University of Puerto Rico – Mayagüez
Department of Computer Science and Engineering

CIIC 4020 / ICOM 4035 – Data Structures
Prof. Juan O. López Gerena
Spring 2019-2020
Laboratory #6 – Sorted Lists

In this lab activity, we'll implement the Sorted List ADT discussed in lectures. We'll be working on the following implementations:

- Using a linked structure
- Using a dynamic array
- Embedding a List

You should start by importing the provided zip file, which contains some code to help get you started, including a tester. Once you import the code, open the Sorted List interface; it contains comments in Javadoc format to document the expected behavior.

Exercise 1: Using a linked structure

Complete the implementation of the SortedLinkedList class. Note that the following methods are already implemented in the `AbstractSortedList` class:

- **`contains`**
- **`size`**
- **`isEmpty`**
- **`clear`**

Also note that we are NOT using a dummy header, so you must handle carefully the special cases when adding/removing at the beginning of the list.

To take full advantage of the tester, you should implement the methods in the following order:

1. **`toArray`**
2. **`add`**
3. **`get`**
4. **`firstIndex`**
5. **`remove`**
6. **`removeIndex`**

As you implement each method, un-comment the corresponding block of code from the tester so that you may test your implementation.

Exercise 2: Using a Dynamic Array

Complete the implementation of the `SortedArrayList` class. Most of the methods have already been implemented; you only need to implement the following:

1. **`add`**
2. **`removeIndex`**

Modify the tester to use `SortedArrayList` instead of `SortedLinkedList` and test your code.

You should take the time to compare the **`firstIndex`** and **`remove`** methods of this class (already implemented for you) and those same methods in the `SortedLinkedList` class (implemented by you in the previous exercise). In particular, analyze the use of the private methods **`getIndex`** and **`binarySearch`**, already implemented and documented so that may study their behavior and functionality.

Exercise 3: To binary search, or not to binary search, that is the question…

In the previous exercise we were able to take advantage of binary search because our list is sorted. However, in our first exercise, we also had a sorted list, yet we didn't use binary search. The question for you to answer is:

Could we also use binary search in `SortedLinkedList`?

- If you answered no, why not?
- If you answered yes, why didn't/shouldn't we do it?

Within the `SortedArrayList` class, search for "Exercise 3" (under the **`binarySearch`** method) and enter your answer in the space provided. I encourage you to discuss this exercise with your classmates; share and contrast ideas.

Exercise 4: Using an embedded List

Complete the implementation of the `SortedEmbeddedList` class. In contrast to the other exercises, the only method that has been already implemented is **`toArray`**. Moreover, to fully harness the power of the embedded list, you must override the methods that were implemented in `AbstractSortedList`. However, many of the implementations should be simple one-liners because you will be delegating the work to the embedded list (after all, that's why we're using it). Once you have finished, modify the tester to use this class and test your implementation.

Note that the embedded list has no notion of order. You must establish the order by using an approach similar to Exercise 1, where you continue comparing values until you find the appropriate position. The difference is that now you will use the corresponding list methods to obtain the values (for comparisons) and to insert values into their rightful position.

The `List`, `ArrayList`, and `LinkedList` classes have been included in the zip file in a separate package. The **`remove(int index)`** method has been updated to return the element at that index, mimicking the behavior of Java's built-in ArrayList. The `SortedEmbeddedList` code already included uses the LinkedList class; keep it that way. *Do not use binary search in the **`add`** method this time*.

Exercise 5: Analyze Running Time

Fill out the table below by entering the Big-O running time of each of the listed methods under the different Sorted List implementations. It is assumed that the dynamic array is the only implementation that uses binary search. You should fill each entry with one of the following:
O(1), O(log n), O(n), O(n log n), O(n^2), where n is the length of the sorted list.

Method	**Linked structure**	**Dynamic array**	**Embedded LinkedList**	**Embedded ArrayList**
add				
remove				
removeIndex				
firstIndex				
get				

Note: A file named `Exercise 5.txt` has been included in the zip file. You must enter your answers there, so that the file may be submitted with your code.

The ArrayList class was included in the zip file in case you wanted to review it for the purpose this exercise. You may wish to do the same, if necessary, with the LinkedList class. Once again, I encourage you to discuss this exercise with your classmates.