CPS 112: Computational Thinking with
Algorithms and Data Structures
Homework 6 (20 points)

Handed out: Nov 1, 2017 (Wednesday)
Due: 11:59 pm Nov 8, 2017 (Wednesday)
Late submissions accepted with penalty until 11:59 pm Nov 10, 2017 (Friday)

If you run into trouble or have questions, arrange to meet with me or a tutor!

Before you submit:

- **Please review the homework guidelines available on Canvas**

- Read the assignment very carefully to ensure that you have followed all instructions and satisfied all requirements.

- Create a zip file named like this: *yourusernameHWX.zip* (with your username and replacing the *X* with the assignment number) that contains a directory named *yourusernameHWX*, which contains all of your .java files, your debug log, and your cover sheet.

- Make sure to compile and thoroughly test all of your programs before you submit your code. **Any file that does not compile will receive a 0!**

- Ensure that your code conforms to the style expectations set out in the homework guidelines.

    - Make sure your variable names are descriptive and follow standard capitalization conventions.

    - Put comments wherever necessary. Comments at the head of each file should include a description of the contents of the file **(no identifying information please!)**. Comments at the beginning of methods describe what the method does, what the parameters are, and what the return value is. Use comments elsewhere to help your reader follow the logical flow of your code.

    - *Program readability and elegance are as important as correctness.* After you've written your function, read and re-read it to eliminate any redundant/unused lines of code, and to make sure variable and function names are intuitive and relevant.

Part 0: Fancier Iterators

In class we've been implementing sorting algorithms for arrays. How about lists? You can efficiently access an array-backed list at any index, just like an array, but indexing is expensive for linked lists. Of course we know how to solve that problem: iterators! So, in this project you will implement sorting algorithms that use only iterators, so they have the same complexity for linked lists as they have for array-backed lists.

Of course, to sort you will need some fancier iterators! You have been supplied with the classes `SimpleList`, `SimpleArrayList`, `SimpleLinkedList`, which are lists that store `int`s. Also `SimpleListIterator`, `SimpleArrayListIterator`, and `SimpleLinkedListIterator`, which are iterators for the lists.

The iterators provide the following methods:

- `next()` – returns the item after the last item returned
- `hasNext()` – returns `true` if there is an item after the last item returned, `false` otherwise
- `previous()` – returns the item before the last item returned
- `hasPrevious()` – returns `true` if there is an item after the last item returned, `false` otherwise
- `set(item)` – replaces the last item returned with the given item
- `equals(SimpleListIterator)` – returns `true` if `this` is pointing to the same position as the given iterator, `false` otherwise
- `before(SimpleListIterator)` – returns `true` if `this` is pointing to a position before that of the given iterator, `false` otherwise
- `clone()` – returns a new iterator pointing to the same position as `this`
- `toString()` – returns a string containing the index the iterator is pointing to (useful for debugging)

You'll implement your functions in `Sorts.java`. The main function takes a list of integers as command line arguments and then calls the various sorting functions with iterators from array-backed and linked lists.

(10 pts) Part 1: Selection Sort

In *Sorts.java* implement the `selectionSort` method. It takes a single iterator, which you should assume is positioned just before the first element of the list (that is, if you call `next` it will return the first element). It should perform the Selection Sort algorithm.

Important rules:

- You may not move the given iterator; you should `clone` it, and move the clone(s).
- You may only move iterators forward; you may not use the `previous` method.
- You may only use the `toString` method for debugging; you may not use it to extract the index that an iterator is pointing to.
- Your method must have O(1) space complexity. In particular, you may not use the iterator to create a copy of the list.

Note that because you may only move your iterators forward, you will have to implement a different

variation of the Selection Sort algorithm than the one we discussed in class. In class we searched for the maximum unsorted element and swapped it with the last unsorted element. Instead, you should search for the *minimum* unsorted element and swap it with the *first* unsorted element.

Please, I beg you, do some examples by hand before you get started! If you know what you are trying to do before you start coding, you will be *so much* better off.

Make sure you thoroughly test your algorithm on different kinds of lists. You may need some special cases to handle some kinds of input.

(10 pts) Part 2: Insertion Sort

In *Sorts.java* implement the `insertionSort` method. It takes a single iterator, which you should assume is positioned just before the first element of the list (that is, if you call `next` it will return the first element). It should perform the Insertion Sort algorithm.

To do insertion sort, you will have to move iterators backwards sometimes. Other than that, the same rules apply.

Important rules:

- You may not move the given iterator; you should `clone` it, and move the clone(s).
- You may only use the `toString` method for debugging; you may not use it to extract the index that an iterator is pointing to.
- Your method must have O(1) space complexity. In particular, you may not use the iterator to create a copy of the list.

Again, you should do some examples by hand before you get started! If you know what you are trying to do before you start coding, you will be *so much* better off.

Again, make sure you thoroughly test your algorithm on different kinds of lists. You may need some special cases to handle some kinds of input.