CSE 143 Sample Midterm Exam #2 Key

1.

List	Output
(a) `[2, 6, 1, 8]`	`[1, 2, 6, 8]`
(b) `[30, 20, 10, 60, 50, 40]`	`[10, 30, 40, 20, 60, 50]`
(c) `[-4, 16, 9, 1, 64, 25, 36, 4, 49]`	`[-4, 1, 25, 4, 16, 9, 64, 36, 49]`

2.

```
public static boolean isConsecutive(ArrayList<Integer> list) {
    for (int i = 0; i < list.size() - 1; i++) {
        if (list.get(i) + 1 != list.get(i + 1)) {
            return false;
        }
    }
    return true;
}
```

3.

```
public static void interleave(Queue<Integer> q) {
    if (q.size() % 2 != 0) {
        throw new IllegalArgumentException();
    }
    Stack<Integer> s = new Stack<Integer>();
    int halfSize = q.size() / 2;
    for (int i = 0; i < halfSize; i++) {
        s.push(q.remove());
    }
    while (!s.isEmpty()) {   // s2q(s, q);
        q.add(s.pop());
    }
    for (int i = 0; i < halfSize; i++) {
        q.add(q.remove());
    }
    for (int i = 0; i < halfSize; i++) {
        s.push(q.remove());
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
        q.add(q.remove());
    }
}
```

4. Two solutions are shown.

```
public static Map<Integer, Integer> union(Map<Integer, Integer> m1,
                                          Map<Integer, Integer> m2) {
    Map<Integer, Integer> result = new TreeMap<Integer, Integer>();
    for (int key : m1.keySet()) {
        result.put(key, m1.get(key));
    }
    for (int key : m2.keySet()) {
        if (result.containsKey(key)) {
            result.put(key, result.get(key) + m2.get(key));
        } else {
            result.put(key, m2.get(key));
        }
    }
    return result;
}

public static Map<Integer, Integer> union(Map<Integer, Integer> m1,
                                          Map<Integer, Integer> m2) {
    Map<Integer, Integer> result = new TreeMap<Integer, Integer>();
    for (int key : m1.keySet()) {
        int value = m1.get(key);
        if (m2.containsKey(key)) {
            int value2 = m2.get(key);
            value += value2;
        }
        result.put(key, value);
    }
    for (int key : m2.keySet()) {
        int value = m2.get(key);
        if (!result.containsKey(key)) {
            result.put(key, value);
        }
    }
    return result;
}
```

5.

```
ListNode list2 = list.next.next;          // list2 -> 3
list.next.next.next.next = list.next;     // 4 -> 2
ListNode temp = list;                     // temp -> 1
list = list.next.next.next;               // list -> 4
list2.next = temp;                        // 3 -> 1
list.next.next = null;                    // 2 /
list2.next.next = null;                   // 1 /
```

6.

```
public void removeLast(int n) {
    if (front != null) {
        ListNode current = front;
        ListNode spot = null;
        while (current.next != null) {
            if (current.next.data == n) {
                spot = current;
            }
            current = current.next;
        }
        if (spot != null) {
            spot.next = spot.next.next;
        } else if (front.data == n) {
            front = front.next;
        }
    }
}
```

7. Two solutions are shown.

```
public class Rational implements Comparable<Rational> {
    ...

    public int compareTo(Rational other) {
        double myValue = getNumericValue();
        double otherValue = other.getNumericValue();
        if (myValue > otherValue) {
            return 1;
        } else if (myValue < otherValue) {
            return -1;
        } else if (denominator < other.getDenominator()) {
            return 1;
        } else if (denominator < other.getDenominator()) {
            return -1;
        } else {
            return 0;
        }
    }
}
```

```
public class Rational implements Comparable<Rational> {
    ...

    public int compareTo(BankAccount other) {
        if (Math.signum(getNumericValue() - other.getNumericValue()) != 0.0) {
            return (int) Math.signum(getNumericValue() - other.getNumericValue());
        } else {
            return denominator - other.getDenominator();
        }
    }
}
```

8.

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

(b) Initial array: {100, 87, 15, 92, 45, 38, 61, 20}
after 1 pass: {**15**, 87, **100**, 92, 45, 38, 61, 20}
after 2 passes: {15, **20**, 100, 92, 45, 38, 61, **87**}
after 3 passes: {15, 20, **38**, 92, 45, **100**, 61, 87}

(c)
```
{100, 87,  15, 92,  45, 38,  61, 20}
{100, 87,  15, 92} {45, 38,  61, 20}
{100, 87} {15, 92} {45, 38} {61, 20}
{100}{87} {15}{92} {45}{38} {61}{20}
{87, 100} {15, 92} {38, 45} {20, 61}
{15, 87, 92, 100}  {20, 38,  45, 61}
{15, 20, 38, 45,    61, 87, 92, 100}
```

9.

Call	Output
`mystery(3, 3);`	`=3=`
`mystery(5, 1);`	`*`
`mystery(1, 5);`	`5 4 =3= 2 1`
`mystery(2, 7);`	`7 6 5 * 4 3 2`
`mystery(1, 8);`	`8 7 6 5 * 4 3 2 1`

10.
```
public static String repeat(String s, int n) {
    if (n < 0) {
        throw new IllegalArgumentException();
    } else if (n == 0) {
        return "";
    } else if (n == 1) {
        return s;
    } else if (n % 2 == 0) {
        String temp = repeat(s, n / 2);
        return temp + temp;
        // alternative without temp: return repeat(s + s, n / 2);
    } else {
        return s + repeat(s, n - 1);
    }
}
```