CMSC330 Fall 2013 Quiz #3

Name __________________________________________

Discussion Time    9am    10am    11am    Noon    1pm
TA Name (circle):  Ilse   Daniel   Casey   Yoav   Ilse
                   Richard   Richard   Richard

Instructions
• Do not start this test until you are told to do so!
• You have 15 minutes for this quiz.
• This is a closed book exam. No notes or other aids are allowed.
• For partial credit, show all of your work and clearly indicate your answers.
• Write neatly. Credit cannot be given for illegible answers.

1. (6 pts) OCaml Types and Type Inference
   a. (2 pts) Give the type of the following OCaml expression
      fun d o g -> [ g ; o ]           Type =
   
   b. (2 pts) Write an OCaml expression with the following type
      'a -> 'a list -> 'a list           Code =

   c. (2 pts) Give the value of the following OCaml expression. If an error exists,
      describe the error.
      let x = 1 in let x = x+2 in let x = x+3 in x+1       Value =

2. (14 pts) OCaml Programming
   a. (6 pts) Write a curried function apply_n which when given an int n, a function f,
      and a value v, applies f to v n times (i.e., evaluates f(f(f(…f(v))). You may assume
      initially n ≥ 0. For n=0 simply return v. You do not need to use map/fold.

      Example:
      let f x = x + 1;;
      apply_n 0 f 1       // returns 1
      apply_n 1 f 1       // returns 2
      apply_n 5 f 1       // returns 6
b. (8 pts) Using either map or fold and an anonymous function, write a function `bubble` which when given a list, returns a list created by going through the list from left to right and swapping two adjacent ints `x` and `y` when `x > y`. Applying `bubble` `n` times to a list of length `n` would result in a list sorted in ascending order (i.e., bubble sort). You may assume list elements may be compared with `>`.  

You are allowed to use `List.rev` (reverses a list) and the (curried) map and fold functions provided, but no other OCaml library functions. Your solution must run in O(n) time for input lists of length n (note that using append instead of prepend will usually make your algorithm O(n^2)). You are not allowed to use imperative OCaml features such as `int ref`.

```
let rec map f l = match l with
  | [] -> []
  | (h::t) -> (f h)::(map f t)

let rec fold f a l = match l with
  | [] -> a
  | (h::t) -> fold f (f a h) t
```

Example:

```
bubble [] ;;     // returns []
bubble [2;1] ;;   // returns [1;2]
bubble [3;2;1] ;;    // returns [2;1;3]
bubble [3;2;4;1] ;;    // returns [2;3;1;4]
apply_n 4 bubble [3;2;4;1] ;;    // returns [1;2;3;4]
```