Homework 3

Due: Wednesday, January 28th, 11:59pm

The next few problems concern the following ADT (same ADT from section):

```
/**
 * 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);

  /**
   * Creates and returns a new list containing all of the numbers of obj
   * except for the number n, which is no longer included.
   * @param n the number to not include in the new list.
   * @returns remove(n, obj), where
   *      remove(n, nil)    := nil
   *      remove(n, m :: L) := remove(n, L)      if m = n
   *      remove(n, m :: L) := m :: remove(n, L) if m /= n
  */
  public IntSet remove(int n);
}
```

Task 1 – Concrete the Square [12 pts]

Let the abstraction function for `IntSetImpl` be:

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

Note that arrays can be thought of as lists in math.
Consider the following representation invariants for `IntSetImpl`, a possible concrete representation of the `IntSet` ADT from above:

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

For each implmentation of `IntSet`, 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?
   An example response to this part could be: "The abstract state is an array of characters that is sorted in alphabetical order with duplicates." This is simply an example of response format, not a hint to the question.

a)

```
public IntSetImpl(int[] elems) {
    this.elems = elems;
    for (int i = 0; i < elems.length - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < elems.length; j++) {
            if (elems[j] < elems[minIndex]) {
                minIndex = j;
            }
        }
        int temp = elems[i];
        elems[i] = elems[minIndex];
        elems[minIndex] = temp;
    }
}
```

b)

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

c)

```
public IntSetImpl(int[] elems) {
    boolean[] seen = new boolean[elems.length];
    int count = 0;
    for (int i = 0; i < elems.length; i++) {
        if (!seen[i]) {
            count++;
            for (int j = i+1; j < elems.length; j++) {
                if (elems[j] == elems[i]) {
                    seen[j] = true;
                }
            }
        }
    }
    this.elems = new int[count];
    int index = 0;
    for (int i = 0; i < elems.length; i++) {
        if (!seen[i]) {
            this.elems[index++] = elems[i];
        }
    }
}
```

d)

```
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);
}
```

Task 2 – Set It and Forget It [10 pts]

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

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

Note: the specifications of an ADT include a description of itself at the top. Reference the ADT on the first page to consider what components need to be changed.

Task 3 – The Good, the Add, and the Ugly [12 pts]

In this problem, we will consider three different concrete representations, similar to Task 1.

```
    public class IntSetImpl implements IntSet {

(1)   // AF: obj = this.elems
      private int[] elems;

(2)   // AF: obj = this.elems
      // RI: this.elems contains no dups
      private int[] elems;

(3)   // AF: obj = this.elems
      // RI: this.elems contains no dups and is sorted in ascending order
      private int[] elems;

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

For each of the following implementations of add, state the concrete representation(s) (1–3) for which the implementation would satisfy the specification of the add method in our provided ADT. Just like in section, in each case, briefly explain why.

a)

```
public IntSet add(int n) {
  int[] newElems = new int[this.elems.length + 1];
  System.arraycopy(this.elems, 0, newElems, 1, this.elems.length);
  newElems[0] = n;
  return new IntSetImpl(newElems);
}
```

b)

```
public IntSet add(int n) {
  if (this.contains(n)) {
    return this;
  } else {
    int[] newElems = new int[this.elems.length + 1];
    System.arraycopy(this.elems, 0, newElems, 1, this.elems.length);
    newElems[0] = n;
    return new IntSetImpl(newElems);
  }
}
```

c)

```
public IntSet add(int n) {
  int[] newElems = new int[this.elems.length +1]
  boolean inserted = false;
  int j = 0;
  for (int i = 0; i < this.elems.length; i++) {
    if (this.elems[i] == n) {
      return this;
    }
    if (! inserted && n < this.elems[i]) {
        newElems[j++] = n;
        inserted = true;
    } else {
        newElems[j++] = this.elems[i];
    }
  }
  if (!inserted) {
    newElems[j] = n;
  }
  return new IntSetImpl(newElems);
}
```

Task 4 – Flick of the List [12 pts]

In this problem, we will formalize an English description of one possible concrete representation of an association list, a list of $(\text{key}, \text{value})$ pairs. This is the abstract state of the `Map` type. The primary operation on such a list is to retrieve the (first) value associated with a given key in the list. 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 $1 :: 2 :: 3 :: \text{nil}$ and $7 :: 8 :: 9 :: \text{nil}$ would produce the list $(1, 7) :: (2, 8) :: (3, 9) :: \text{nil}$.

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 using our testing heuristics.

c) Write a formal specification for a concrete representation, where the keys and values are each stored in their own array. Follow the format of Task 3's concrete representations.

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

d) There are states where the key and value arrays are not the same length. In your concrete represenation from part (c), why can we guarantee that functions will only be called on arguments that are defined? In other words, how does our concrete representation handle the case where key-value arrays are different lengths?

Task 5 – Optional: A Priest, a Minister, and an A-I [0 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.