Assignment 9-10

Lesson 8:

Level 1:

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 $\cup$ B (with no duplicates).

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 efficient algorithm to see who wins the election S represents, assuming the candidate with the most votes wins.

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.

A. Implement in JavaScript the Merge Sort pseudo code algorithm given in Lesson 9 for sorting an array. Add this sorting algorithm to the sort algorithms from Assignments 7 and 8. Provided is the ArraySort-test.js file extended to run all of the sorts we have seen so far from Assignments 7, 8, and 9. Compare the number of key comparisons of each algorithm.
B. Using one of your sort algorithms, implement in JavaScript your pseudo code solution to problem C 4.10 to find the winner of an election.

Level 2.

C. Implement your pseudo code solution to question R 4.5.
D. Modify your algorithm to B above to handle a tie, i.e., more than one winner.

Lesson 10.

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?

Take the Quick Sort algorithm provided and add it to the sort algorithms and test it like the others. How does Quick Sort compare to the other sorting algorithms?