CSE 143, Spring 2014
Midterm Key

1. `ArrayList` Mystery

a) `[2, -5, 6]` `[-5]`

b) `[5, -5, 7, -7, 10]` `[6, -5, 8, -7]`

c) `[2, 4, 6, 8, 10]` `[4, 8]`

d) `[2, 4, 7, 8, 9, 12]` `[4, 8, 9]`

2. Recursive Tracing

a) `mystery(38);` `8-3-8`

b) `mystery(123);` `3-2-1-2-3`

c) `mystery(287);` `7-8-2-8-7`

d) `mystery(7692);` `2-9-6-7-6-9-2`

3. Linked Lists

Problem 1

```
list2.next.next = list1;
list1 = list2.next;
list2.next = null;
```

Problem 2

```
list1.next.next = list2;
list2 = list1.next;
list1.next = list2.next.next;
list2.next.next = null;
```

Problem 3

```
list1.next = list2.next;
list2.next = list2.next.next;
ListNode temp = list1;
list1 = list2;
list2 = temp;
list2.next.next = null;
```

Problem 4

```
list1.next.next = list2;
ListNode temp = list2.next;
list2.next = list2.next.next;
list2 = list1.next;
list1.next = temp;
temp.next = null;
```

4. ArrayIntList

```
public int removeLast(int value) {
   int index = -1;
   for (int i = 0; i < size; i++)
      if (elementData[i] == value)
         index = i;

   if (index != -1) {
      for (int i = index; i < size - 1; i++)
         elementData[i] = elementData[i + 1];
      size--;
   }
   return index;
}

public int removeLast(int value) {
   for (int i = size; i >= 0; i--) {
      if (elementData[i] == value) {
         for (int j = i; j < size - 1; j++)
            elementData[j] = elementData[j + 1];
         size--;
         return i;
      }
   }
   return -1;
}
```

5. Stacks and Queues

```
public static void sort(Queue<Integer> q) {
   Stack<Integer> s = new Stack<Integer>();
   int size = q.size();
   for (int i = 0; i < size; i++) {
      int value = q.remove();
      if (value < 0) {
         s.push(value);
      } else {
         q.add(value);
      }
   }

   int negatives = s.size();
   s2q(s, q);

   for (int i = 0; i < size - negatives; i++) {
      q.add(q.remove());
   }
}

public static void sort(Queue<Integer> q) {
    Stack<Integer> s = new Stack<Integer>();
    q2s(q, s);
    s2q(s, q);
    int oldSize = q.size();
    for (int i = 0; i < oldSize; i++) {
        if (q.peek() < 0) {
            q.add(q.remove());
        } else {
            s.push(q.remove());
        }
    }
    s2q(s, q);
}

public static void sort(Queue<Integer> q) {
    Stack<Integer> s = new Stack<Integer>();
    q2s(q, s);
    s2q(s, q);
    int size = q.size();
    for (int i = 0; i < size; i++) {
        if (q.peek() >= 0) {
            q.add(q.remove());
        } else {
            s.push(q.remove());
        }
    }
    s2q(s, q);
    q2s(q, s);
    s2q(s, q);
}
```

6. Collections

```
public static Set<String> cancelCourse(String course,
                                       Map<String, Set<String>> schedules) {
   Set<String> affected = new TreeSet<String>();
   for (String name : schedules.keySet()) {
      Set<String> courses = schedules.get(name);
      if (courses.contains(course)) {
         courses.remove(course);
         affected.add(name);
      }
   }
   return affected;
}
```

7. Recursive Programming

```
public static int doubleN(int n, int digit) {
    if (digit < 0 || digit > 9) {
        throw new IllegalArgumentException();
    } else if (n < 0) {
        return -doubleN(-n, digit);
    } else if (n == 0) {
        return 0;
    } else if (n % 10 == digit) {
        return 100 * doubleN(n / 10, digit) + digit * 11;
    } else {
        return 10 * doubleN(n / 10, digit) + n % 10;
    }
}

public static int doubleN(int n, int digit) {
    if (digit < 0 || digit > 9) {
        throw new IllegalArgumentException();
    } else if (n < 0) {
        return -doubleN(-n, digit);
    } else if (n < 10) {
        if (n != digit) {
            return n;
        } else {
            return n * 11;
        }
    } else if (n % 10 == digit) {
        return 100 * doubleN(n / 10, digit) + digit * 11;
    } else {
        return 10 * doubleN(n / 10, digit) + n % 10;
    }
}
```