Homework 3

Due: Friday, October 17th, 6pm

The next few problems concern the following ADT:

```
/**
 * Represents an immutable collection of integers.
 *
 * Clients can think of a set as a list of integers. However, they can only ask
 * if an integer is present or not. The order of the integers does not matter.
 * The number of times that an integer appears in the list does not matter.
 */
public interface IntSet {
  /**
   * Determines whether n is in the list.
   * @param n the number to look for in the list
   * @returns contains(n, obj), where
   *     contains(n, nil)    := false
   *     contains(n, m :: L) := true           if m = n
   *     contains(n, m :: L) := contains(n, L) if m /= n
   */
  public boolean contains(int n);

  /**
   * Creates and returns a new list containing n as well as all of obj.
   * @param n the number to add to the new list.
   * @returns n :: obj
   */
  public IntSet add(int n);

  /** ... */
  public IntSet remove(int n);
}
```

Task 1 – Representation 64 [8 pts]

Let the abstraction function for `IntSetImpl` be:

```
// AF: obj = this.elems
```

Consider the following representation invariants for `IntSetImpl`, a possible concrete representation of the `IntSet` ADT from above:

- RI 1: The array `this.elems` contains no duplicates.
- RI 2: The array `this.elems` is sorted in ascending order.

Now consider the following four concrete representations of `IntSet`:

```
// (1)
public IntSetImpl(int[] elems) {
    this.elems = elems;
}

// (2)
public IntSetImpl(int[] elems) {
    int[] temp = new int[elems.length];
    int count = 0;
    for (int i = 0; i < elems.length; i++) {
        int key = elems[i];
        boolean elemExist = false;
        for (int j = 0; j < count; j++) {
            if (temp[j] == key) {
                elemExist = true;
                break;
            }
        }
        if (!elemExist) {
            temp[count] = key;
            count++;
        }
    }
    this.elems = Arrays.copyOf(temp, count);
}
```

```
// (3)
public IntSetImpl(int[] elems) {
    this.elems = elems;
    for (int i = 1; i < elems.length; i++) {
        int key = elems[i];
        int j = i - 1;
        while (j >= 0 && elems[j] > key) {
            elems[j + 1] = elems[j];
            j--;
        }
        elems[j + 1] = key;
    }
}

// (4)
public IntSetImpl(int[] elems) {
    this.elems = elems;
    for (int i = 1; i < elems.length; i++) {
        int key = elems[i];
        int j = i - 1;
        while (j >= 0 && elems[j] > key) {
            elems[j + 1] = elems[j];
            j--;
        }
        elems[j + 1] = key;
    }
    int count = 0;
    for (int i = 0; i < this.elems.length; i++) {
        if (i == 0 || this.elems[i] != this.elems[i - 1]) {
            this.elems[count++] = this.elems[i];
        }
    }
    this.elems = Arrays.copyOf(this.elems, count);
}
```

For each concrete representation, please answer the following questions:

1. Which of the RIs (if any) does it satisfy after the constructor is called?
2. If it satisfies an RI, what is its abstract state?
3. Describe the resulting abstract state in English.

Explain all your answers.

a) Answer the above questions for the first concrete representation.

b) Answer the above questions for the second concrete representation.

c) Answer the above questions for the third concrete representation.

d) Answer the above questions for the fourth concrete representation.

Task 2 – The Best Is Set To Come [9 pts]

Suppose that we changed the abstract state of `IntSet` to be a list *with no duplicates*.

a) Rewrite the specifications to use this new abstract state. If the specification of a method does not need any changes, you can just say "`/** .. as before .. */`".

b) Is this specification or the original one simpler?

c) In both this and the original specification, the abstract state was a list. However, this abstract state also has extra conditions that will always be true. What is the term for the extra conditions that will always hold for an abstract state?

Task 3 – ASort The Mission [7 pts]

Consider the implementation of `IntSetImpl` (same as Task 3 from Section 3), which ensures that the representation invariant is satisfied by sorting the elements in the constructor:

```
public class IntSetImpl implements IntSet {
    // AF: obj = this.elems
    // RI: this.elems is sorted in ascending order
    private int[] elems;

    public IntSetImpl(int[] elems) {
        this.elems = elems;

        // Put the elements in sorted order.
        for (int i = 1; i < elems.length; i++) {
            int key = elems[i];
            int j = i - 1;
            while (j >= 0 && elems[j] > key) {
                elems[j + 1] = elems[j];
                j--;
            }
            elems[j + 1] = key;
        }
    }
}
```

a) Why is it not possible to test this constructor in a JUnit test at present? Explain why adding the following method would allow us to do so:

```
public int[] getElemsForTesting() {
    return this.elems;
}
```

b) Explain why `getElemsForTesting` is a better name for this method than `getElems`.

c) Suppose that we implement `contains` with binary search and test it as follows:

```
@Test
public void testContains() {
    IntSet set = new IntSetImpl(new int[] {3, 2, 1});
    assertEquals(false, set.contains(0));
    assertEquals(true, set.contains(1));
    assertEquals(true, set.contains(2));
    assertEquals(true, set.contains(3));
    assertEquals(false, set.contains(4));
}
```

Now suppose that this test fails on the line that calls `set.contains(0)`. Just looking at this test alone, where in the `IntSetImpl` class could the bug that caused this failure be?

d) Explain why unit tests of non-static methods will always test more than just that one method. Explain why that will make it more difficult when a test fails.

e) In what way would the following alternative implementation of the constructor be a better option?

```
/**
 * Creates an IntSet containing the given elements.
 * @param elems The elements of the new set. Must already be *sorted*.
 */
public IntSetImpl(int[] elems) {
    this.elems = elems;
}
```

f) More generally, what can we say about how constructors should be written in order to make sure that debugging failed tests is easy?

Task 4 – Optional: A Priest, a Minister, and an A-I [10 pts]

In this problem, we will use AI to implement the methods of `IntSetImpl`. For parts b-c and f-g, refer to the concrete representations found in Task 1.

Start with the following dummy implementation:

```
public class IntSetImpl implements IntSet {
    private int[] elems;

    public IntSetImpl(int[] elems) {
        this.elems = elems;
    }

    public boolean contains(int n) {
        return false;
    }

    public IntSet add(int n) {
        return this;
    }

    IntSet remove(int n) {
        return this;
    }
}
```

a) Which AI are you using?

Prompt the AI to fill in the body of the `contains` method. Write out your prompt and the code that the AI produces.

b) With which of the concrete representations from Task 1 (if any) would have this code be correct?

c) Delete the generated code for `contains`.

Now, add comments for specification (3), and prompt it again to fill in the body of `contains`. What code does it produce?

d) Based on what you saw in these last two parts, would you agree or disagree with the following statement: "Comments are important for explaining the code to other humans, but AI doesn't need them."

e) Delete the generated code for `contains` and the comments above `elems`. Next, we will try a similar experiment with the `add` method.

Prompt the AI to fill in the body of the `add` method. Write out your prompt and the code that the AI produces.

f) With which of the concrete representations from Task 1 (if any) would have this code be correct?

g) Delete the generated code for `add`.

Now, add comments for one of the specifications from section for which the generated code was not correct, and try prompting it again with the same prompt. Does it produce correct code? If not, try rephrasing your comments a couple of times to see if different phrasing works.

Task 5 – Extra Credit: List Me, List Me, Now You've Gotta Kiss Me [12 pts]

In this problem, we will formalize an English description of one possible concrete representation of an association list, which is the abstract state of the `Map` type.

Recall that an association list is a list of $(\text{key}, \text{value})$ pairs. The primary operation on such a list is to retrieve the (first) value associated with a given key in the list.

In this problem, you are free to make use of any of the functions included in our List reference.

a) The function "zip" takes two lists and returns a single list of pairs, where each element of the first list is paired with the element of the second list at the same index For example, zipping $1 :: 2 :: 3 ::$ nil and $7 :: 8 :: 9 ::$ nil would produce the list $(1, 7) :: (2, 8) :: (3, 9) ::$ nil.

Write a formal definition of this function in our math notation. This should *only be defined* when both lists are the same length.

b) How many test cases are required to get proper coverage for our zip function? Explain your answer and also give a specific set of test inputs that would give proper coverage.

c) Write a formal specification for a concrete representation that stores the keys and values each in their own array.

Remember that the abstract state of an array is a List. (The array is itself an ADT.)

d) Explain how your AF and RI each deal with states where the two arrays do not have the same length. (Note that the type system cannot prevent this, so it must be handled by the AF and RI.)

e) Write a formal specification of the `containsKey(int)` method that checks whether the given key is associated with some value.