

CSE 143 Sample Final Exam #2 Key

1.

<u>Statement</u>	<u>Output</u>
var1.method2();	Ocean 2
var2.method2();	Pond 2
var3.method2();	Pond 2
var4.method2();	error
var5.method2();	Bay 2
var6.method2();	Ocean 2
var1.method3();	Lake 3/Ocean 2
var2.method3();	error
var3.method3();	error
var4.method3();	error
var5.method3();	Lake 3/Bay 2
var6.method3();	Lake 3/Ocean 2
((Ocean) var5).method1();	error
((Lake) var3).method3();	Lake 3/Pond 2
((Lake) var4).method1();	error
((Ocean) var1).method1();	Bay 1/Pond 2
((Bay) var4).method1();	Bay 1/Pond 2
((Lake) var2).method3();	error
((Ocean) var5).method1();	error
((Pond) var4).method2();	Bay 2

2.

```
public class CalendarDate extends Date implements Comparable<CalendarDate> {
    private int year;

    public CalendarDate(int year, int month, int day) {
        super(month, day);
        this.year = year;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public String toString() {
        return year + "/" + super.toString();
    }

    public void nextDay() {
        super.nextDay();
        if (getMonth() == 1 && getDay() == 1) {
            year++; // wrap year
        }
    }

    public int compareTo(CalendarDate other) {
        if (year != other.year) {
            return year - other.year;
        } else if (getMonth() != other.getMonth()) {
            return getMonth() - other.getMonth();
        } else {
            return getDay() - other.getDay();
        }
    }
}
```

3. Two possible solutions appear below. The second is shorter because it uses recursion.

```
// precondition: this list and other list are sorted
public void removeAll(LinkedList other) {
    if (front != null && other.front != null) {
        ListNode current2 = other.front;
        while (current2 != null && front != null &&
            current2.data <= front.data) {
            if (current2.data < front.data) {
                current2 = current2.next;
            } else { // current2.data == front.data
                front = front.next;
            }
        }
        if (front != null) {
            ListNode current1 = front;
            while (current1.next != null && current2 != null) {
                if (current1.next.data < current2.data) {
                    current1 = current1.next;
                } else if (current1.next.data == current2.data) {
                    current1.next = current1.next.next;
                } else { // current1.next.data > current2.data
                    current2 = current2.next;
                }
            }
        }
    }
}

// precondition: this list and other list are sorted
public void removeAll(LinkedList other) {
    front = removeAllHelper(front, other.front);
}

private ListNode removeAllHelper(ListNode front1, ListNode front2) {
    if (front1 == null || front2 == null) {
        return front1;
    } else if (front1.data < front2.data) {
        front1.next = removeAllHelper(front1.next, front2);
    } else if (front1.data == front2.data) {
        front1 = removeAllHelper(front1.next, front2);
    } else { // front1.data > front2.data
        front1 = removeAllHelper(front1, front2.next);
    }
    return front1;
}
```

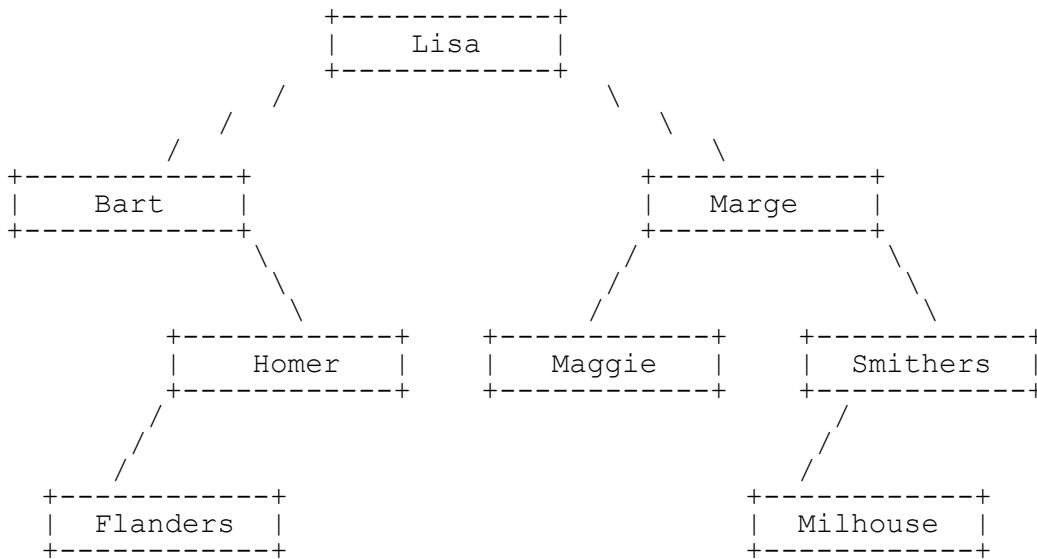
4.

(a) Indexes examined: 7, 11, 13, 12 Value returned: -14

(b) **{-9, 5, 8, 14, 0, -1, -7, 3}**
{-9, -7, 8, 14, 0, -1, 5, 3}
{-9, -7, -1, 14, 0, 8, 5, 3}

(c) { 8, 5, -9, 14, 0, -1, -7, 3}
{ 8, 5, -9, 14} { 0, -1, -7, 3} split
{ 8, 5} {-9, 14} { 0, -1} {-7, 3} split
{ 8} { 5} {-9} {14} { 0} {-1} {-7} { 3} split
{ 5, 8} {-9, 14} {-1, 0} {-7, 3} merge
{-9, 5, 8, 14} {-7, -1, 0, 3} merge
{-9, -7, -1, 0, 3, 5, 8, 14} merge

5. (a)



(b)

Pre-order: Lisa, Bart, Homer, Flanders, Marge, Maggie, Smithers, Milhouse

In-order: Bart, Flanders, Homer, Lisa, Maggie, Marge, Milhouse, Smithers

Post-order: Flanders, Homer, Bart, Maggie, Milhouse, Smithers, Marge, Lisa

6.

```
public int countMultiples(int value) {
    if (value == 0) {
        throw new IllegalArgumentException();
    }
    return countMultiples(overallRoot, value);
}

private int countMultiples(IntTreeNode root, int value) {
    if (root == null) {
        return 0;
    } else {
        int sum = countMultiples(root.left, value) +
                  countMultiples(root.right, value);
        if (root.data % value == 0) {
            return 1 + sum;
        } else {
            return sum;
        }
    }
}
```

7.

```
public int matches(IntTree other) {
    return matches(overallRoot, other.overallRoot);
}

private int matches(IntTreeNode root1, IntTreeNode root2) {
    if (root1 == null || root2 == null) {
        return 0;
    } else {
        int sum = matches(root1.left, root2.left) +
                  matches(root1.right, root2.right);
        if (root1.data == root2.data) {
            return 1 + sum;
        } else {
            return sum;
        }
    }
}
```