Solution to CSE143X Section #20 Problems

1. Preorder traversal 4, 2, 7, 9, 3, 6, 1, 8, 5, 0
   Inorder traversal   7, 9, 2, 3, 6, 4, 8, 1, 0, 5
   Postorder traversal 9, 7, 6, 3, 2, 8, 0, 5, 1, 4

2. One possible solution appears below.

```
public void writeSquares(int n) {
    if (n < 1) {
        throw new IllegalArgumentException();
    }

    if (n == 1) {
        System.out.print(1);
    } else if (n % 2 == 1) {
        System.out.print(n * n + ", ");
        writeSquares(n - 1);
    } else {
        writeSquares(n - 1);
        System.out.print(", " + n * n);
    }
}
```

3.

Statement	Output
var1.method1();	compiler error
var2.method1();	Rain 1
var1.method2();	Sleet 2/Snow 2/Sleet 3
var2.method2();	Rain 2
var3.method2();	Sleet 2/Snow 2/Fog 3
var4.method2();	compiler error
var5.method2();	Sleet 2/Snow 2/Fog 3
var1.method3();	Sleet 3
var2.method3();	Snow 3
var3.method3();	Fog 3
var4.method3();	compiler error
var5.method3();	Fog 3
((Rain)var4).method1();	runtime error
((Fog)var5).method1();	Fog 1
((Sleet)var3).method1();	compiler error
((Sleet)var3).method3();	Fog 3
((Fog)var6).method3();	runtime error
((Snow)var4).method2();	Snow 2
((Sleet)var4).method3();	runtime error
((Rain)var6).method3();	Snow 3

4. Two possible solutions appear below.

```
public Set<Point> extractEqual(Set<Point> data) {
    Set<Point> result = new HashSet<>();
    for (Point p : data) {
        if (p.getX() == p.getY()) {
            result.add(p);
        }
    }
    return result;
}
```

```
    public Set<Point> extractEqual(Set<Point> data) {
        Set<Point> result = new HashSet<>();
        Iterator<Point> i = data.iterator();
        while (i.hasNext()) {
            Point p = i.next();
            if (p.getX() == p.getY()) {
                result.add(p);
            }
        }
        return result;
    }
```

5. One possible solution appears below.

```
    public static int parityMatches(Stack<Integer> s1, Stack<Integer> s2) {
        if (s1.size() != s2.size()) {
            throw new IllegalArgumentException();
        }
        Queue<Integer> q = new LinkedList<>();
        int count = 0;
        while (!s1.isEmpty()) {
            int num1 = s1.pop();
            int num2 = s2.pop();
            if (num1 % 2 == num2 % 2) {
                count++;
            }
            q.add(num1);
            q.add(num2);
        }
        while (!q.isEmpty()) {
            s1.push(q.remove());
        }
        while (!s1.isEmpty()) {
            q.add(s1.pop());
        }
        while (!q.isEmpty()) {
            s2.push(q.remove());
            s1.push(q.remove());
        }
        return count;
    }
```

6. One possible solution appears below.

```
    public int purebredOdds() {
        return purebredOdds(overallRoot);
    }

    private int purebredOdds(IntTreeNode root) {
        if (root == null || root.data % 2 == 0) {
            return 0;
        } else {
            return 1 + purebredOdds(root.left) + purebredOdds(root.right);
        }
    }
```

7. One possible solution appears below.

```
public Map<String, List<Integer>> indexMap(List<String> data) {
    Map<String, List<Integer>> result = new TreeMap<>();
    for (int i = 0; i < data.size(); i++) {
        String next = data.get(i);
        if (!result.containsKey(next)) {
            result.put(next, new ArrayList<>());
        }
        result.get(next).add(i);
    }
    return result;
}
```

8. One possible solution appears below.

```
class BookData implements Comparable<BookData> {
    private String title;
    private int reviews;
    private double total;

    public BookData(String title) {
        this.title = title;
        this.reviews = 0;
        this.total = 0.0;
    }

    public void review(double rating) {
        reviews++;
        total += rating;
    }

    public double getRating() {
        return total / reviews;
    }

    public String toString() {
        return title + " " + getRating() + " (reviews: " + reviews + ")";
    }

    public int compareTo(BookData other) {
        if (getRating() > other.getRating()) {
            return -1;
        } else if (getRating() < other.getRating()) {
            return 1;
        } else {
            return other.reviews - reviews;
        }
    }
}
```

9. One possible solution appears below.

```
    public void stretch() {
        overallRoot = stretch(overallRoot);
    }

    private IntTreeNode stretch(IntTreeNode root) {
        if (root != null) {
            root.left = stretch(root.left);
            root.right = stretch(root.right);
            if (root.left != null && root.right == null) {
                root = new IntTreeNode(1, root, null);
            } else if (root.left == null && root.right != null) {
                root = new IntTreeNode(1, null, root);
            }
        }
        return root;
    }
```

10. One possible solution appears below.

```
    public void mergeFrom(LinkedIntList other) {
        if (other.front != null) {
            if (front == null) {
                front = other.front;
            } else {
                ListNode current1 = front;
                ListNode current2 = other.front;
                 if (front.data <= other.front.data) {
                    current1 = current1.next;
                } else {
                    front = other.front;
                    current2 = current2.next;
                }
                ListNode current3 = front;
                while (current1 != null && current2 != null) {
                    if (current1.data <= current2.data) {
                        current3.next = current1;
                        current1 = current1.next;
                    } else {
                        current3.next = current2;
                        current2 = current2.next;
                    }
                    current3 = current3.next;
                }
                if (current1 != null) {
                    current3.next = current1;
                } else {
                    current3.next = current2;
                }
            }
            other.front = null;
        }
    }
```