CS312 Preliminary Exam 1
Fall 2007

There are 5 problems on this exam. You have one and a half hours for the exam. This is a closed-book examination; you *may not* use outside materials.

Name : ______________________________

Net ID: ______________________________

Problem	Points	Score
1	20	
2	24	
3	24	
4	16	
5	16	
Total	100	

1. Short Questions [20 pts] (parts a–d)

(a) [5 pts] Consider the following function definition: `fun f x y = y x x`
Pick the appropriate type for `f` from the list below:

A. `'a -> ('a -> 'a -> 'b) -> 'b`
B. `'a -> ('a -> 'b) -> ('a -> 'b)`
C. `'a -> ('a -> 'b) -> 'b`
D. `'a * ('a -> 'b) -> ('b -> 'a)`

The type of `f` is:

(b) [5 pts] Show the evaluation of the expression below using the substitution model:

```
let fun foo (x:int) (y:int) : int = x * y
    fun bar (x:int->int, y:int) : int = x y
in
    bar(foo 2, 3)
end

->
```

(c) [5 pts] Which of the following equalities are true:

(A)	`[]::x = [x]`	(D)	`x::[] = x`
(B)	`[]::x = [[],[x]]`	(E)	`x::[] = [x]`
(C)	`[]::x = [[],x]`	(F)	`x::[] = [x, []]`

True equalities:

(d) [5 pts] Order the following functions in ascending order with respect to their order of growth: $n^{2.1}$ $\quad n\,log\,(n^2)$ $\quad 2n^2 + n$ $\quad (log\,n)^2$.

Answer:

2. List Manipulations [24 pts] (parts a–b)

(a) [12 pts] Consider a function `remthree` that removes every third element from a list. For instance:

```
remthree [1, 2, 3, 4, 5, 6, 7, 8] = [2, 3, 5, 6, 8]
```

Use pattern matching to implement function `remthree`.

(b) [12 pts] Consider a function that removes all consecutive duplicates from a list. For instance:

```
remdups [1, 2, 2, 3, 3, 3, 1, 1] = [1, 2, 3, 1]
```

The function can be implemented using fold, as follows:

```
val remdups : int list -> int list = foldr func []
```

where `func` is an appropriate function and `foldr` has the standard definition:

```
fun foldr(f: 'a * 'b -> 'b) (v:'b) (l:'a list) =
    case l of [] => v
            | h::t => f(h, foldr f v t)
```

Write the code for function `func`. Make sure you indicate the types of the argument(s) and return value in the function definition.

3. **Data Abstraction** [24 pts] (parts a–d)

The following defines a data abstraction interface and a partial implementation of this interface:

```
signature INTSET = sig
  (* A "set" is a set of integer numbers.
   * Examples: {3, 1, 2}, {~1, 1}, {} *)
  type set

  (* empty() returns an empty set. *)
  val empty : unit -> set
  (* add(s,n) adds element n to set s. *)
  val add: set * int -> set
  (* ... *)
  val oper: set * set -> set
  (* size(s) is the number of elements in the set s. *)
  val size: set -> int
end

structure Set : INTSET = struct
  type set = int list

  fun empty() = []
  fun add(s, n) = raise Fail "unimplemented!"
  fun size l = List.length l

  fun oper(s1, s2) =
     case (s1, s2) of
        ( ([],_) | (_, []) ) => []
      | (h1::t1,h2::t2) => case Int.compare(h1,h2) of
                              EQUAL => h1::oper(t1,t2)
                            | LESS => oper(t1,s2)
                            | GREATER => oper(s1,t2)
end
```

(a) [4 pts] Write an appropriate specification for function `oper`, specifying the operation that it implements.

(b) [6 pts] Write an appropriate abstraction function and representation invariant for this implementation.

(c) [4 pts] Explain why the invariant must hold for this implementation.

(d) [10 pts] Write the appropriate code for function `add`.

4. **Induction** [16 pts]

Consider a tree data type and two functions, `edges` and `nodes`, defined for this type:

```
datatype tree = Nil | Node of int * tree * tree

fun nodes(l: tree): int =
  case l of Nil => 0
          | Node(_, l, r) => 1 + nodes(l) + nodes(r)

fun edges(l: tree): int =
  case l of Nil => ~1
          | Node(_, l, r) => 2 + edges(l) + edges(r)
```

Use induction to prove that `nodes(t) = edges(t) + 1` for all trees. Make sure you clearly show all of the steps in your proof, and state the kind of induction you're using.

5. **Complexity** [16 pts] (parts a–b)

Consider the following implementations for an exponentiation function that raises a number `x` to a power `n`:

```
(* Returns x^n. Requires n >= 0. *)
fun exp1 (x:int, n:int) =
 case (n, n mod 2) of
   (0, _) => 1
 | (_, 0) => x * x * exp1(x, n - 2)
 | (_, _) => x * exp1(x, n - 1)

(* Returns x^n. Requires n >= 0. *)
fun exp2 (x:int, n:int) =
  case (n, n mod 2) of
    (0, _) => 1
  | (_, 0) => exp2 (x * x, n div 2)
  | (_, _) => x * exp2(x, n - 1)
```

(a) [10 pts] For each of the functions above, derive recurrence relations describing their running time $T(n)$, where n is the second argument of each function.

(b) [6 pts] State the asymptotic complexity of each of the two functions with respect to n, their second argument. You don't need to prove your results.