Assignment 5

R-4.5 Suppose we are given two *n*-element sorted sequences A and B that should not be viewed as sets (that is, A and B may contain duplicate entries). Give an O(*n*)-time pseudo-code algorithm for computing a sequence representing the set A ∪ B (with no duplicates).

R-4.9 Suppose we modify the deterministic version of the quick-sort algorithm so that, instead of selecting the last element in an *n*-element sequence as the pivot, we choose the element at rank (index) $\lfloor n/2 \rfloor$, that is, an element in the middle of the sequence. What is the running time of this version of quick-sort on a sequence that is already sorted?

C-4.10 Suppose we are given an *n*-element sequence S such that each element in S represents a different vote in an election, where each vote is given as an integer representing the ID of the chosen candidate. Without making any assumptions about who is running or even how many candidates there are, design an *O(n* log *n)*-time algorithm to see who wins the election S represents, assuming the candidate with the most votes wins.

Modify the algorithm inPlaceQuickSort to handle the general case efficiently when the input Sequence S may have many duplicate keys. Hint: partition into three segments, the first segment containing keys less than the randomly selected pivot (unsorted), the second containing keys equal to the pivot, and the third segment containing keys greater than the pivot (unsorted). The inPlacePartition will return two indices (p1, p2) where p1 is the index of the first element equal to the pivot and p2 is the index of the last element equal to the pivot; the elements equal to the pivot are in sorted order, that is, all elements equal to the pivot will be in their sorted position within the Sequence. You are to write a pseudo code algorithm for inPlacePartition. The top-level of the algorithm that calls inPlacePartition will be as follows:

Algorithm *inPlaceQuickSort*(***S, lo, hi***)
  Input Sequence ***S***, ranks ***lo*** and ***hi*** represent a segment of the Sequence S to sort
  Output Sequence ***S*** with the elements with rank between ***lo*** and ***hi*** rearranged in sorted order
    if ***lo*** < ***hi*** **then**
      (*p1,p2)* ← ***inPlacePartition***(***S, lo, hi***) // keys between p1 and p2 are sorted
      inPlaceQuickSort(***S, lo, p1*** - 1)
      inPlaceQuickSort(***S, p2*** + 1***, hi***)

Let L be a **List** of objects colored either red, green, or blue. Design an **in-place** algorithm **sortRBG(L)** that places all red objects in list L before the blue colored objects, and all the blue objects before the green objects. Thus the resulting List will have all the red objects followed by the blue objects, followed by the green objects. **Hint:** use the method swapElements to move the elements around in the List. **To receive full credit**, you must use positions for traversal, e.g., first, last, after, before, swapElements, etc. which is necessary to make it in-place.