CSE 655 ***Scheme* Lab** **Due: 11:59 P.M., December 2, 2011 (Lab closes at 6:00).**
Au 2011, 9:30 section

Grade: This assignment is worth 40 points.

Note: We haven't worked on *Scheme* enough for me to give you a really interesting project in it, so this is really a fairly simple (and, by itself, pointless) assignment. All it does is make sure that you get some practice in thinking functionally (which is a useful thing) and play with *Scheme* a little bit. If you want to get into *Scheme*, and it is worth getting into, the book *The structure and interpretation of computer programs* by Abelson and Sussman is highly recommended.

Problem: The assignment is to write a function, `eliminateNsort`, which will receive two lists of numbers as arguments. Suppose *L1, L2* are the two lists. The result returned by the function should be obtained from *L1, L2* as follows: start with *L1* and remove from it all those elements that are also in *L2*; then sort the remaining elements into non-decreasing order to get the result list.

The terminology in the last paragraph is somewhat incorrect. You do not actually want to remove anything from *L1*; you are doing this assignment functionally so values of variables are not supposed to change. (*Scheme* does have a *set!* operation which will let you change the values of variables; do not use it; it is not meant for such problems. Also do not use the *sort* function that is in the Scheme library. Going further, you are to follow the blanket restriction of not using any built-in Scheme functions that we have not mentioned in class.) Rather, the value returned by `eliminateNsort` will be a (new) list which will have all the elements of *L1*, except those that appear in *L2*, and further, this list will be sorted.

Make sure that if some value *x* appears more than once in *L1*, then *x* appears the same number of times in the result returned by `eliminateNsort` unless *x* also appears in *L2* in which case it will not at all appear in the result.

Test your function with several combinations of *L1, L2* values (don't forget empty lists). You will find the "`,load`" command of Scheme useful since it allows you to define a function in a file and then "load it in". You can get help on the load command (as well as some other scheme48-specific commands) by using the "`,help`" command of scheme48. Also useful is the "`,trace` command; it lets you 'trace' a function everytime it is called.

Approach: The trick to designing complex *Scheme* (or any other) programs is of course to define many functions and to do as little as possible in each. If you follow this rule, you will find this assignment reasonably straightforward to complete. To say that somewhat differently: do not try to do two things in a function; if you do, you will likely get in trouble. So any time you find the need for computing something that is somewhat different from the things that the functions you already have (or the one you are currently defining) are capable of computing, introduce a new function rather than trying to add on that new functionality to an existing function.

What to submit: You should submit a single file that contains the definitions of all your functions, including the main one `eliminateNsort`. The file itself should be called `myfns`. Do not use any other name for the file or for the main function. Other functions that you define may have whatever names you choose. `eliminateNsort` should be the last function defined in your file.

Use white space appropriately so that your function definitions are easy to read. And include documentation in the same file (NOT a separate README file). Comment lines in *Scheme* programs start with a semicolon.

Submit your lab by placing your file in the appropriate Carmen Dropbox.