**Assignment 9-10**

**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 ∪ B (with no  
duplicates).**

Algorithm removeDups(S)

p:= S.first()

while ! S.isLast(p) do

if containDups(S) then

p:= S.after(p)

S.remove(S.before(p))

if containDups(S,p) then

S.remove(S.before(p))

return S

Algorithm containDups(S)

p:= S.first()

while ! S.isLast(p) do

q:= S.after(p)

while !S.isLast(q) do

if p.element() ! = q.element() then

return false

q:= S.after(q)

if p.element() ! = q.element() then

return false

p:=S.after(p)

return true

Algorithm Union (A, B)

a:= A.first()

b:= B.first()

S:= new sequence

while ! A.isLast(a) do

S.insertLast(a)

a:= A.after(a)

S.insertLast(a)

while ! B.isLast(b) do

S.insertLast(b)

b:= B.after(b)

S.insertLast(b)

removeDups(S)

return S

**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.**

**Algorithm sortRBG(L)**

p:= L.first()

while !L.isLast(p) do

sortBGHelper(L)

sortRBHelper(L)

sortBGHelper(L)

sortRBHelper(L)

return L

**Algorithm sortRBHelper(L)**

r:= L.first() // for position of red

b:=L.after(r) // for position of blue

while !L.isLast(b) do

if r.element() != RED /\ b.element() == RED then

swapElements(r,b)

r:= L.after(r)

b:= L.after(b)

if r,element () == RED then

r:= L.after(r)

b:= L.after(b)

if r.element() != RED /\ b.element() == RED then // for the last position of b

swapElements(r,b)

return L

**Algorithm sortBGHelper(L)**

b:= L.first() // for position of red

g:=L.after(b) // for position of blue

while !L.isLast(g) do

if b.element() != BLUE /\ g.element() == BLUE then

swapElements(b,g)

b:= L.after(b)

g:= L.after(g)

if b,element () == BLUE then

b:= L.after(b)

g:= L.after(g)

if b.element() != BLUE /\ g.element() == BLUE then // for the last position of g

swapElements(b,g)

return L

**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) ⎣*n*/2⎦, 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?**