CS314 Fall 2011 Midterm 2 Solution and Grading Criteria.

Grading acronyms:
AIOBE - Array Index out of Bounds Exception may occur
BOD - Benefit of the Doubt. Not certain code works, but, can't prove otherwise
ECF - Error carried forward.
Gacky or Gack - Code very hard to understand even though it works or solution is not elegant.
GCE - Gross Conceptual Error. Did not answer the question asked or showed fundamental misunderstanding
LE - Logic error in code.
NAP - No answer provided. No answer given on test
NN - Not necessary. Code is unneeded. Generally no points off
NPE - Null Pointer Exception may occur
OBOE - Off by one error. Calculation is off by one.

1. Answer as shown or -2 unless question allows partial credit.
No points off for differences in spacing, capitalization, commas, and braces

A. 15
B. 5
C. 46
D. v v x x u u y y
E. 25
F. 12 seconds
G. quicksort (ave case nlogn, worst case n^2)
H. 1999 (2000 acceptable)
I. O(N^2) (contains is O(N))
J. O(N^2)
K. Sort then search, choice 2
  1000 * 64,000 > 128,000 * 17 + 17 * 1000
  6.4 * 10E7 < 2.2E6

```
L.    -15
         \
         32
        /
       15
      /
     0
      \
       7
```

M. ZACJXHPM
N. CAXJZPHM
O. CXJAPMHZ

2. Comments. I apologize for the comment about O(1) space. The TAs and I did not communicate well over what that meant. Most people ignored that the O(1) and used recursion as I intended. If not, we graded leniently.

I thought this would be an easy problem. A simple tree traversal with a test for a certain property.

Common problems:

- Biggest problem was sending an int variable. It was incremented in recursive calls and this leads to returning an answer that is much too large.
- Not having base case of value = null, not handling empty tree case
- Not making recursive call on single child. That node or nodes deeper in the tree may have nodes with two children.

Suggested Solution:

```
public int numNodesWithTwoChildren() {
     return helper(root);
}

private int helper(BTNode<E> n) {
     if(n == null)
          return 0;
     int count = 0;
     if(n.getLeft() != null && n.getRight() != null)
          count++;
     count += helper(n.getLeft());
     count += helper(n.getRight());
     return count;
}
```

General Grading Criteria: 10 points

- helper 1 point
- correct base case 3 points (null or other checks)
- check two children correctly 2 points
- correct recursive calls 3 points
- return correct answer (local var or multiple returns) 1 point

Analysis:

A. Big O all methods should be O(N) 1 point

B. smallest = 0   2 points

largest = (N / 2 - 1)   N / 2 okay   2 points

3. Comments: A hard linked list problem. There were 3 or 4 special cases which made the question interesting. I saw a variety of correct solutions.

Common problems:

- not handling empty case
- not updating first if tree was not empty (or more than 1 item) but element added is smallest and goes in front
- not using compareTo correctly
- altering the list
- not actually moving through the list
- assuming an iterator was available
- not adding as last node if necessary

Suggested Solution:

```
public boolean add(E val) {
     // empty case and smallest value cases
     if( first == null || val.compareTo(first.getData()) < 0) {
          first = new Node(val, first);
          return true;
     // general case, I will use a trailer
     Node lead = first.getNext();
     Node trail = first;
     while( lead != null ) {
          int diff = val.compareTo(lead.getData());
          if(diff == 0)
               return false; // already here
          else if(diff < 0)
               // this is the right spot, in between trail and lead
               trail.setNext(new Node(val, lead));
               return true;
          trail = lead;
          lead = lead.getNext();
     }
     // if we get here, must add at end. Imagine list with one
     // element and it is larger than first element
     trail.setNext(new Node(val, null));
     return true;
}
```

General Grading Criteria: 20 points

- handle empty case - 2
- adjust first if necessary - 1
- trailer or look ahead - 3
- loop, correct stopping case - 2
- use compareTo correctly - 2
- handle already present correctly - 1
- if time to add, add correctly - 1
- move through list correctly - 4
- add at end if necessary - 1
- return correct value - 1

4. Comments. Probably the easiest coding question. Students did well. A test of using iterators and other classes. The E should have been PairSet<E> but that was ignored for the question. I was more interested in the algorithm.

Common problems:

- using the for each loop, FuzzySets were not iterable
- not resetting inner iterator

Suggested Solution

```
public FuzzySet<E> getFuzzyIntersection(FuzzySet<E> other) {

    FuzzySet<E> result = new FuzzySet<E>();
    Iterator<SetPair<E>> thisIt = iterator();
    while(it.hasNext())
        SetPair<E> thisItem = thisIt.next();

        // look for thisItem in other set

        Iterator<SetPair<E>> otherIt = other.iterator();
        boolean found = false;
        while(!found && otherIt.hasNext()) {
            SetPair<E> otherItem = otherIt.next();

            if(thisItem.getElem().equals(otherItem.getElem())) {
                // found it! add to result and stop looking
                done = true;
                double degree = thisItem.getDegree() *
                                    otherItem.getDegree();
                SetPair newPair = new SetPair(thisItem.getElem(), degree);
                result.add(newPair);
            }
        }
    }
    return result;
}
```

General Grading criteria: 17 points

- create result 1 point
- iterate through one set (order can be switched) 2 points
- nested loop 2 point
- refresh inner iterator each time 3 points
- correct use of iterators 2 points
- inside inner loop check equality of items 2 points
- create new SetPair if match and set degree correctly 2 points
- add new SetPair to result 2 points
- return result 1 point

5. Comments: A hard problem. Having to return an array list of dice positioned correctly made the question a lot harder. A saw a number of different, correct approaches.

Common problems:

- just using a nested loop which generates dice.length * 6 possibilities. There are dice.length ^ 6 possibilities and the nested loop does not try them all.
- not having a base case
- returning early before trying other choices
- not testing to see if a solution was found and stopping if it was
- not removing dice from ArrayList if choice didn't work (unless added all at start. That was a clever alternate solution.)

Suggested Solution:

```
public ArrayList<Die> solvePuzzle(Die[] dice) {
     ArrayList<Die> result = new ArrayList<Die>();
     helper(dice, 0, '?', result);
     return result;
}

private boolean helper(Die[] dice, in pos, char lastColor,
                              ArrayList<Die> result) {
     if(pos == dice.length)
          return true; // solved! dice must be positioned correctly

     // not solved, take 1 die and try all 6 positions
     Die die = dice[pos];
     result.add(die); // adding pointer, so if I change die
                      // changes in result
     for(int i = 0; i < 6; i++) {
          die.positionLeftFace(i);
          // check if first die or matches previous color
          // last color is left side, getColor is right side
          if( pos == 0 || lastColor == die.getColorSide(i)) {
               // matches! go on to next, opposite face is right side
               if(helper(dice, pos + 1, result,
                          die.getColorOppositeSide(i)) {
                    // found solution! stop making choices
                    return true;
               }
          }
     }
     // never found solution, back track
     result.remove(result.size() - 1);
     return false;
}
```

General Grading criteria: 15 points

create helper - 1, correct base case - 3, recursive case, loop 6 sides on one die - 3
position current die - 1, check color matches on dice - 3, go on if current set up okay - 3
check if solved and stop if it is - 3, return correctly answer helper and original - 1