CSE 143, Winter 2012
Final Exam Key

1.

Statement	Output
`var1.a();`	`Kurt 1 / Jerry 3 / Kurt 3`
`var1.c();`	`Jerry 3 / Kurt 3`
`var2.a();`	`Kurt 1 /Chris 3`
`var2.b();`	`Chris 2 / Jerry 3 / Kurt 3`
`var3.a();`	`Kurt 1 /Eddie 3`
`var3.b();`	`error`
`var4.a();`	`Kurt 1 /Chris 3`
`var5.a();`	`error`
`((Chris) var5).a();`	`error`
`((Jerry) var1).a();`	`Kurt 1 / Jerry 3 / Kurt 3`
`((Jerry) var4).a();`	`Kurt 1 / Chris 3`
`((Chris) var3).b();`	`error`
`((Eddie) var3).b();`	`Kurt 1 / Eddie 3 / Eddie 2`
`((Jerry) var4).c();`	`Chris 3`
`((Kurt) var5).c();`	`Jerry 3 / Kurt 3`

2.

```
public class RiggedDice extends Dice implements Comparable<RiggedDice> {
    private int min;

    public RiggedDice(int count, int min) {
        super(count);   // OK for super() to be after exception test
        if (min < 0 || min >= 6) {
            throw new IllegalArgumentException();
        }
        this.min = min;
    }

    public int compareTo(RiggedDice other) {
        int myTotal = total();
        int hisTotal = other.total();
        if (myTotal != hisTotal) {
            return myTotal - hisTotal;
        } else {
            return min - other.min;
        }
    }

    public int getMin() {
        return min;
    }

    public void roll(int index) {
        super.roll(index);
        while (getValue(index) < min) {    // do/while is okay
            super.roll(index);
        }
    }

    public int total() {
        return super.total() + 1;
    }

    public String toString() {
        return "rigged " + super.toString() + " min " + min;
    }
}
```

3.

{cinq=five, deux=two, four=quatre, one=un, three=trois}

{board=skate, car=drive, computer=play}

{begin=end, boy=girl, ebert=siskel, first=last, heads=tails}

{cotton=rain, light=tree, seed=tree, tree=violin}

4.

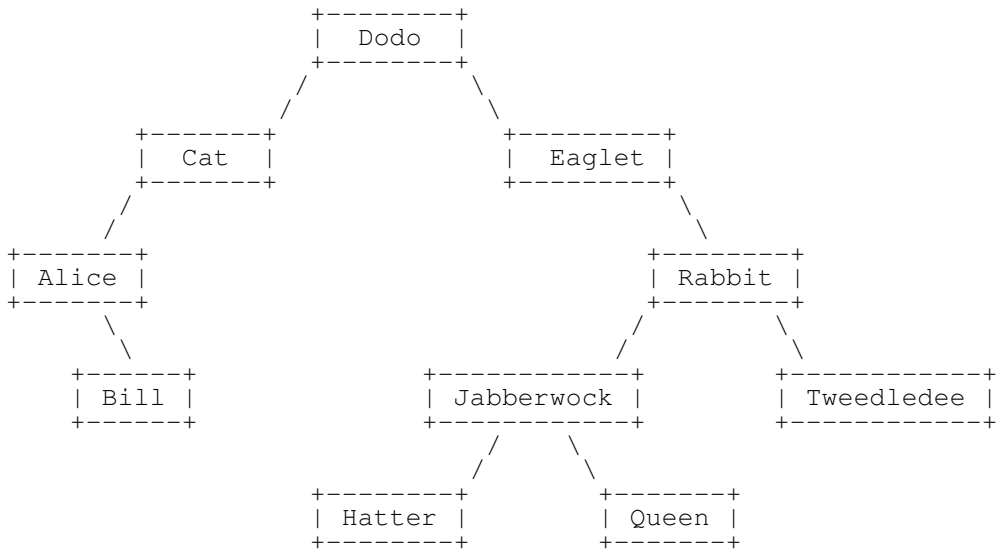
```
public void sortPairs() {
    if (front != null && front.next != null) {
        // special case if we need to swap the first two elements
        if (front.data > front.next.data) {
            ListNode second = front.next;
            front.next = front.next.next;
            second.next = front;
            front = second;
        }

        ListNode current = front.next;
        while (current.next != null && current.next.next != null) {
            if (current.next.data > current.next.next.data) {
                ListNode second = current.next.next;
                current.next.next = current.next.next.next;
                second.next = current.next;
                current.next = second;
            }
            current = current.next.next;
        }
    }
}

public void sortPairs() {
    if (front == null || front.next == null) {
        return;
    }

    ListNode current = front;
    if (front.data > front.next.data) {
        front = front.next;
        current.next = front.next;
        front.next = current;
    }

    current = front.next;
    while (current.next != null && current.next.next != null) {
        if (current.next.data <= current.next.next.data) {   // < is okay
            current = current.next.next;
        } else {
            ListNode temp = current.next;
            current.next = current.next.next;
            temp.next = current.next.next;
            current.next.next = temp;
            current = temp;
        }
    }
}
// recursive
public void sortPairs() {
    front = sortPairs(front);
}
private ListNode sortPairs(ListNode current) {
    if (current != null && current.next != null) {
        if (current.data > current.next.data) {
            ListNode second = current.next;
            current.next = current.next.next;
            second.next = current;
            current = second;
        }
        current.next.next = sortPairs(current.next.next);
    }

    return current;
}
```

5.
(a) visited: 6, 2, 4, 5
returned: -7

(b) 4: {-4, 17, 3, 94, 46, 8, 29, 12}

(c) 2: merge sort achieves an O(N log N) runtime by dividing the array in half at each step and then recursively sorting and merging the halves back together.

6. (a)



(b)

Pre: Dodo, Cat, Alice, Bill, Eaglet, Rabbit, Jabberwock, Hatter, Queen, Tweedledee
In: Alice, Bill, Cat, Dodo, Eaglet, Hatter, Jabberwock, Queen, Rabbit, Tweedledee
Post: Bill, Alice, Cat, Hatter, Queen, Jabberwock, Tweedledee, Rabbit, Eaglet, Dodo

7.

```
// counting up (passing current level)
public void swapChildrenAtLevel(int n) {
    if (n <= 0) {
        throw new IllegalArgumentException();
    }
    swapChildrenAtLevel(n, 1, overallRoot);
}

private void swapChildrenAtLevel(int n, int currentLevel, IntTreeNode root) {
    if (root != null) {
        if (currentLevel == n) {
            IntTreeNode temp = root.left;
            root.left = root.right;
            root.right = temp;
        } else if (currentLevel < n) {
            swapChildrenAtLevel(n, currentLevel + 1, root.left);
            swapChildrenAtLevel(n, currentLevel + 1, root.right);
        }
    }
}

// counting down
public void swapChildrenAtLevel(int n) {
    if (n <= 0) throw new IllegalArgumentException();
    swapChildrenAtLevel(n, overallRoot);
}

private void swapChildrenAtLevel(int n, IntTreeNode root) {
    if (root != null) {
        if (n == 1) {
            IntTreeNode temp = root.left;
            root.left = root.right;
            root.right = temp;
        } else if (n > 0) {
            swapChildrenAtLevel(n - 1, root.left);
            swapChildrenAtLevel(n - 1, root.right);
        }
    }
}

// useless x=change(x)
public void swapChildrenAtLevel(int n) {
    if (n <= 0) throw new IllegalArgumentException();
    overallRoot = swapChildrenAtLevel(overallRoot, 1, n);
}

private IntTreeNode swapChildrenAtLevel(IntTreeNode root, int level, int n) {
    if (root == null) {
        return null;
    } else {
        if (level < n) {
            root.left  = swapChildrenAtLevel(root.left,  level + 1, n);
            root.right = swapChildrenAtLevel(root.right, level + 1, n);
        } else {
            IntTreeNode temp = root.left;
            root.left = root.right;
            root.right = temp;
        }
        return root;
    }
}
```

8.

```
// "count up" solution
public void makePerfect() {
    int height = height(overallRoot);
    overallRoot = makePerfect(overallRoot, 1, height);
}

private IntTreeNode makePerfect(IntTreeNode root, int level, int height) {
    if (level <= height) {
        if (root == null) {
            root = new IntTreeNode(0);
        }
        root.left = makePerfect(root.left, level + 1, height);
        root.right = makePerfect(root.right, level + 1, height);
    }
    return root;
}

// "count down" solution
public void makePerfect() {
    overallRoot = makePerfect(overallRoot, height(overallRoot));
}

private IntTreeNode makePerfect(IntTreeNode root, int h) {
    if (h > 0) {
        if (root == null) {
            root = new IntTreeNode(0);
        }
        root.left = makePerfect(root.left, h - 1);
        root.right = makePerfect(root.right, h - 1);
    }
    return root;
}

// "look-ahead" bad x=change(x) solution (NOT full credit; -5)
public void makePerfect() {
    if (overallRoot != null) {                  // -2 if not checking for null here
(most don't)
        int height = height(overallRoot);
        overallRoot = makePerfect(overallRoot, 1, height);
    }
}

private IntTreeNode makePerfect(IntTreeNode root, int level, int height) {
    if (level < height) {                       // <-- note < instead of <= here
        if (root.left == null) {                // BAD; look-ahead
            root.left = new IntTreeNode(0);
        }
        if (root.right == null) {               // BAD; look-ahead
            root.right = new IntTreeNode(0);
        }
        root.left = makePerfect(root.left, level + 1, height);
        root.right = makePerfect(root.right, level + 1, height);
    }
    return root;
}
```