Sorting is a fundamental part of computer science with many books being written on the subject. In this lab you will implement two basic sorting algorithms that you have encountered in lectures: selection sort and insertion sort.

### Insertion sort

We may describe insertion sort using English, or we can set down the pseudocode. Here's a bit of both:

Insertion sort works the same way many people sort a hand of cards. We imagine that everything to the left of a certain point is already sorted. We take the first item to the right of that and "pull it out" (leaving a "gap"). We then move everything in the sorted part one place over to the right until a gap opens up at just the right place for us to "insert" what we pulled out. Our left-hand-side is still sorted, but now it is one item longer, and the right-hand-side is one item shorter.

```
for each position p in array a except the first
   pull out the item at p and store it in variable 'key'
   move each item that is to the left of position p, and is
      greater than key, one place to the right
   put key in the leftmost vacated position
```

### Selection sort

The basic idea is to find the smallest item in a list. Swap it with the first item. Find the smallest item in the remainder of the list, swap it with the first item of the remainder. Repeat until finished.

Here is a slightly more detailed description of selection sort, in both English and pseudocode:

Selection sort assumes that you know how to pick out the *smallest* item in an array of items. Once again, we conceptually break the array into two pieces, left and right. We have n items altogether.

Pick the smallest item in the section from 0 to (n - 1) and swap it with whatever is in position 0. Pick the smallest item in the section 1 to (n - 1) and swap it with whatever is in position 1. Now pick the smallest item in section 2 to (n - 1) and swap it with whatever is in position 2…you get the idea. Stop when you get to position n - 2, since the item now in the last position (n - 1) must be the largest item.

```
for each position p in the array a except the last one
    find the smallest item from position p to position (n – 1)
    swap the smallest item with whatever is at position p now
```

### Provided files

You have been provided with some test files and three Java classes in the directory `/home/cshome/coursework/241/pickup/09`. You should be able to copy them to your `~/241/09` directory and then compile the code using the usual command:

```
javac -d .  -Xlint *.java
```

If you run the code like this:

```
java week09.SortApp < 200-nums
```

it will sort the input numbers and print them out to *stdout* one per line (as well as print out to *stderr* the number of comparisons used when sorting the numbers).

If you run the code like this:

```
java week09.SortApp -g < 200-nums
```

it will pop up a GUI that will allow you to observe the sort as it takes place. Note, that you must use the variables $i$, $j$, $nums[]$ and $comparisons$ which are declared in the class `Sorter` as well as call the `update()` method appropriately in order to be able to view your sort taking place in the GUI.

---

### Part one (1%)

Using `SillySort` as a basis, implement `SelectionSort` and adjust `SortApp` and `Sorter` so that you can run selection sort like this:

```
java week09.SortApp 1 -g < 200-nums
```

The only changes you should need to make to the provided code is to uncomment the `SELECTION` case inside the `createSorter()` method of `SortApp`, and uncomment the `SELECTION` enumerated type at the top of the `Sorter` class.

---

### Part two (1%)

Repeat the similar steps required to implement and test an `InsertionSort` class. You can run insertion sort using an argument of 2, like this:

```
java week09.SortApp 2 -g < 200-nums
```

When both your selection sort and insertion sort are working correctly you can use the command `241-check` to make sure they pass all of our tests. If all is well then you can submit them using `241-submit` as usual.