Assignment 10

C-4.16 Given a sequence S of *n* comparable elements, describe an efficient method for determining whether there are two equal elements in S. What is the running time of your method?

C-4-19 Let S be a sequence of n elements on which a total order relation is defined. An ***inversion*** in S is a pair of elements x and y such that x appears before y in S but x > y. Describe an algorithm running in O(n log n) time for determining the number of inversions in S, i.e., the number of inversions of element **x** in **S** is the count of the number of elements that came before **x** in the original input but are greater than **x** and should be after **x** in the sorted ordering. **Hint:** modify the merge-sort algorithm to solve this problem.

A. Given a Tree ***T***, write a pseudo code algorithm **findDeepestNodes(T)**, that returns a Sequence of pairs **(*v*, *d*)** where ***v*** is an internal node of tree ***T*** and ***d*** is the depth of ***v*** in ***T***. The function must return all internal nodes that are at the maximum depth (no other nodes). What is the time complexity of your algorithm?

B. Suppose you were given the task of sorting thousands of paper documents by name. If you had the documents stacked to the ceiling in one of our large classrooms with twelve tables, devise a plan for sorting these documents.

C. If you have finished everything else and you want to work on a challenging problem for fun.

Design a pseudo code algorithm **createBST(S)** that takes a sorted Sequence S of numbers and creates a balanced binary search tree with height O(log n). Your algorithm must only use the Tree ADT operations for insertion.
Hint: start with an empty tree T, insert the root with **insertRoot(e)** where **e** is the element at rank n/2; then insert the rest using operations **insertLeft(v, e)** or **insertRight(v, e)**; for the **insertLeft(v,e)**, **v** must be an internal node with an external left child; similarly, for **insertRight(v, e)**, **v** must be an internal node with an external right child. Note that these insertion methods return the node that has been inserted into the tree so there is no need to search for the node that was just inserted. Another hint: in the new tree T, a search for a key will reflect a binary search in a sorted Sequence or Array which should eliminate/prune half the tree at each key comparison (drawing the picture from an example should help). What is the time complexity of your algorithm? Can be done in O(n) time.

C-4.25 Bob has a set A of *n* nuts and a set B of *n* bolts, such that each nut in A has a unique matching bolt in B. Unfortunately, the nuts in A all look the same, and the bolts in B all look the same as well. The only kind of comparison that Bob can make is to take a

nut-bolt pair (a,b), such that a is from A and b is from B, and test it to see if the threads are larger, smaller or a perfect match with the threads of b. Describe an efficient algorithm for Bob to match up all of his nuts and bolts. What is the running time of this algorithm, in terms of nut-bolt tests that Bob must make?