CSE 143 Sample Final Exam #3 Key

1.

Statement	Output
`var1.method1();`	`Raph 1`
`var1.method2();`	`error`
`var1.method3();`	`error`
`var2.method1();`	`Leo 1`
`var2.method2();`	`Mike 2/Leo 1/Don 2`
`var2.method3();`	`Leo 3/Leo 1`
`var3.method1();`	`error`
`var4.method1();`	`Leo 1`
`var4.method2();`	`Leo 1/Don 2`
`var4.method3();`	`error`
`((Don) var1).method2();`	`Raph 1/Don 2`
`((Mike) var2).method2();`	`Mike 2/Leo 1/Don 2`
`((Raph) var3).method1();`	`Raph 1`
`((Don) var3).method2();`	`error`
`((Leo) var4).method3();`	`Leo 3/Leo 1`

2.

```
public class Playa extends Person implements Comparable<Playa> {
    private Set<Person> fiancees;

    public Playa(String name) {
        super(name);
        fiancees = new HashSet<Person>();
    }

    public void engageTo(Person other) {
        super.engageTo(other);
        if (other == null) {
            fiancees.clear();
        } else {
            // if this were a List, I would need to call contains to avoid duplicates
            fiancees.add(other);
        }
    }

    public boolean isSingle() {
        return countFiancees() == 0;
    }

    public int countFiancees() {
        return fiancees.size();
    }

    public int compareTo(Playa other) {
        if (countFiancees() != other.countFiancees()) {
            return countFiancees() - other.countFiancees();
        } else {
            return getName().compareTo(other.getName());
        }
    }
}
```

3.

```
public void expand(int factor) {
    if (factor <= 0) {
        front = null;
    } else {
        ListNode current = front;
        while (current != null) {
            current.data /= factor;
            for (int i = 1; i < factor; i++) {
                current.next = new ListNode(current.data, current.next);
                current = current.next;
            }
            current = current.next;
        }
    }
}
```

4.

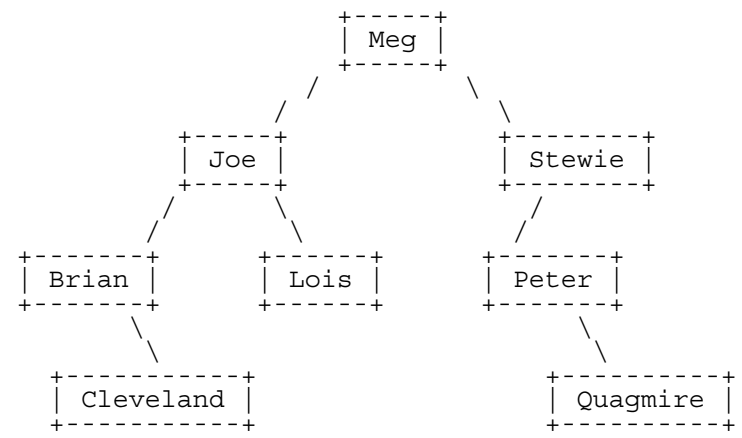
(a) Indexes examined: 6, 2, 4, 3 Value returned: -4

(b) {**9**, **63**, 45, 72, 27, 18, 54, 36}
{9, **18**, 45, 72, 27, **63**, 54, 36}
{9, 18, **27**, 72, **45**, 63, 54, 36}

(c)

{63, 9, 45, 72, 27, 18, 54, 36}	
{63, 9, 45, 72} {27, 18, 54, 36}	split
{63, 9} {45, 72} {27, 18} {54, 36}	split
{63} {9} {45} {72} {27} {18} {54} {36}	split
{ 9, 63} {45, 72} {18, 27} {36, 54}	merge
{ 9, 45, 63, 72} {18, 27, 36, 54}	merge
{ 9, 18, 27, 36, 45, 54, 63, 72}	merge

5. (a)



(b)

Pre-order: Meg, Joe, Brian, Cleveland, Lois, Stewie, Peter, Quagmire

In-order: Brian, Cleveland, Joe, Lois, Meg, Peter, Quagmire, Stewie

Post-order: Cleveland, Brian, Lois, Joe, Quagmire, Peter, Stewie, Meg

6. Two solutions are shown.

```
public int nodesAtLevels(int min, int max) {
    if (min < 0 || min > max) {
        throw new IllegalArgumentException();
    }
    return nodesAtLevels(overallRoot, 1, min, max);
}

private int nodesAtLevels(IntTreeNode root, int level, int min, int max) {
    if (root == null || level > max) {
        return 0;
    } else if (level < min) {
        return nodesAtLevels(root.left, level + 1, min, max)
                + nodesAtLevels(root.right, level + 1, min, max);
    } else {
        return 1 + nodesAtLevels(root.left, level + 1, min, max)
                + nodesAtLevels(root.right, level + 1, min, max);
    }
}
```

```
public int nodesAtLevels(int min, int max) {
    if (min < 0 || min > max) {
        throw new IllegalArgumentException();
    }
    return nodesAtLevels(overallRoot, min, max);
}

private int nodesAtLevels(IntTreeNode root, int min, int max) {
    if (root == null) {
        return 0;
    } else if (min > 0) {
        return nodesAtLevels(root.left, min - 1, max - 1)
                + nodesAtLevels(root.right, min - 1, max - 1);
    } else if (max > 0) {
        return 1 + nodesAtLevels(root.left, min - 1, max - 1)
                + nodesAtLevels(root.right, min - 1, max - 1);
    }
}
```

7.

```
public void trim(int min, int max) {
    overallRoot = trim(overallRoot, min, max);
}

private IntTreeNode trim(IntTreeNode root, int min, int max) {
    if (root != null) {
        if (root.data < min) {
            root = trim(root.right, min, max);
        } else if (root.data > max) {
            root = trim(root.left, min, max);
        } else {
            root.left  = trim(root.left,  min, max);
            root.right = trim(root.right, min, max);
        }
    }
    return root;
}
```