

(2) i. Give a tight $O$-bound for the work of `myAppend`. Make sure you explicitly state what quantities you are analyzing the work in terms of. Briefly explain why your answer is correct.

**Solution:**

(2) ii. Give a tight $O$-bound for the span of `myAppend`. Make sure you explicitly state what quantities you are analyzing the span in terms of. Briefly explain why your answer is correct.

**Solution:**

(b) Consider the following reverse function:

```
fun reverse' (s : 'a Seq.seq) : 'a Seq.seq =
    Seq.reduce (fn (x,y) => myAppend (y, x),
                Seq.empty(),
                Seq.map (Seq.singleton, s))
```

`Seq.singleton` and `Seq.empty` take constant time.

(2) i. Give a tight $O$-bound for the work of `reverse'`, in terms of the length of `s`. Briefly explain your answer.

**Solution:**

(2) ii. Give a tight $O$-bound for the span of `reverse'`, in terms of the length of `s`. Briefly explain your answer.

**Solution:**

(c) Consider the following alternative implementation of the reverse function:

```
fun reverse (s : 'a Seq.seq) : 'a Seq.seq =
    Seq.tabulate (fn i => Seq.nth ((Seq.length s) - (i + 1), s), Seq.length s)
```

(2) i. Give a tight $O$-bound for the work of `reverse`, in terms of the length of `s`. Briefly explain why there is a discrepancy between this and the work of `reverse'`.

**Solution:**

(2) ii. Give a tight $O$-bound for the span of `reverse`, in terms of the length of `s`. Briefly explain why there is a discrepancy between this and the span of `reverse'`.

**Solution:**

2. **Map Fusion**

Earlier in the course, we had a function `raiseBy : int list * int -> int list` that added its `int` argument to each element of the `int list`. We proved that for all values `l : int list`, `a: int`, `b: int`,

$$\mathtt{raiseBy}(\mathtt{raiseBy}(\mathtt{l},\mathtt{a}),\mathtt{b}) \cong \mathtt{raiseBy}(\mathtt{l},\mathtt{a}+\mathtt{b})$$

This is a special case of a property called *map fusion*. Recall the `map` function:

```
fun map (f : 'a -> 'b, l : 'a list) : 'b list =
    case l of
        [] => []
      | x :: xs => f x :: map (f,xs)
```

Mapping `f` over some list `l` and then mapping another function `g` over the result gives a list that is equivalent to the one you would get if you map `fn x => g (f x)` ("g composed with f") over the original `l`. We can write this more concisely by defining an abbreviation for function composition, the function that applies `f` and then applies `g`:

```
fun (g : 'b -> 'c) o (f : 'a -> 'b) = fn x => g (f x)
```

The property we would like to prove is that for all lists `l`,

$$\texttt{map (g o f, l)} \cong \texttt{map(g, map f l)}$$

Unfortunately, this is *false* for certain `g` and `f`. We say that a function `f` is *total* if for all values `v`, `f v` is valuable: that is, a function is total iff it is valuable on all inputs.

(5) (a) Give functions `g` and `f` and a list `l` such that `map (g o f, l` and `map (g, map (f, l))` have different behaviors. Hint: consider `f` and `g` that are not total.

**Solution:**

However, we *can* prove this property for total `f` and `g`.

You may assume the following lemma:

**Lemma 1.** *For all types* `a`, `b` *and values* `f : a->b`, *if* `f` *is total then* `map(f,l)` *is valuable.*

Your job is to prove

**Theorem 1.** *For all types* `a`, `b`, `c`, *all values* `f : a -> b` *and* `g : b -> c`, *and* `l : a list` *if* `f` *and* `g` *are total, then*

$$\texttt{map(g, map(f,l))} \cong \texttt{map(g o f, l)}$$

Proceed by induction on the structure of `l`. **Be careful to note where you are using valuability, and explain why the expressions involved are valuable—where would your proof break for the non-total functions in your example above?**

(5) (a) **Solution:** Case for `[]`:

*To show:*

(15) (b) **Solution:** Case for `x::xs`, where `x` and `xs` are values:

*IH:*

*To show:*

**Solution:** (continue here if necessary)