Solution to CSE143X Section #19 Problems

1.

Method Call	Value Returned
mystery1(1)	[[0]]
mystery1(2)	[[0, 1, 2], [1, 2, 3]]
mystery1(3)	[[0, 1, 2, 3, 4], [1, 2, 3, 4, 5], [2, 3, 4, 5, 6]]
mystery1(4)	[[0, 1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6, 7], [2, 3, 4, 5, 6, 7, 8], [3, 4, 5, 6, 7, 8, 9]]

2.

Method Call	Output Produced
mystery2(grid, 0, 2, 2);	10 11
mystery2(grid, 2, 3, 2);	12 9 12
mystery2(grid, 1, 4, 1);	4 8 2 3

3.

Two-Dimensional Array	Contents of Set Returned
[[1, 2], [3, 4]]	[1, 2, 13, 14]
[[7], [], [8, 8, 9, 10]]	[7, 28, 29, 30]
[[3, 14], [5, 13, 4], [4, 3, 1]]	[3, 14, 15, 21, 23, 24]

4.

Method Call	Contents of Set Returned
mystery4(grid, 2, 2)	[6, 7]
mystery4(grid, 0, 2)	[1, 2, 5, 8]
mystery4(grid, 3, 3)	[1, 2, 3, 7, 9]

5. One possible solution appears below.

```
public void recordGrade(Map<String, Map<String, Double>> grades,
                        String id, double grade, String course) {
    if (!grades.containsKey(id)) {
        grades.put(id, new TreeMap<>());
    }
    Map<String, Double> next = grades.get(id);
    if (next.containsKey(course)) {
        grade = Math.max(grade, next.get(course));
    }
    next.put(course, grade);
}
```

6. One possible solution appears below.

```
public Set<Point> removePoints(Map<Integer, List<Point>> points, int n) {
    Set<Point> removed = new HashSet<>();
    if (points.containsKey(n)) {
        Iterator<Point> itr = points.get(n).iterator();
        while (itr.hasNext()) {
            Point p = itr.next();
            if (p.getX() < p.getY()) {
                itr.remove();
                removed.add(p);
            }
        }
    }
    return removed;
}
```

7. One possible solution appears below.

```
public int evenSum() {
    int sum = 0;
    boolean even = true;
    ListNode current = front;
    while (current != null) {
        if (even) {
            sum += current.data;
        }
        even = !even;
        current = current.next;
    }
    return sum;
}
```

8. One possible solution appears below.

```
public void removeDuplicates() {
    ListNode current = front;
    while (current != null) {
        ListNode current2 = current;
        while (current2.next != null) {
            if (current2.next.data == current.data) {
                current2.next = current2.next.next;
            } else {
                current2 = current2.next;
            }
        }
        current = current.next;
    }
}
```

9. Two possible solutions appear below. The second is shorter because it is written recursively.

```
public void switchPairs() {
    if (front != null && front.next != null) {
        ListNode current = front.next;
        front.next = current.next;
        current.next = front;
        front = current;
        current = current.next;
        while (current.next != null && current.next.next != null) {
            ListNode temp = current.next.next;
            current.next.next = temp.next;
            temp.next = current.next;
            current.next = temp;
            current = temp.next;
        }
    }
}
```

```
public void switchPairs() {
    front = switchPairs(front);
}
```

```
    private ListNode switchPairs(ListNode list) {
        if (list != null && list.next != null) {
            ListNode temp = list.next;
            list.next = temp.next;
            temp.next = list;
            list = temp;
            list.next.next = switchPairs(list.next.next);
        }
        return list;
    }
```

10. One possible solution appears below.

```
    public void takeSmallerFrom(LinkedIntList other) {
        if (front != null && other.front != null) {
            if (front.data > other.front.data) {
                ListNode temp = front;
                front = other.front;
                other.front = temp;
                temp = front.next;
                front.next = other.front.next;
                other.front.next = temp;
            }
            ListNode current1 = front;
            ListNode current2 = other.front;
            while (current1.next != null && current2.next != null) {
                if (current1.next.data > current2.next.data) {
                    ListNode temp = current1.next;
                    current1.next = current2.next;
                    current2.next = temp;
                    temp = current1.next.next;
                    current1.next.next = current2.next.next;
                    current2.next.next = temp;
                }
                current1 = current1.next;
                current2 = current2.next;
            }
        }
    }
```