CSE 143, Winter 2011
Midterm Exam Key

1. `ArrayList` Mystery

List	Output
(a) `[1, 20, 3, 40]`	`[1, 3, 20, 40]`
(b) `[80, 3, 40, 20, 7]`	`[3, 20, 7, 40, 80]`
(c) `[40, 20, 60, 1, 80, 30]`	`[20, 1, 30, 60, 40, 80]`

2. Recursive Tracing

Call	Result
a) mystery(8);	0 0 0 : 2 4 8
b) mystery(25);	1 0 0 1 : 3 6 12 25
c) mystery(46);	0 1 1 1 0 : 2 5 11 23 46

3. Stacks and Queues

```
public static void reverseFirstK(int k, Queue<Integer> q) {
    if (q == null || k > q.size()) {
        throw new IllegalArgumentException();
    } else if (k > 0) {
        Stack<Integer> s = new Stack<Integer>();   // first k elements -> S
        for (int i = 0; i < k; i++) {
            s.push(q.remove());
        }

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

        for (int i = 0; i < q.size() - k; i++) {   // wrap around rest of elements so
            q.add(q.remove());                     // k reversed ones appear at front
        }
    }
}
```

```
public static void reverseFirstK(int k, Queue<Integer> q) {
    if (q == null || k > q.size()) {
        throw new IllegalArgumentException();
    }
    if (k > 0) {
        Stack<Integer> s = new Stack<Integer>();
        q2s(q, s);
        s2q(s, q);
        int size = q.size();
        for (int i = 0; i < size - k; i++) {
            s.push(q.remove());
        }
        s2q(s, q);
    }
}
```

4. Collections

```
// ninja solution
public static int duplicateValues(Map<String, String> m) {
    Set<String> set = new HashSet<String>(m.values());
    return m.values().size() - set.size();
}


// "set of previously seen values" solution
public static int duplicateValues(Map<String, String> m) {
    int dupes = 0;
    Set<String> seen = new HashSet<String>();
    for (String v : m.values()) {
        if (seen.contains(v)) {
            dupes++;
        } else {
            seen.add(v);     // also works if you just always add, without 'else'
        }
    }
    return dupes;
}


// "set of previously seen values, but loop over keySet() instead of values()" solution
public static int duplicateValues(Map<String, String> m) {
    int dupes = 0;
    Set<String> seen = new HashSet<String>();
    for (String k : m.keySet()) {
        String v = m.get(k);
        if (seen.contains(v)) {
            dupes++;
        } else {
            seen.add(v);     // also works if you just always add, without 'else'
        }
    }
    return dupes;
}


// "set of unique, count of total" solution
public static int duplicateValues(Map<String, String> m) {
    int total = 0;
    Set<String> unique = new HashSet<String>();
    for (String v : m.values()) {
        total++;
        unique.add(v);
    }
    return total - unique.size();
}


// "name->count map" solution
public static int duplicateValues(Map<String, String> m) {
    Map<String, Integer> counts = new HashMap<String, Integer>();
    for (String v : m.values()) {
        if (counts.containsKey(v)) {
            counts.put(v, counts.get(v) + 1);
        } else {
            counts.put(v, 0);
        }
    }

    int count = 0;
    for (int v : counts.values()) {
        count += v;
    }
    return count;
}
```

5. Linked Lists

```
// "typical" solution
public void frontToBack() {
    if (front != null && front.next != null) {
        ListNode oldFront = front;
        front = front.next;
        ListNode curr = front;
        while (curr.next != null) {
            curr = curr.next;
        }

        curr.next = oldFront;
        oldFront.next = null;
    }
}


// "single temp variable, .next.next" solution
public void frontToBack() {
    ListNode curr = front;
    if (curr != null) {                 // optional test:  && curr.next != null
        while (curr.next != null) {
            curr = curr.next;
        }

        curr.next = front;
        front = front.next;
        current.next.next = null;
    }
}
```

6. Recursion

```
// "typical, lots of if/else" solution
public static int largestDigit(int n) {
    if (n < 0) {
        return largestDigit(-n);
    } else if (n < 10) {
        return n;
    } else {
        int last = n % 10;
        int rest = largestDigit(n / 10);
        if (last > rest) {
            return last;
        } else {
            return rest;
        }
    }
}


// "slightly shorter, no elses" solution
public static int largestDigit(int n) {
    if (n < 0)   return largestDigit(-n);
    if (n <= 9)  return n;
    return Math.max(n % 10, largestDigit(n / 10));
}


// "private helper" solution
public static int largestDigit(int n) {
    return helper(Math.abs(n), 0);
}
private static int helper(int n, int max) {
    if (n < 10) {
        return Math.max(n, max);
    } else {
        return helper(n / 10, Math.max(n % 10, max));
    }
}
```