CSE143X Section #3 Problems

For problems 1-3, you are writing a method for the ArrayIntList class:

```
public class ArrayIntList {
    private int[] elementData;
    private int size;

    <methods>
}
```

Unless otherwise noted, assume that you may not call any other methods of the class in solving the problem.

1. Write a method isPairwiseSorted that returns whether or not a list of integers is pairwise sorted (true if it is, false otherwise). A list is considered pairwise sorted if each successive pair of numbers is in sorted (non-decreasing) order. For example, if a variable called list stores the following sequence of values:

   ```
   [3, 8, 2, 5, 19, 24, -3, 0, 4, 4, 8, 205, 42]
   ```

   then the following call:

   ```
   list.isPairwiseSorted()
   ```

   should return the value true because the successive pairs of this list are all sorted: (3, 8), (2, 5), (19, 24), (-3, 0), (4, 4), (8, 205). Notice that the extra value 42 at the end had no effect on the result because it is not part of a pair. If the list had instead stored the following:

   ```
   [1, 9, 3, 17, 4, 28, -5, -3, 0, 42, 308, 409, 19, 17, 2, 4]
   ```

   then the method should return the value false because the pair [19, 17] is not in sorted order. If a list is so short that it has no pairs, then it is considered to be pairwise sorted.

2. Write a method called mirror that doubles the size of a list of integers by appending the mirror image of the original sequence to the end of the list. The mirror image is the same sequence of values in reverse order. For example, if a variable called list stores this sequence of values:

   ```
   [1, 3, 2, 7]
   ```

   and we make the following call:

   ```
   list.mirror();
   ```

   then it should store the following values after the call:

   ```
   [1, 3, 2, 7, 7, 2, 3, 1]
   ```

   Notice that it has been doubled in size by having the original sequence appearing in reverse order at the end of the list. You may not make assumptions about how many elements are in the list although you may assume that the array has sufficient capacity to store the new list.

3. Write a method called fromCounts that converts an ArrayIntList of counts into a new ArrayIntList of values that the method returns. Assume that the ArrayIntList that is called stores a sequence of integer pairs that each indicate a count and a number. For example, suppose that an ArrayIntList called list stores the following sequence of values:

```
[5, 2, 2, -5, 4, 3, 2, 4, 1, 1, 1, 0, 2, 17]
```

This sequence of pairs indicates that you have 5 occurrences of 2, followed by two occurrences of -5, followed by 4 occurrences of 3, and so on. If we make the following call:

```
ArrayIntList list2 = list.fromCounts();
```

Then the variable list2 should store the following sequence of values:

```
[2, 2, 2, 2, 2, -5, -5, 3, 3, 3, 3, 4, 4, 1, 0, 17, 17]
```

Assume that the ArrayIntList that is called stores a legal sequence of pairs (which means it will always have an even size) and that the default constructor for ArrayIntList will construct an array of sufficient capacity to store the result. Your method should not change the original list. If the sequence of pairs is empty, the result should be an empty list.

4. Below is a "bad" version of the ArrayIntList. It has mostly the same functionality as the version discussed in lecture, but it has poor style. Approximately half of the points for each programming assignment in this class will be devoted to style issues, so it is important to understand style issues. What is bad about this version?

```
// Stuart Reges (my name is Elroy Jetson)
// This is the ArrayIntList class.

import java.util.*;

public class ArrayIntList {
    int[] elementData; // element data
    int size;          // size
    int capacity;      // capacity
    public static int defaultCapacity = 100;

    public ArrayIntList() {
        elementData = new int[100];
        size = 0;
        capacity = 100;
    }

    // capacity should be not be negative
    public ArrayIntList(int capacity) {
        if (capacity < 0) {
            throw new IllegalArgumentException();
        } else {
            elementData = new int[capacity];
            size = 0;
            this.capacity = capacity;
        }
    }

    public int size() {
        return size;
    }
```

```
// pre : 0 <= index < size() (throws exception if not)
public int get(int index) {
    if (index < 0 || index >= size) {
        throw new IndexOutOfBoundsException();
    }
    for (int i = 0; i < size; i++) {
        if (i == index) {
            return elementData[i];
        }
    }
    return 0;
}

public String toString() {
    if(size==0) {
        return"[]";
    } else {
          String result="["+elementData[0];
          for(int i=1;i<size;i++) {
           result+=", "+elementData[i];
          }
         return result+"]";
    }
}

// uses a for loop to find out if the array has the given value
public boolean contains(int value) {
    int count = 0;
    for (int i = size - 1; i >= 0; i--) {
        if (elementData[i] == value) {
            count++;
        }
    }
    if (count == 0) {
        return false;
    } else {
        return true;
    }
}

// returns the position of the given value in the array, only checking up
// to the current size
public int indexOf(int value) {
    int index = 0;
    for (int i = size - 1; i >= 0; i--) {
        if (elementData[i] == value) {
            index = i;
        }
    }
    if (elementData[index] == value) {
        return index;
    } else {
        return -1;
    }
}
```

```
    // pre : size() < capacity (throws IllegalStateException if not)
    // post: appends the value to the end of the list
    public void add(int value) {
        if (size > capacity - 1) {
            throw new IllegalStateException();
        }
        elementData[size] = value;
        size++;
    }

    // pre : size() < capacity (throws IllegalStateException if not) &&
    //       0 <= index <= size() (throws IndexOutOfBoundsException if not)
    // post: inserts the value at the index
    public void add(int index, int value) {
        for (int i = size; i >= index; i--) {
            if (index < 0 || index > size) {
                throw new IndexOutOfBoundsException();
            } else if (size > capacity - 1) {
                throw new IllegalStateException();
            } else if (i > index) {
                elementData[i] = elementData[i - 1];
            } else {
                elementData[i] = value;
                size++;
            }
        }
    }

    // post: returns the capacity of the list
    public int capacity() {
        return capacity;
    }

    // pre : 0 <= index < size() (throws IndexOutOfBoundsException if not)
    // post: removes value at the index and decreases the size by 1
    public void remove(int index) {
        if (index < 0 || index >= size) {
            throw new IndexOutOfBoundsException();
        }
        for (int i = index; i < size - 1; i++) {
            elementData[i] = elementData[i + 1];
        }
        size--;
    }

    // appends values to the end of the list
    public void addAll(ArrayIntList other) {
        int[] data = Arrays.copyOf(elementData, size + other.size);
        while (other.size() > 0) {
            data[size++] = other.elementData[0];
            other.remove(0);
        }
        elementData = data;
    }
}
```