Sample Solution to Midterm Test
CIS 313 - 2004 Winter

#1. [10 points] Consider the following method to sort an array A of n integers using BST methods:

```java
BST T;
for (int i=0; i<n; i++){
    T.insert(A[i]);
}
i = 0;
inorder(T.root());
```

where the procedure inorder is defined as

```java
int inorder(node p) {
    if p!=null then {
        inorder(p.left());
        A[i] = p.value();
        i++;
        inorder(p.right());
    }
}
```

a) What is the worst case running time of this approach?

This is going to be $\Theta(n^2)$, as we may have a skew tree, each insertion into which will be $\Theta(n)$ with n insertions.

b) What is the average case running time? Explain.

The total insertion time is the internal path length of the tree. As we saw in class (Jan. 26), the expected value of $(1+n)/n$ is $\Theta(\log n)$. So the expected value of I is $\Theta(n \log n)$. Since the inorder traversal is $\Theta(n)$, $\Theta(n \log n)$ is the average time for this approach.

#2. [10 points] Use the methods of a queue to take a tree T and print out the values stored at each node in reverse level order. That is, the nodes of each level should be visited right to left, instead of left to right.

The following method to provide a level order traversal was given in class (Jan 14):

```java
queue Q
if T.root()!=null then Q.enqueue(T.root())
while not Q.isEmpty() {
    p = Q.dequeue()
    if p!=null then {
        Q.enqueue(p.left());
        Q.enqueue(p.right());
    }
}
```
print p.value
if p.left!=null then Q.enqueue(p.left)
if p.right!=null then Q.enqueue(p.right) }

For a reverse level order, we simply modify the order in which the children are enqueued:

```java
queue Q
if T.root()!=null then Q.enqueue(T.root())
while not Q.IsEmpty() {
    p = Q.dequeue()
    print p.value
    if p.right!=null then Q.enqueue(p.right)
    if p.left!=null then Q.enqueue(p.left) }
```

#3. **[10 points]** Into an initially empty AVL tree, insert the values 10, 19, 18, 16, 17, 15, 11, 14, 13, 9, 12. Show some of the intermediate steps.

The final tree is as follows:

```
  14
 /   \
11   16
 |   |
10   13
 |   |   |
9    12 15
        |   |
        17 18
        |
        19
```
#4. [10 points] Insert the previous sequence into an initially empty (2,4)-tree. Show the tree after each split (and at the end).

Again, we just show the final tree here:

![Tree Diagram]

#5. [10 points] Give a recursive procedure that, given a node $p$ and two integers $min$ and $max$, will determine the number of nodes in the subtree rooted at $p$ whose values fall between $min$ and $max$ (inclusive). Call your procedure rangeSize($p$, $min$, $max$).

For 8 points, perform a depth first traversal of the entire tree:

```plaintext
procedure rangeSearch(node p, int min, int max)
    total = 0
    if p != null then {
        total += rangeSearch(p.left, min, max)
        if min <= p.value <= max then total +=
        total += rangeSearch(p.right, min, max)
    }
    return total
```

For full credit, prune the search space of the tree. This solution allows for duplicate values – assuming no ties would give a slightly different but acceptable solution.
procedure rangeSearch(node p, int min, int max)
    case
        (p=null)
            return 0
        (max<p.value)
            return rangeSearch(p.left, min, max)
        (max=p.value)
            return 1 + rangeSearch(p.left, min, max)
        (min<p.value<max)
            return 1+rangeSearch(p.left, min, max) +
            rangeSearch(p.right, min, max)
        (p.value<min)
            return 1 + rangeSearch(p.right, min, max)