**Shaker Sort (Non-Recursive Sorting)**

**Summary**

Implement shaker sort (also known as cocktail-shaker sort and bidirectional bubble sort). Shaker sort differs from bubble sort in that it pushes small items toward the front of the array as well as pushing large items toward the back. This improves efficiency *a very tiny bit*.

Your version of shaker sort should also include the improvement of noticing where the last swap was so that it doesn't waste time investigating parts of the array that are already sorted.

**Details**

I suggest you begin by downloading and renaming the **BubbleSort** program linked above. This is a modified version of the code available in the course notes.

The first modification is actually to have the sorting method keep track of where the last swap was. Change the name of the variable i to lastUnsorted -- it's going to keep track of the end of the unsorted part of the array. Now, instead of just subtracting 1 from lastUnsorted each time, set it equal to the index of the last swap made.

Create a variable for the last swap position. Reset it to zero each time you start the outer loop. Then update it to j whenever you swap a[j] with a[j+1].

Test your program by confirming that the list [A, B, C, D] takes 0 assignments and 3 comparisons to sort, while the list [B, C, A, D, E] takes 6 assignments and 5 comparisons.

At this point change your for loop into a while loop. It'll make the later revisions less confusing. (Re-run the program to make sure you didn't break it!)

Now add the code to push small items toward the front of the array. This means adding a loop after you set lastUnsorted equal to the position of the last swap. The new loop is just the reverse of the inner loop for pushing large items to the back. That is, it starts at the last unsorted element of the array and works its way back to the first unsorted element of the array. (So you're going to need to keep track of the first unsorted element of the array, too. It starts at zero, and each time gets reset to the last swap you did in this new loop.)

**Hint:** In the new loop, compare a[j - 1] with a[j] and save j as the position of the last swap.

You'll need to revise your outer loop and your first inner loop so that they start at the first unsorted position (rather than at zero).

Test your code by checking that the list [A, G, B, C, E, F, D] takes 21 assignments and 12 comparisons to sort. (The counts for the other lists shouldn't have changed.)

Finally, activate the sortNumbers(); command to make sure that your sorting method is working properly. It should report about 108 million operations in total.

**Grading Outline**

* 60% -- Your program works like it should.
* 30% -- Your program shows good design.
* 20% -- Submitted material meets the standard requirements as described in the [rules for submissions](http://cs.smu.ca/~myoung/csci2341/Rules.html) and the [style rules](http://cs.smu.ca/~myoung/csci2341/Style.html).

[SUBMIT](http://cs.smu.ca/~myoung/csci2341/protected/getSubmission.cgi?10)   /   [Check](http://cs.smu.ca/~myoung/csci2341/protected/viewasgn.cgi?A10)