Jeremiah McDonald

Heaps and Trees

ZYBooks Labs 7 and 8

Lab 7:

```
import java.util.*;

public class BSTChecker {
	public static Node checkBSTValidity(Node rootNode) {
		// Your code here (remove the placeholder line below)
    return checkBSTValidityHelp(rootNode, null, null);
	}

	private static Node checkBSTValidityHelp(Node node, Integer min, Integer max){
	  if(node == null){
	    return null;
	  }

	  if((min != null && node.key <= min) || (max != null && node.key >= max)){
	    return node;
	  }

	  Node leftViolation = checkBSTValidityHelp(node.left, min, node.key);
	  if(leftViolation != null){
	    return leftViolation;
	  }

	  Node rightViolation = checkBSTValidityHelp(node.right, node.key, max);
	  if(rightViolation != null){
```

```
            return rightViolation;
          }

          return null;
        }
}
```

Lab 8:

```
public class ExtendedAVLNode extends AVLNode {
  private int subtreeKeyCount;

  public ExtendedAVLNode(int key){
    super(key);
    subtreeKeyCount = 1;
  }

        @Override
        public int getSubtreeKeyCount() {
                return subtreeKeyCount;
        }

        public void updateSubtreeKeyCount(){
          int leftCount = 0;
          int rightCount = 0;

          if(getLeft() instanceof ExtendedAVLNode){
            leftCount = ((ExtendedAVLNode) getLeft()).getSubtreeKeyCount();
          }
```

```
        if (getRight() instanceof ExtendedAVLNode){
          rightCount = ((ExtendedAVLNode) getRight()).getSubtreeKeyCount();
        }

        subtreeKeyCount = 1 + leftCount + rightCount;
      }
}
```

```
public class ExtendedAVLTree extends AVLTree {
      // Each node in an ExtendedAVLTree is an ExtendedAVLNode
      @Override
      protected BSTNode makeNewNode(int key) {
            return new ExtendedAVLNode(key);
      }

      // Your code here
      public void insertNode(BSTNode node){
        super.insertNode(node);
        updateAllSubtreeCounts((ExtendedAVLNode) getRoot());
      }

      public boolean removeNode(BSTNode node){
        boolean result = super.removeNode(node);
```

```
    updateAllSubtreeCounts((ExtendedAVLNode) getRoot());
    return result;
  }

  private void updateAllSubtreeCounts(ExtendedAVLNode node){
    if(node == null){
      return;
    }

    if (node.getLeft() instanceof ExtendedAVLNode){
      updateAllSubtreeCounts((ExtendedAVLNode) node.getLeft());
    }

    if (node.getRight() instanceof ExtendedAVLNode){
      updateAllSubtreeCounts((ExtendedAVLNode) node.getRight());
    }

    node.updateSubtreeKeyCount();
  }

  protected BSTNode rotateLeft(BSTNode node){
    BSTNode result = super.rotateLeft(node);

    if(node instanceof ExtendedAVLNode){
      ((ExtendedAVLNode) node).updateSubtreeKeyCount();
    }

    if (node.getParent() instanceof ExtendedAVLNode){
```

```
            ((ExtendedAVLNode) node.getParent()).updateSubtreeKeyCount();
        }

        return result;
}

  protected BSTNode rotateRight(BSTNode node){
    BSTNode result = super.rotateRight(node);

    if (node instanceof ExtendedAVLNode) {
        ((ExtendedAVLNode) node).updateSubtreeKeyCount();
    }

    if (node.getParent() instanceof ExtendedAVLNode) {
        ((ExtendedAVLNode) node.getParent()).updateSubtreeKeyCount();
    }

    return result;
  }

        @Override
        public int getNthKey(int n) {
          // Your code here (remove placeholder line below)
                return getNthKeyHelp((ExtendedAVLNode) getRoot(), n);
        }

        private int getNthKeyHelp(ExtendedAVLNode node, int n){
          if (node == null) {
```

```
            throw new IllegalArgumentException("Invalid value for n.");
        }

        int leftCount = 0;
        if(node.getLeft() instanceof ExtendedAVLNode) {
            leftCount = ((ExtendedAVLNode) node.getLeft()).getSubtreeKeyCount();
        }

        if (n < leftCount) {
            return getNthKeyHelp((ExtendedAVLNode) node.getLeft(), n);
        } else if (n == leftCount) {
            return node.getKey();
        } else {
            return getNthKeyHelp((ExtendedAVLNode) node.getRight(), n - leftCount - 1);
        }
            }

}
```