Prelim Sample Solution by Dan Grossman

CS410, Summer 1998

17 July 1998

1. Assume $f(n)$ is $O(g(n))$. Then for some constants c, n_o, by definition $f(n) < cg(n)$ for all $n > n_o$. By math, $g(n) > \frac{1}{c}f(n)$ for all $n > n_o$. Since c is a constant, so is $\frac{1}{c}$. Then by definition, $g(n)$ is $\Omega(f(n))$.

2. (a) If $T(n) = aT(n/b) + cn^k$ for constants a, b, c, k, then the Master Theorem can solve for $T(n)$.

   (b) T(n) = T(n-1) + O(1)

   (c)
   ```
   int addArray(int [] arr) {
       int total = 0;
       for (int i = 0; i < arr.length; i++)
           total += arr[i];
       return total
   }
   ```
   The "n" in the recurrence relation refers to the length of `arr`.

3. (a)
   ```
   Object lookup (int key) {
       ListNode x = head;
       while (x.kth.key < key)
           x = x.kth;
       while (x.next.key < key)
           x = x.next;
       return x.val;
   }
   ```

   (b) (This was the extra credit, but goes here logically.)
   ```
   void delete(int key) {
       ListNode x = head;
       while (x.kth.kth.key < key)
           x = x.kth;
       while (x.kth.next.key < key)
           x = x.next;
       while (x.next.key < key) {
           x.kth = x.next.kth;
           x = x.next;
       }
       x.kth = x.next.kth;
       x = x.next;
   }
   ```

4. (a)
   ```
   void printAll(Tree t) {
       Stack s = new Stack();
       s.push(t.root);
       while (!s.isEmpty()) {
           Node n = s.pop();
           print(n);
           if (n.right != null)
   ```

```
            s.push(n.right);
    if (n.left != null)
            s.push(n.left);
     }
}
```

(b)
```
void printAll(Tree t) {
     Queue q = new Queue();
     s.push(t.root);
     while (!q.isEmpty()) {
         Node n = q.dequeue();
         print(n);
         if (n.left != null)
            q.enqueue(n.left);
         if (n.right != null)
            q.enqueue(n.right);
     }
}
```

(c) The only differences are using a stack or queue and whether we put in the left or the right first. We can abstract the differences by extending each ordering policy from a common abstract class. (Saying so was enough for full credit). For purposes of explanation, here's the full code; you did not have to write all this on your exam.

```
abstract class OrderChildren {
   void putChildren(Node left, Node right) {
        Node first, second;
        if (leftFirst()) {
            first = left;
            second = right;
        } else {
            first = right;
            second = left;
        }
        if (first != null)
           put(first);
        if (second != null)
           put(second);
   }
   abstract boolean leftFirst();
   abstract void put(Node n);
   abstract Node get();
   abstract boolean isEmpty();
}

class PartA extends OrderChildren {
   private Stack stack = new Stack();
   boolean leftFirst() { return false; }
   void put(Node n) { stack.push(n);}
   Node get() { (Node) stack.pop();}
   boolean isEmpty() { return stack.isEmpty();}
}

class PartB extends OrderChildren {
   private Queue queue = new Queue();
   boolean leftFirst() { return true;}
```

```
    void put(Node n) { queue.enqueue(n); }
    void get() { (Node) queue.dequeue();
    boolean isEmpty() { return queue.isEmpty();}
}

void printHelper(Tree t, OrderChildren h) {
    Node x = t.root;
    h.put(x);
    while (!h.isEmpty()) {
        x = h.get();
        print(x);
        h.putChildren(x);
    }
}

void doPartA (Tree t) {
    printHelper(t, new PartA());
}

void doPartB (Tree t) {
    printHelper(T, new PartB());
}
```

5. (a) 2-3 trees support insert, min, and delete in $O(\log n)$ time. The min is at the leftmost leaf. To delete it, we take it out and merge nodes to update minimum-value information and eliminate any nodes having one child. To insert a node, we put it at the correct position among the sorted leaves, update any minimum-value information and split nodes to eliminate any nodes having four children.

   (b) On insert, we can put the new value anywhere at the levaes we like and do the rebalancing as before. To find the min, we cannot just keep going left. Instead we must go to the child with the lowest min value stored at it. But this still supports finding the min in time $O(\log n)$. The deletion is as before.

   (c) Let the height of P_1 be h_1 and the height of P_2 be h_2. Make the root of P_2 be the child of any node in P_1 at height $h_1 - h_2 - 1$. Rebalance up revising minimum-value information and eliminating any nodes having four children.