Problem Set 3

CSCI 136: Fall 2019
Handout PS 3
Due: Friday., Nov. 8 *at start of class*

Instructions: We encourage you to do all of these problems, but please hand in only the ones labeled "Hand In"!

Trees

(*5 points*) ..... Hand In

A node in a tree that is not a leaf node is called an *internal* node. For this problem, we'll say that a binary tree is *dense* if each internal node has exactly 2 children.

Prove that the number of internal nodes in a non-empty dense binary tree is one less than the number of leaves. [Hint: Induction]
Note: The encoding tree produced by Huffman's algorithm is dense, and so this proves that the number of nodes in the tree is proportional to the size of the alphabet being encoded—a claim made, but not proven, in class.

(*0 points*) ..... Practice

The height method in the BinaryTree class looks like this

```
public int height()
{
    if (isEmpty()) return -1;
    return 1 + Math.max(left.height(),right.height());
}
```

Prove that this method correctly determines the height of the binary tree. [Hint: Induction—which kind?]

(*5 points*) ..... Hand In

The isFull method in the BinaryTree class looks like this

```
public boolean isFull()
{
    if (isEmpty()) return true;
    if (left().height() != right().height()) return false;
    return left().isFull() && right().isFull();
}
```

Prove that this method correctly determines whether the tree is full

(*5 points*) ..... Hand In

The levelOrder method for traversing a BinaryTree with root `t` looks like this

```
public static <E> void levelOrder(BinaryTree<E> t) {
        if (t.isEmpty()) return;

        // The queue holds nodes for in-order processing
        Queue<BinaryTree<E>> q = new QueueList<BinaryTree<E>>();
        q.enqueue(t); // put root of tree in queue

        while(!q.isEmpty()) {
                BinaryTree<E> next = q.dequeue();
                touch(next);
                if(!next.left().isEmpty()  ) q.enqueue( next.left() );
```

```
                if(!next.right().isEmpty() ) q.enqueue(next.right());
        }
}
```

Prove that this method runs in time proportional to the number of nodes in the tree with root `t`. [That is, it runs in $O(n)$ time where n is the number of nodes in the tree.]

(*0 points*) ..... Practice

Prove that the PreOrder iterator BTPreOrderIterator visits the nodes of a binary tree in the same order as the preorder traversal method described in class. [The code for both of these can be found in the Lecture20 slide deck.]