NAME:__________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Max</th>
<th>Obtained</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
20 points **Problem 1:** Based on the implementations that we have explored in class, give the best Big-O running time (i.e., the tightest upper bound possible) for each data structure and each operation.

(a) [5 points] What is the Big-O runtime (e.g., O(1), O(log n), or O(n)) for **searching** for an element in each **sequential data structure** with n elements?
   - ArrayList: \( O(n) \)
   - LinkedList: \( O(n) \)
   - OrderedArrayList: \( O(\log n) - \text{use binary search} \)
   - OrderedLinkedList: \( O(n) - \text{no dynamic access to elements} \)

(b) [5 points] What is the best Big-O runtime (e.g., O(1), O(log n), or O(n)) for **inserting** an element into each **sequential data structure** with n elements?
   - ArrayList: \( O(1) \)
   - LinkedList: \( O(1) \)
   - OrderedArrayList: \( O(n) - \text{have to shift elements} \)
   - OrderedLinkedList: \( O(n) - \text{have to find element first (see 1.a)} \)

(c) [5 points] What is the best Big-O runtime (e.g., O(1), O(log n), O(h), or O(n)) for **searching** for an element in each **hierarchical data structure** with n elements and tree height h?
   - BinaryTree: \( O(n) \)
   - BinarySearchTree: \( O(h) \)
   - AVL Tree: \( O(\log n) - \text{balanced tree} \)
   - (2-4) Tree: \( O(\log n) - \text{balanced tree} \)

(d) [5 points] What is the best Big-O runtime (e.g., O(1), O(log n), O(h), or O(n)) for **inserting** an element into each **hierarchical data structure** with n elements and tree height h?
   - BinaryTree: \( O(h) - \text{have to insert at a leaf which may be h away from root} \)
   - BinarySearchTree: \( O(h) \)
   - AVL Tree: \( O(\log n) - \text{balanced tree} \)
   - (2-4) Tree: \( O(\log n) - \text{balanced tree} \)
Problem 2: A portion of a simplified version of the BinarySearchTree class is shown below. Unlike our full implementation, each instance of a Position stores an element that is an integer, as opposed to a key-value pair of generics (e.g., <K,V>). Sample code for the simplified BinaryTree interface and Position class are shown on Page 9. (You may tear off the page containing this source code so you can examine the code side by side.)

(a) [20 points] Implement the treeRange() method below so that it runs as fast as possible in terms of asymptotic (Big-O) time. (You may not add additional instance variables or methods to the BinarySearchTree class.)

(b) [4 points] What is the asymptotic run-time of your implementation? Briefly explain. - O(h) runtime

public class BinarySearchTree implements BinaryTree{
    // Instance variables:
    protected Position root;
    protected Comparator<Integer> C;
    ...
    // Class Methods
    public Position find(int key){
        Position curPos = treeSearch(key, root());
        if (isInternal(curPos)) return curPos;
        return null;
    }
    protected Position treeSearch(K key, Position pos) {
        if (isExternal(pos)) return pos; // key not found; return external node
        else {
            int curKey = pos.element();
            int comp = C.compare(key, curKey);
            if (comp < 0)
                return treeSearch(key, left(pos)); // search left subtree
            else if (comp > 0)
                return treeSearch(key, right(pos)); // search right subtree
            return pos; // return internal node where key is found
        }
    }
    ...
    /** treeRange() returns the difference between the maximum element and
    * minimum element in the tree. For example, if the minimum element is 2
    * and the maximum element is 7, then treeRange() returns 5
    ***/
    public int treeRange() {
        if (isEmpty()) return 0;
        Position minPos = root();
        Position maxPos = root();
        while(!isExternal(left(minPos))) minPos = left(minPos);
        while(!isExternal(right(maxPos))) maxPos = left(maxPos);
        return maxPos.element() - minPos.element();
    }
}
} // end of treeRange() method
} // end of class
26 points Problem 3: Binary Search Trees vs. AVL Binary Search Trees.

(a) [2 points] What property makes AVL trees different from generic binary search trees? Please describe this property.
   Height-Balance Property: the height of each node’s children differs by at most 1.

(b) [6 points] Draw an example of a binary search tree that is not a valid AVL tree.
   The left three in the solution of 3.d is an example of an unbalance AVL tree since the root’s left child has a height of 2 and the root’s right child has a height of 4.

(c) [6 points] For the AVL tree shown below, draw the resulting tree after inserting 32 into the following tree. Please re-draw the tree each time the tree is restructured.
(d) [6 points] Using the resulting AVL tree from Part c, draw the resulting tree after removing 15. Please re-draw the tree each time the tree is restructured. (Tip: it will be helpful to redraw your solution from the previous page.)

(e) [3 points] What is the maximum number of times that the tree will have to be restructured after an **insertion**?

\[ O(1) \] - one tree restructuring restores the **global** height-balance property. See Section 10.2 in the textbook for details.

(f) [3 points] What is the maximum number of times that the tree will have to be restructured after a **removal**?

\[ O(\log n) \] - one tree restructuring restores the **local** height-balance property. See Section 10.2 in the textbook for details.
Problem 4: Data structures to support an Interactive Election Map

Background

On election night, millions of American will be glued to their televisions as election results pour in from around the nation. You have been hired by CNBC to develop visualization software that will display an interactive election map of the country.

The color of each state on the national map will be bright orange if candidate McBama has been declared to be the winner, or bright purple if candidate Ocain has been declared winner. If no candidate has been declared winner, then the state will be shaded orange or purple according to the percentage of reported votes for each candidate. That is, the intensity of the color will increase as one candidate has a bigger percentage of votes. If there is no reported information or if there is a tie, the state will be colored gray.

In addition, each state also has an integer number of “electoral” votes associated with it. For example, Pennsylvania has 21 electoral votes (out of a total of 538 electoral votes).

The winner of the election is the candidate who wins more than half of the electoral votes. A candidate gets all of the electoral votes from a state if that candidate wins the popular vote in that state. The popular vote for a state is determined by summing the popular votes in each county. Over the course of election night, the Associated Press reports the vote count for each county in each state as they become available. Often a candidate will be declared the winner of a state before all of the votes are counted if experts determine that one candidate has an insurmountable lead.

Question

Your job is to design a set of classes to be able to support the functionality of the interactive CNBC election maps. For example, you will need to be able to:

- store (and update) the vote counts for each county as they are reported by the Associated Press
- quickly report the total number of votes for each candidate by county, by state and for the entire nation
- mark a state as "won" if the experts determine that a particular candidate has won the popular vote for the state
- display the total number of electoral votes for each candidate
- declare a candidate as the victor of the national election

Please describe your list of classes. For each class, you should describe both the instance variables and the important public methods. You will be graded on the simplicity and elegance of your object-oriented design.
There can be many "good" solutions to this design question. Here is one that would receive full credit.

Class / Instance Variables / Methods

Nation / list of candidates, list of states / sumElectoralVotes(Candidate), sumPopularVote(Candidate), winningCandidate()

State / name, number of electoral votes, winning candidate, list of counties / sumPopularVote(Candidate), setWinningCandidate(Candidate) getWinningCandidate(), getElectorCount(),

County / name, list of <candidates, number of votes>/ setVotes(Candidate, count), getVotes(Candidate),

Candidate / name / getName()
Problem 5 (Bonus): Prove that the product of two odd integers is odd.

(Hint: use the definition of an odd number - a integer $x$ is odd iff there exist another integer $i$ such that $i = 2 \cdot k + 1$.)

Let $x, y$ be odd integers. Then there exists two integers $i, j$ such that $x = 2i + 1$ and $y = 2j + 1$.

\[
xy = (2i + 1)(2j + 1) = 4ij + 2i + 2j + 1 = 2(2ij + i + j) + 1
\]

Since $2ij + i + j$ is an integer, $xy$ is an odd integer.
Code for Problem 2:

public interface BinaryTree {

    /** Returns root of the tree **/ 
    public Position<E> root();

    /** Returns the left child of a node. */
    public Position<E> left(Position<E> v);

    /** Returns the right child of a node. */
    public Position<E> right(Position<E> v);

    /** Returns whether a node has a left child. */
    public boolean hasLeft(Position<E> v);

    /** Returns whether a node has a right child. */
    public boolean hasRight(Position<E> v);

}

public class Position{
    private int element; // element stored at this node
    private Position left, right, parent; // adjacent nodes

    public int element(){ return element;}
    public void setElement(int newElement){element = newElement;}

}