CSE 143 Sample Midterm Exam #4 Key

1.

List	Output
(a) `[72, 20]`	`[72, 20]`
(b) `[1, 2, 3, 4, 5, 6]`	`[1, 20, 18, 15, 11, 6]`
(c) `[10, 20, 30, 40]`	`[10, 90, 70, 40]`

2. Two possible solutions are shown below.

```
public static void removeShorterStrings(ArrayList<String> list) {
    for (int i = 0; i < list.size() - 1; i++) {
        String first = list.get(i);
        String second = list.get(i + 1);
        if (first.length() <= second.length()) {
            list.remove(i);
        } else {
            list.remove(i + 1);
        }
    }
}
```

```
public static void removeShorterStrings(ArrayList<String> list) {
    for (int i = list.size() - 1; i > 0; i -= 2) {
        if (list.get(i - 1).length() <= list.get(i).length()) {
            list.remove(i - 1);
        } else {
            list.remove(i);
        }
    }
}
```

3.

```
public static boolean isSorted(Stack<Integer> s) {
    if (s.size() <= 1) {
        return true;
    }

    Queue<Integer> q = new LinkedList<Integer>();
    boolean isSorted = true;
    int prev = s.pop();
    q.add(prev);
    while (!s.isEmpty()) {
        int x = s.pop();
        if (x > prev) {
            isSorted = false;
        }
        q.add(x);
        prev = x;
    }

    while (!q.isEmpty()) {     // q2s(q, s);
        s.push(q.remove());
    }
    while (!s.isEmpty()) {     // s2q(s, q);
        q.add(s.pop());
    }
    while (!q.isEmpty()) {
        s.push(q.remove());    // q2s(q, s);
    }

    return isSorted;
}
```

4. Two possible solutions are shown below.

```
public static Map<String, Integer> commonCustomers(
        Map<String, Integer> tvAccounts,
        Map<String, Integer> internetAccounts) {
    Map<String, Integer> common = new HashMap<String, Integer>();
    for (String name : tvAccounts.keySet()) {
        if (internetAccounts.containsKey(name)) {
            common.put(name, tvAccounts.get(name));
        }
    }
    return common;
}

public static Map<String, Integer> commonCustomers(
        Map<String, Integer> m1, Map<String, Integer> m2) {
    Map<String, Integer> common = new HashMap<String, Integer>(m1);
    common.keySet().retainAll(m2.keySet());
    return common;
}
```

5.

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

6.

```
public void printPairsSwitched() {
    ListNode current = front;
    while (current != null && current.next != null) {
        System.out.print(current.next.data + " ");
        System.out.print(current.data + " ");
        current = current.next.next;
    }
    if (current != null) {
        System.out.print(current.data + " ");
    }
    System.out.println();
}
```

7. Two possible solutions are shown below.

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

    public int compareTo(MovieRating other) {
        if (numRatings == 0 || other.numRatings == 0) {
            return numRatings - other.numRatings;
        }

        double myRating = cumulativeRating / numRatings;
        double otherRating = other.cumulativeRating / other.numRatings;
        if (myRating < otherRating) {
            return -1;
        } else if (myRating > otherRating) {
           return 1;
        } else {
           return 0;
        }
    }
}
```

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

    public int compareTo(MovieRating other) {
        if (numRatings == 0 || other.numRatings == 0) {
            return numRatings - other.numRatings;
        } else {
            return (int) Math.signum(cumulativeRating / numRatings,
                         other.cumulativeRating / other.numRatings);
        }
    }
}
```

8.

(a) Indexes examined: 7, 3, 1, 2
Value returned: -4

(b) Initial array: {4, 444, 44, 33, 333, 3, 5, 555}
after 1 pass: {**3**, 444, 44, 33, 333, **4**, 5, 555}
after 2 passes: {3, **4**, 44, 33, 333, **444**, 5, 555}
after 3 passes: {3, 4, **5**, 33, 333, 444, **44**, 555}

(c)
```
{  4, 444,  44,  33,  333,   3,  5, 555}
{  4, 444,  44,  33} {333,   3,  5, 555}
{  4, 444} {44,  33} {333,   3} {5, 555}
{  4}{444} {44}{ 33} {333}{  3} {5}{555}
{  4, 444} {33,  44} {  3, 333} {5, 555}
{  4,  33,  44, 444} {  3,   5,333, 555}
{  3,   4,   5,  33,   44, 333,444, 555}
```

9.

Call	Value Returned
mystery(0)	0
mystery(5)	4
mystery(13)	86
mystery(297)	702
mystery(-3456)	-6543

10. Two solutions are shown below.

```
public static boolean isSorted(int x) {
    if (x < 0) {
        return isSorted(-x);
    } else if (x < 10) {
        return true;
    } else {
        int last = x % 10;
        int rest = x / 10;
        int secondToLast = rest % 10;
        return (secondToLast <= last && isSorted(rest));
    }
}
```

```
public static boolean isSorted(int x) {
    if (x < 0) {
        return isSorted(-x);
    } else {
        return x < 10 || (x / 10 % 10 <= x % 10 && isSorted(x / 10));
    }
}
```