Quiz 2 *Name:* _____

10/17/12

1. For each code snippet, state whether it is safe and argue why or why not.

(a)

```
1  def hist(a:Rail[Int], b: Rail[Int]) {
2      finish for(var i:int=0; i < a.length; i++) async {
3         finish {
4            val bin = a(i) % b.length;
5            atomic b(bin)++;
6         }
7      }
8  }
```

This code is safe. Elements in Rail b are updated atomically (i.e., operations on shared mutable state commute) so the parallel program gives the same answer as a serial one.

(b)

```
1  class Node {
2     val id:Int;
3     var edges:Rail[Node];
4     var parent:Node = null;
5     def this(id:Int,e:Rail[Node]) { this.id=id; this.edges=e; }
6     def mst() { finish traverse(); }
7     def traverse() {
8        for (edge in edges.values()) async {
9              atomic {
10                  if (edge.parent == null) {edge.parent=this; edge.traverse();}
11                  else if (edge.parent.id > this.id) edge.parent.id = this.id;
12             }
13       }
14    }
15 }
```

- Suppose id is unique and "edge.parent.id = this.id" is changed to "edge.parent = this". In that case, this program is safe. It computes a self-defined Minimum-Spanning-Tree, where each node is only connected to the node with smallest id in its original neighbors. Since this tree is unique, the parallel and serial executions give the same result.
- If id is not unique or we still use "edge.parent.id = this.id" the program is not safe, because there are multiple possible results from the serial execution.

2. Consider the serial tree sum code below.

```
1  class Tree {
2      var i:Int = 0;
3      var result:Int;
4      var left:Tree, right:Tree;
5      def this(){}
6      def this(l:Tree, r:Tree) { this.left=l; this.right=r; }
7
8      def sum(nthreads:Int) {
9           if (left != null) ls = left.sum(nthreads);
10          if (right != null) rs = right.sum(nthreads);
11          result = (left == null ? 0 : left.result) + (right == null ? 0 : right.result);
12     }
13 }
```

(a) Write a parallel version of Tree.sum() to that will provide as good speedups as possible on a multi-core machine with 2 to 30 cores. Note that the tree may have millions of nodes. You are free to use async, finish, atomic, when, collecting finish.

(b) Argue why your code will scale well up to 30 threads.

(c) Is your code safe? Why or why not?

See the source code at `http://www.cs.columbia.edu/~martha/courses/4130/au12/Tree.x10`. All versions of parallel sum are safe, as there is no shared mutable state. Only par2 and par3 scale well, with only par3 scaling even with unbalanced trees. See the comments at the top of the file for data.