Solutions

1. True/False [20 pts] (parts a–j)
   Each correct answer is 2 pts; each wrong answer is -2 pts; and each blank answer is 0 pts.

   (a) Software testing proves the presence of bugs, but cannot prove their absence.
      True

   (b) The type that SML infers for the expression fn (x,y) => fn x => (y,x) is:
      'a * 'b -> 'c -> 'b * 'c.
      True

   (c) The function f(n) = lg(n lg n) is O(lg n).
      True

   (d) At each collection, a copying collector must traverse all of the data in the program
       (including data unreachable from the roots).
      False

   (e) The implementation of Dijkstra’s shortest-paths algorithm requires a stack data structure.
      False

   (f) The function foldl is tail-recursive.
      True

   (g) It is possible that a hash table with n elements and a load factor of 2 has a bucket that
       contains all of the n elements.
      True

   (h) When a program exhibits temporal locality, it will access the same memory location in
       the near future.
      True

   (i) Any lookup operation in a splay tree with n nodes is O(lg n).
      False

   (j) Data races can occur during the execution of message-passing concurrent programs.
      False
2. Sets [20 pts] (parts a–c)

The following is a standard set interface:

signature SET = sig
  (* A 'a set is a set of items of type 'a. *)
  type 'a set

  (* empty is the empty set *)
  val empty : 'a set

  (* add(s,e) is s union {e} *)
  val add: 'a set * 'a -> 'a set

  (* fold over the elements of the set *)
  val fold: ('a*'b->'b) -> 'b -> 'a set -> 'b
end

(a) [5 pts] Extend the interface with a function remove that removes an item from a set. Provide a signature and a specification for remove. Define an appropriate exception if necessary.

Answer:

exception NotFound

(* remove (s,e) is the set s - {e} *)
* Raises: NotFound if e does not belong to s *)
val remove: 'a set * 'a -> 'a set

(b) [7 pts] Write an implementation for function remove using the other functions in the signature. Assume that an equality funtion for items equal:'a*'a->bool is also available ('a being the type of the items in the set). Your function remove should not visit any of the items more than once.

Answer:

fun remove(s, e) =
  let val (r,f) = fold (fn (x,(s',f)) => if equal(x,e) then (s',true)
    else (add(s',x),f))
    (empty,false) s
  in
    if f then r else raise NotFound
  end

(c) [8 pts] Consider now a function cartprod that takes two sets of items and yields a set of pairs representing the Cartesian product:

(* cartprod(s1,s2) is the cartesian product of s1 and s2 *)
val cartprod: 'a set * 'a set -> ('a * 'a) set

Remember that the Cartesian product $A \times B$ of two sets $A$ and $B$ is the set of all pairs $(a,b)$ where $a \in A$ and $b \in B$. That is, $A \times B = \{(a,b) \mid a \in A, b \in B\}$.

For simplicity, assume that sets are implemented using lists (type 'a set = 'a list).

Below are some examples of using cartprod:
cartprod ([1,2], [3,4]) = [(1,3), (1,4), (2,3), (2,4)]
cartprod ("a", ["b", "c"]) = ["a", "b", "a", "c"]
cartprod ([1,2], []) = []

Write the function cartprod, assuming a list implementation of sets. You may not use
the list concatenation operator "@" in your solution.

Note: It is possible to write cartprod such that it works for any implementation of sets,
not only lists. Feel free to write such a function.

Answer: The implementation of cartprod for lists, without folding:

fun cartprod(s1, s2) =
    case (s1,s2) of
    ([],_) => []
   | (h1::t1, _) =>
       let fun prod(s) = case s of
       [] => cartprod(t1,s2)
       | h2::t2 => (h1,h2)::prod(t2)
     in
     prod(s2)
   end

The general implementation of cartprod using fold is simpler:

fun cartprod(s1, s2) =
fold (fn (x,s') =>
       (fold (fn (y,s'') => add (s'',(x,y))) s' s2))
  empty s1

3. Trees [20 pts] (parts a–c)

The following is the standard datatype for binary search trees containing integer values:

datatype tree = Leaf | Node of tree * int * tree

(a) [5 pts] Consider two functions min and max that compute the smallest and the largest
numbers in a tree:

(* min(t) is the smallest element of t,
 * or the largest integer if t is a leaf. *)
val min : tree -> int

(* max(t) is the largest element of t,
 * or the smallest integer if t is a leaf. *)
val max : tree -> int

Using these functions, write a function repOK : tree -> bool that returns true if and
only if the tree satisfies the binary search tree invariant. (For an informal description of
the invariant you’ll receive partial credit).

Answer:

fun repOK(t) =
    case t of Leaf => true
   | Node(l, v, r) => max(l) <= v andalso v <= min(r)
       andalso repOK(l) andalso repOK(r)
(b) [5 pts] Several kinds of binary trees (including AVL, red-black, and splay trees) use rotations for rebalancing. The following is the basic right rotation:

```ml
fun rotate (t:tree) :tree =
  case t of
    Node(Node(A,x,B), y, C) => Node(A, x, Node(B,y,C))
  | _ => t
```

Show that the above function `rotate` maintains the binary search tree invariant.

**Answer:** Assume that the invariant holds for `t` at the beginning of `rotate`.

On the first arm of the case construct, `t` matches the pattern `Node(Node(A,x,B), y, C)`.

Therefore, \( \max(A) \leq x \leq \min(B) \leq \max(B) \leq y \leq \min(C) \). Also, all of the nodes in `A`, `B`, and `C` satisfy the binary search tree invariant. This shows that the tree `Node(A,x,Node(B,y,C))` is also a binary search tree.

On the second arm of the case, the invariant trivially holds because the returned tree is identical.

(c) [10 pts] Consider now an imperative implementation of binary search trees:

```ml
datatype itree = Leaf | Node of (itree ref) * int * (itree ref)

(* irotate(t) performs a right rotation at the root of t
 * Effects: destructively updates t
 * Returns: the new root after the rotation. *)
val irotate: itree -> itree
```

Write an implementation for `irotate`.

**Answer:**

```ml
val irotate(ty:itree): itree =
  case ty of
    Node(L as ref(tx as Node(A,x,B)), y, C) =>
      (L := !B; B := ty; tx)
  | _ => ty
```

4. **Correctness and Complexity** [20 pts] (parts a–g)

Consider the following program:

```ml
fun f (x, y) =
  if y = 0 then 1 else
  let
    val p = f (x, y div 2)
    val sp = p * p
  in
    if y mod 2 = 0 then sp else x * sp
  end
```

(a) [3 pts] What does `f(x,y)` compute?

**Answer:** `f(x,y)` is \( x^y \).
The following questions ask you to prove the correctness of function \( f \), with respect to your answer above.

(b) [2 pts] Write the property \( P(n) \) that you need to prove and specify the initial value \( n_0 \) of \( n \).

**Answer:** \( P(n) = f(x, n) \) is \( x^n \), for all \( x \). The initial value of \( n \) is \( n_0 = 0 \).

(c) [2 pts] State whether you’ll use strong or weak induction.

**Answer:** Strong induction.

(d) [2 pts] Prove the base case.

**Answer:** For \( n = 0 \) and for any \( x \), \( f(x, n) = f(x, 0) \) evaluates to 1. Since \( x^0 = 1 \), we get that \( f(x, 0) = x^0 \) for all \( x \).

(e) [4 pts] State the induction hypothesis and prove the induction step.

**Answer:** Induction Hypothesis: assume that \( P(m) \) holds for any \( 0 \leq m < n \). We want to prove that \( P(n) \) holds.

Since \( n > 0 \), \( P(n) \) evaluates the false arm of the first if statement. On that branch, the program computes \( p = f(x, \lfloor n/2 \rfloor) \) and \( sp = p^2 \). By IH, because \( \lfloor n/2 \rfloor < n \), we get that \( p = x^{\lfloor n/2 \rfloor} \). Hence, \( sp = x^{2\lfloor n/2 \rfloor} \). Note that \( 2\lfloor n/2 \rfloor \) is not necessarily equal to \( n \), because \( \lfloor n/2 \rfloor \) is the integer division.

We have two cases. If \( n \bmod 2 = 0 \), the program executes the first branch of the inner if expression. In this case, \( n = 2k \) for some \( k \), so \( p = x^k \) and \( sp = p^2 = x^{2k} = x^n \). The returned value is \( sp \), so \( f(x, n) = x^n \).

If \( n \bmod 2 = 1 \), the program executes the second branch. In this case, \( n = 2k + 1 \) for some \( k \), so \( p = x^k \) and \( sp = p^2 = x^{2k} = x^{n-1} \). The returned value is \( x \ast sp \), so \( f(x, n) = x \ast sp = x^n \).

In either case, \( f(x, n) = x^n \), which completes the proof.

Next, analyze the run-time complexity of \( f \).

(f) [4 pts] Write the recurrence relations for the running time of \( f(2, n) \). Use constants \( c_1, c_2, \) etc. for operations that take constant time.

**Answer:** Let \( T(n) \) be the running time of \( f(2, n) \). Then:

\[
T(0) = c_1
\]
\[
T(n) = T(\lfloor n/2 \rfloor) + \begin{cases} 
  c_2 & \text{if } n \text{ is even} \\
  c_2 + c_3 & \text{if } n \text{ is odd}
\end{cases}
\]

(g) [3 pts] What is the run-time complexity of \( f(2, n) \)? You don’t have to prove your result.

**Answer:** \( T(n) \) is \( O(\log n) \).

5. Environment Model [20 pts] (parts a–d)

The program below is written in a ML-like language that doesn’t allow recursive functions, but has references and higher-order functions:

```ml
let val rf = ref (fn x => x)
val f = fn y => let val (n,p,q) = y in

```
if n = 0 then y else (!rf)(n - 1, q, p + q) end

val () = rf := f in

f(5,1,1)
(* GC *)
end

(a) [3 pts] What are the types of f and rf?
Answer:
f : int * int * int -> int * int * int
rf : (int * int * int -> int * int * int) ref

(b) [3 pts] What is the result of the evaluation?
Answer: (0,8,13).

(c) [10 pts] Draw the environment diagram that arises during the evaluation of the second call to f (i.e., when the program starts evaluating the function body for that call).
Answer:

(d) [4 pts] What heap cells (not environment entries!) can a garbage collector reclaim at the program point labeled GC in the code?
Answer: The collector can reclaim the closure fn x => x, as well as five tuples created during the execution: (5,1,1), (4,1,2), (3,2,3), (2,3,5), and (1,5,8).
The cells that cannot be collected are: the closure fn y => ..., the ref cell, and the returned tuple (1,8,13). The first two cannot be reclaimed yet because they are still reachable from variables in scope.