**Homework 3. CSCI 330**

**Due date: Monday February 5th**

1. *Write a program in C++ that implements Quicksort without using recursion.* (hint: this needs a stack; see the lecture video from Jan 23. Use the C++ STL to get a Stack. Your stack objects will contain the ends of the array slice. You can use the partition function from your C++ text. You should not be writing much code on your own.)
2. *Implement a “bottom-up” Mergesort in Lisp.* This can be implemented by following these steps:
   1. Partition the list into sorted pairs. If the list is (1 7 2 1 8 6 5 3 7 9 4), the list will end up as ((1 7) (1 2) (6 8) (3 5) (7 9) (4)). (hint: you need to capture how this process will proceed. For instance, after processing the first two items, we have   
      ((1 7)) and (2 1 8 6 5 3 7 9 4); next we have ((1 7) ( 1 2)) and (8 6 5 3 7 9 4) etc.)
   2. In each pass, merge adjacent lists. As we merge, we move them into another list. Thus, after merging first two, we get   
      (( 1 1 2 7)) and ((6 8) (3 5) (7 9 ) (4)), and then  
      (( 1 1 2 7) (3 5 6 8)) and ((7 9 ) (4)), and finally (( 1 1 2 7) (3 5 6 8) (4 7 9)). This completes the first pass.   
      In the second pass, we merge only ( 1 1 2 7) and (3 5 6 8), to get   
      ((1 1 2 3 5 6 7 8) (4 7 9)). In the third pass, we get the sorted list. (hint: to capture this process, we will need two things; a list of all the lists that have been merged, and a list of all the lists yet to be merged)
   3. Finally we need to capture the process of merging two lists. Recall that merging process has two lists as input, and one list as output. While the process is going on, we therefore have to track 3 lists.
3. *Implement Insertion sort in Lisp.* 
   1. While insertion sort is in progress, we track 2 lists: the sorted items and the unsorted items. (Hint: what should these look like when the process starts and when the process ends? Can we use that information to decide termination?)
   2. In each pass we start with two lists. At the end of the pass, we would have moved one more item from the sorted to the unsorted list. When is this process trivially accomplished? What will the recursive call look like?
   3. Moving requires an operation that can insert an item into a sorted list. To make this actually mimic the array-based algorithm efficiently, we will maintain the sorted list in **descending** order. Again, to represent this insertion process, we need to track the items that have been examined, the item to be inserted, and the items yet to be examined. There are two ways in which the process can terminate – what are they?

**How to submit:**

Create a folder named Hwork3 in your Coursefiles folder. Submit a file for each question *(.cpp* or *.lisp)*. Put comments into the code to explain what is going on.