CSE 143 Midterm Key
Summer 2013

1. Recursive Tracing

Method Call	Value Returned
`mystery(18, 0)`	0
`mystery(8, 18)`	1
`mystery(25, 25)`	2
`mystery(305, 214)`	0
`mystery(20734, 1724)`	2

2. Recursive Programming

One possible solution appears below:

```
public static String dedup(String s) {
    if (s.isEmpty()) {
        throw new IllegalArgumentException();
    } else if (s.length() == 1) {
        return s;
    } else if (s.charAt(0) == s.charAt(1)) {
        return dedup(s.substring(1));
    } else {
        return s.charAt(0) + dedup(s.substring(1));
    }
}
```

3. Collections Mystery

Parameter `list`	`Map` Returned
`[puppy, cat, bird, turtle]`	`{bird=4, cat=3, puppy=5, turtle=6}`
`[foo, bar, baz, bar, qux]`	`{bar=-1, baz=3, foo=3, qux=3}`
`[a, a, double, bang, end, double, a, calm]`	`{a=-1, bang=4, calm=4, double=-1, end=3}`

4. Collections Programming

One possible solution appears below:

```
public static Set<String> extractShorterThan(Set<String> s, int n) {
    Set<String> result = new TreeSet<String>();
    Iterator<String> iter = s.iterator();
    while (iter.hasNext()) {
        String next = iter.next();
        if (next.length() < n) {
            iter.remove();
            result.add(next);
        }
    }
    return result;
}
```

5. Linked Nodes

One possible solution appears below:

Before	After	Code

p->[1]->[2]
q->[3]
	p->[1]->[3]
	q->[2]
		ListNode temp = q;
		q = p.next;
		p.next = temp;

p->[1]->[2]
q->[3]->[4]
	p->[1]->[3]
	q->[4]->[2]
		q.next.next = p.next;
		p.next = q;
		q = q.next;
		p.next.next = null;

p->[1]->[2]->[3]
q
	p->[3]->[2]
	q->[1]
		p.next.next.next = p.next;
		q = p;
		p = p.next.next;
		q.next = null;
		p.next.next = null;

p->[1]->[2]->[3]
q->[4]->[5]
	p->[2]->[1]->[4]
	q->[3]->[5]
		p.next.next.next = q.next;
		ListNode temp = q;
		q = p.next.next;
		p.next.next = p;
		p = p.next;
		p.next.next = temp;
		temp.next = null;

6. Stacks/Queues

One possible solution appears below:

```
public static boolean isPairwiseConsecutive(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<Integer>();
    boolean isConsecutive = true;
    while (!s.isEmpty())
        q.add(s.pop());
    while (!q.isEmpty())
        s.push(q.remove());
    while (!s.isEmpty()) {
        int n = s.pop();
        q.add(n);
        if (!s.isEmpty()) {
            int m = s.pop();
            q.add(m);
            if (Math.abs(n - m) != 1) {
                isConsecutive = false;
            }
        }
    }
    while (!q.isEmpty())
        s.push(q.remove());
    return isConsecutive;
}
```

7. ArrayIntList Programming

One possible solution appears below:

```
public void takeMax(ArrayIntList other) {
    ensureCapacity(Math.max(size, other.size));
    for (int i = 0; i < other.size; i++) {
        if (elementData[i] < other.elementData[i] || i >= size) {
            elementData[i] = other.elementData[i];
        }
    }
    size = Math.max(size, other.size);
}
```