Exercises for Lab 3:

These must be completed and shown to your lab TA either by the end of this lab, or by the start of the following lab).

Remember that you are to work with a partner during every lab.

1. We haven't talked about sorting in class, but this lab is really about complexity, not sorting. If you want to know more about sorting, you can take a look at Chapter 10 in your Koffman and Wolfgang book.
   Consider the following two pieces of pseudocode for an approach to sorting called bubble sort:

```
procedure bs( A: array )
do
     swapped := false
     for each i in 0 to length(A) - 2 do
          if A[i] > A[i+1] then
               swap( A[i], A[i+1] )
               swapped := true
          end if
     end for
while swapped
end procedure


procedure bs_optimized( B: array )
n := length(B)
do
     n := n - 1
     swapped := false
     for each i in 0 to n - 1 do
          if B[i] > B[i+1] then
               swap( B[i], B[i+1] )
               swapped := true
          end if
     end for
while swapped
end procedure
```

a. Trace the following calls on a piece of paper: `bs(A)` and `bs_optimized(A)`, for the array, A: [4,2,1,8,-3]. Show the state of the list as you enter each iteration of the loop, and indicate each swap. Keep a count of the number of *comparisons*. You and

your lab partner can each pick a function to trace and then compare notes when you're done.

b . What is the worst-case time complexity of the optimized algorithm? Show how you arrived at your answer. (Don't worry about c and n_0 here.)

c. How many steps should your algorithm take (roughly) for n = 10, 100, and 1000?

d. What is the worst-case space complexity of the optimized algorithm? In other words, how much space is it going to take up relative to n. Justify your answer. You needn't worry about the actual size of each element since that can simply be multiplied by the result that you get (e.g. if your space complexity was n and you used 12 bytes per element, the it would be roughly 12n; for this question just figure out the value in terms of n and don't worry about the coefficient).

Not for lab marks, just for practice:

e. Implement each algorithm (non-optimized and optimized). In each implementation, add a variable, "numComps" that tracks how many comparisons are made (i.e. counting the number of times this is performed A[i] > A[i+1] on an array, A). Record that value for an array of size 10, 100 and 1000 in a comment at the top of your code, or in a separate txt file. *Note you'll need to populate your arrays with random numbers to be sorted, unless you want to enter 1000 numbers by hand!*

f. How do the results you achieved in (e) compare to your predictions in (c)? Discuss any differences.