**Assignment 3**

**Level 1:  
A. Design a pseudo code algorithm to take a Sequence and remove all duplicate elements  
from the Sequence. Is the algorithm the same for both a List or a Sequence? Explain.  
Analyze your algorithm twice, once assuming it is a Sequence and then again assuming it  
is a List. Which ADT is a better choice for this problem? Implement your choice in  
JavaScript.**

Algorithm removeDuplicates(L)

p:= L.first()

while !L.isLast(p) do

q:= L.after(p) // to remove the first element among the duplicates

removeDupsHelper(L,p)

p:= q

Algorithm removeDupsHelper(L,p)

r:= L.after(p)

while !L.isLast(r) do

if r.element() = p.element() then

L.remove(p)

p:= r

r:= L.after(r)

**B. Design an algorithm, isPermutation(A,B) that takes two sequences A and B and  
determines whether or not they are permutations of each other, i.e., same elements  
but possibly occurring in a different order. Hint: assume that A and B do not  
contain duplicates (later we will allow duplicates).  
What is the worst-case time complexity of your algorithm? Justify your answer.  
Implement your algorithm in JavaScript using either the Sequence or the List  
class/program provided.**

Algorithm isPermutation(A,B)

If A.size() != B.size() then

return false

p:= A.first()

while !A.isLast(p) do

if isContained(B,p)

return true

p:= A.after(p)

if isConatined(B,p) // for the last position

return true

return false

Algorithm isConatined(B,p)

q:= B.first()

while !B.isLast(q) do

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

return true

else return false

q:= B.after(q)

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

return true

else return false

**C. Let L be a List of objects colored either red or blue. Design an in-place pseudo  
code algorithm sortRB(L) that places all red objects in list L before the blue  
colored objects. Thus the resulting List will have all the red objects followed by  
the blue 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 inplace.**

Algorithm sortRB(L)

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

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

while !L.isLast(b)

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

swapElements(r,b)

r:= L.after(r)

b:= L.after(b)

if r,element () == RED

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

**Level 2:**

**D. Let L be a List of objects colored either red, green, or blue. Design an in-place  
pseudo code 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 inplace.**

**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

**E. Implement C and D in JavaScript using the LinkedList data structure provided.**