Zack Edwards
SSW 315 Homework 4
11/15/2020
I pledge my honor that I have abided by the Stevens honor system - Zack Edwards

1) A full node is a node with two children. Prove that the number of full nodes plus one is equal to the number of leaves in a nonempty binary tree.

   When there is only one root node, there are no full nodes (k) and there is one leaf (k+1), P(k+1)=(0 + 1 = 1).Then when we add a node, either it is a left child and therefore the number of full nodes does not increase and the number of leaves does not increase. If the node is added as a right child of its parent, then the number of full nodes increases by one and so does the number of leaves. Since the number of full nodes starts as one less than the number of leaves, the theorem that the number of leaves equals (k+1) where k is the number of full nodes is true.

2)
   a) Show the result of inserting 3, 1, 4, 6, 9, 2, 5, 7 into an initially empty binary search tree

insert(3)

```
    3
   / \
```

insert(1)

```
    3
   / \
  1
```

insert(4)

```
    3
   / \
  1   4
```

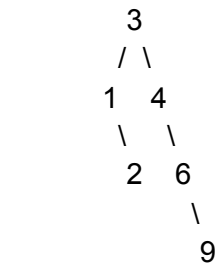
insert(6)

```
    3
   / \
  1   4
       \
        6
```

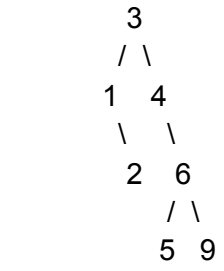
insert(9)

```
    3
   / \
  1   4
       \
        6
         \
          9
```

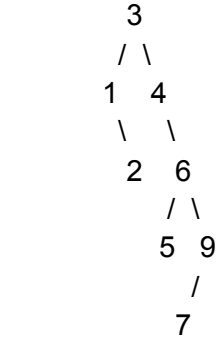
insert(2)



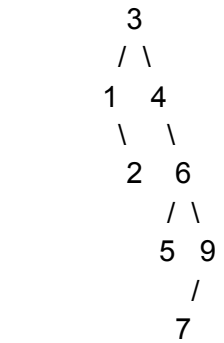
insert(5)



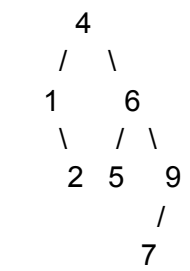
insert(7)



b) Show the result of deleting the root.
Initial tree



deleteRoot



3) Write a routine to list out the nodes of a binary tree in level-order. List the root, then nodes at depth 1, followed by nodes at depth 2, and so on. You must do this in linear time. Prove your time bound

```
public String levelPrint() {
        StringBuilder s = new StringBuilder("");
        Queue q enqueue(root);
        s.append('[');
        while(queue.empty() != true) {
                Node tmp = q.peak();
                s.append(q.dequeue() + ",  ");
                if(tmp.left != null){
                        q.enqueue(tmp.left);
                }
                if(tmp.right != null){
                        q.enqueue(tmp.right);
                }
        }
        s.append( "]");
        return s.toString();
}
```

Since this function visits each node exactly once while traversing, this will result in linear time complexity.

4) Suppose we want to add the operation findKth to our repertoire. The operation findKth(k) returns the kth smallest element item in the tree. Assume all items are distinct. Explain how to modify the binary search tree to support this operation in O(logN) [i.e., O(d)] average time, without sacrificing the time bounds of any other operation.

One way in which you could modify the class to accommodate this method would be to add a variable called ‘size’ to each node, which indicates how many nodes are left child descendants of this node. Then while traversing, compare each node's level to k. If the size is greater then you know the kth smallest element is to the left and you should traverse to the left child. If it is less than k then move to the right child.Everytime you perform this, you cut the size of the tree left to search, and thus the runtime of the search, in half. This results in O(log(n)) for the method findKth().