Solutions

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)`

Answer: *(A)* `'a -> ('a ->'a ->'b) -> 'b`.

(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
```

```
-> let fun bar (x:int->int, y:int) : int = x y
   in  bar ((fn x => fn y => x * y) 2, 3) end
-> (fn (x,y) => x y) ((fn x => fn y => x * y) 2, 3)
-> (fn (x,y) => x y) (fn y => 2 * y, 3)
-> (fn y => 2 * y) 3
-> 2 * 3
-> 6
```

(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, []]`

Only (E) is true.

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

Answer: $(log\,n)^2$ $n\ log\,(n^2)$ $2n^2+n$ $n^{2.1}$

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`.

Answer:

```
fun remthree(l: int list) : int list =
  case l of
    [] => []
  | [_] => []
  | [_,x] => [x]
  | _::x::y::t => x::y::remthree(t)
```

(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.

Answer:

```
fun func(x:int, a:int list): int list =
    case a of [] => [x]
            | h::t => if x = h then a else x::a
```

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.

Answer:

```
(* oper(s1,s2) returns the intersection of sets s1 and s2 *)
```

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

Answer:

```
(* AF: the list [x1,x2,..,xn] represents the set {x1,x2,..,xn}.
   RI: the list [x1,x2,..,xn] is sorted in increasing order
       and contains no duplicates.  *)
```

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

Answer: *The list must contain no duplicates to ensure that* `size()` *works correctly. The list must be sorted so that* `oper()` *correctly computes set intersection.*

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

Answer:

```
fun add(s: set, n: int): set =
    case s of
       nil => [n]
    | h::t => (case Int.compare(n,h) of
                    LESS => n::s
                  | EQUAL => s
                  | GREATER => h::add(t,n))
```

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.

Answer:

Claim: Let P(n) be the statement: nodes(t) = edges(t) + 1 where t is a tree with n nodes, such that $n \geq 0$.

Proof: by strong induction on n, the number of nodes in the tree.

Base Case: P(0) is the statement: nodes(Nil) = 0 and edges(Nil) = 1. Thus nodes(Nil) = 1 + edges(Nil), so the base case holds.

Inductive Step: Assume P(k) holds for all $k \leq n$. *We prove P(n+1). By the substitution model, nodes(t) evaluates to 1 + nodes(l) + nodes(r) where l and r are subtrees with at most n nodes. By our induction hypothesis, nodes(l) = edges(l) + 1 and nodes(r) = edges(r) + 1.*

Putting these together, we have nodes(t) = 1 + (edges(l) + 1) + (edges(r) + 1) = 3 + edges(l) + edges(r).

Also by the substitution model, edges(t) evaluates to 2 + edges(l) + edges(r). So now we have nodes(t) = edges(t) + 1. Therefore P(n+1) holds.

Conclusion: We have proved P(n) implies P(n+1). Therefore nodes(t) = edge(t) + 1 for $n \geq 0$, *as claimed.*

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.

Answer: *Relations for* `exp1`*:*

$$\begin{aligned} T(0) &= c_0 \\ T(2n) &= T(2(n-1)) + c_1 \\ T(2n+1) &= T(2n) + c_2 \end{aligned}$$

Combining the contants in the last two relations we get the following system of recurrences:

$$\begin{aligned} T(0) &= c_0 \\ T(n) &= T(n-1) + c_3 \end{aligned}$$

Relations for `exp2`*:*

$$\begin{aligned} T(0) &= c_0 \\ T(2n) &= T(n) + c_1 \\ T(2n+1) &= T(2n) + c_2 \end{aligned}$$

The last recurrence can be rewritten:

$$T(2n+1) = T(2n) + c_2 = T(n) + c_1 + c_2 = T((2n+1)/2) + c_1 + c_2)$$

The recurrence for $T(2n)$ can be rewritten:

$$T(2n) = T(n) + c_1 = T((2n)/2) + c_1$$

Combining these relations we get the following system:

$$
\begin{array}{rcl}
T(0) & = & c_0 \\
T(n) & = & T(n/2) + c_3
\end{array}
$$

(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.

Answer:

Answer: *Function* `exp1` *is O(n). Function* `exp2` *is O(log n).*