# CSE 143 Sample Midterm Exam #1 Key

1.

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

2. Two solutions are shown.

```
public static void stretch(ArrayList<String> list, int factor) {
    if (factor <= 0) {
        list.clear();
    } else {
        for (int i = 0; i < list.size(); i += factor) {
            for (int j = 0; j < factor - 1; j++) {
                String element = list.get(i);
                list.add(i, element);
            }
        }
    }
}

public static void stretch(ArrayList<String> list, int factor) {
    if (factor <= 0) {
        list.clear();
    } else {
        int size = list.size();
        while (list.size() < size * factor) {   // pad list with zeros
            list.add(0);
        }
        for (int i = list.size() - 1; i >= 0; i--) {
            list.set(i, list.get(i / factor));
        }
    }
}
```

3. Two solutions are shown.

```
public static void compressDuplicates(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<Integer>();
    while (!s.isEmpty()) {
        q.add(s.pop());          // s -> q
    }
    while (!q.isEmpty()) {
        s.push(q.remove());      // q -> s, to reverse the stack order
    }
    while (!s.isEmpty()) {       // s -> q
        q.add(s.pop());
    }
    if (!q.isEmpty()) {          // q -> s, replacing dupes with (count, val)
        int last = q.remove();
        int count = 1;
        while (!q.isEmpty()) {
            int next = q.remove();
            if (next == last) {
                count++;
            } else {
                s.push(count);
                s.push(last);
                count = 1;
                last = next;
            }
        }
        s.push(count);
        s.push(last);
    }
}

public static void compressDuplicates(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<Integer>();
    s2q(s, q);
    q2s(q, s);
    s2q(s, q);
    if (!q.isEmpty()) {          // q -> s, replacing dupes with (count, val)
        int last = q.remove();
        int count = 1;
        while (!q.isEmpty()) {
            int next = q.remove();
            if (next == last) {
                count++;
            } else {
                s.push(count);
                s.push(last);
                count = 1;
                last = next;
            }
        }
        s.push(count);
        s.push(last);
    }
}
```

4.

```
public static int countInAreaCode(Map<String, String> numbers, String areaCode) {
    Set<String> uniqueNumbers = new HashSet<String>();
    for (String name : numbers.keySet()) {
        String phoneNumber = numbers.get(name);
        if (phoneNumber.startsWith(areaCode)) {
            uniqueNumbers.add(phoneNumber);
        }
    }
    return uniqueNumbers.size();
}
```

5.
```
list2.next.next.next = list;     // 4 -> 1
list.next = list2;               // 1 -> 2
list = list2.next.next;          // list -> 4
list2 = list2.next;              // list2 -> 3
list2.next = null;               // 3 /
list.next.next.next = null;      // 2 /
```

6.
```
public boolean isSortedBy(int n) {
    if (n <= 0) {
        throw new IllegalArgumentException();
    }
    ListNode current1 = front;
    ListNode current2 = front;
    while (current2 != null && n > 0) {
        current2 = current2.next;
        n--;
    }
    while (current2 != null) {
        if (current1.data > current2.data) {
            return false;
        }
        current1 = current1.next;
        current2 = current2.next;
    }
    return true;
}
```

7. Two solutions are shown.

```
public class BankAccount implements Comparable {

    ...

    public int compareTo(BankAccount other) {
        if (balance > other.getBalance()) {
            return 1;
        } else if (balance < other.getBalance()) {
            return -1;
        } else if (id > other.getID()) {
            return 1;
        } else if (id < other.getID()) {
            return -1;
        } else {
            return 0;
        }
    }
}

public class BankAccount implements Comparable<BankAccount> {

    ...

    public int compareTo(BankAccount other) {
        if (Math.signum(balance - other.getBalance()) != 0.0) {
            return (int) Math.signum(balance - other.getBalance());
        } else {
            return id - other.getID();
        }
    }
}
```

8.

(a) Indexes examined: 7, 11, 9
Value returned: 9

(b) Initial array: `{51, 41, 21, 71, 11, 81, 61, 31}`
after 1 pass: {**<u>11</u>**, 41, 21, 71, **<u>51</u>**, 81, 61, 31}
after 2 passes: {11, **<u>21</u>**, **<u>41</u>**, 71, 51, 81, 61, 31}
after 3 passes: {11, 21, **<u>31</u>**, 71, 51, 81, 61, **<u>41</u>**}

(c)
```
{51, 41,  21, 71,  11, 81,  61, 31}
{51, 41,  21, 71} {11, 81,  61, 31}
{51, 41} {21, 71} {11, 81} {61, 31}
{51}{41} {21}{71} {11}{81} {61}{31}
{41, 51} {21, 71} {11, 81} {31, 61}
{21, 41,  51, 71} {11, 31,  61, 81}
{11, 21,  31, 41,  51, 61,  71, 81}
```

9.

| Call | Value Returned |
|---|---|
| `mystery(7)` | 8 |
| `mystery(42)` | 53 |
| `mystery(385)` | 496 |
| `mystery(-790)` | -801 |
| `mystery(89294)` | 90305 |

10.
```
public static int digitMatch(int x, int y) {
    if (x < 0 || y < 0) {
        throw new IllegalArgumentException();
    } else if (x < 10 || y < 10) {
        if (x % 10 == y % 10) {
            return 1;
        } else {
            return 0;
        }
    } else if (x % 10 == y % 10) {
        return 1 + digitMatch(x / 10, y / 10);
    } else {
        return digitMatch(x / 10, y / 10);
    }
}
```