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-4.18 Modify Algorithm inPlaceQuickSort (Algorithm 4.17) to handle the general case efficiently when the input sequence, S, may have duplicate keys.

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. **Hint:** try to modify the merge-sort algorithm to solve this problem.

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?

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). Hint: start with an empty tree T and insert the nodes using operation **expandExternal(v)** where **v** is an external node. Another hint: in the new tree T, a search for a key will reflect a binary search in a sorted Sequence or Array (drawing the picture from an example should help).
What is the time complexity of your algorithm?

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. What is the time complexity of your algorithm?