Assignment 9-10

R-4.2 Give a pseudo-code description of the merge-sort algorithm. You can call the merge algorithm as a subroutine.

```
 Algorithm mergeSort(S)
 Input: array S
 Output array S with elements sorted
 n <-- S.length
 Temp <-- new Array of size n
 mergeSortHelper(S, 0, n-1, Temp)

 Algorithm mergeSortHelper(S,lo,hi,Temp)
 Input: arrays S and Temp (work area), and indices lo, hi
 Output: array S with elements between lo and hi in sorted order
 if hi-lo + 1>1 then
 mid <-- floor((lo + hi)/2) …………………………………O(1)
 mergeSortHelper(S, lo, mid, Temp) ……………….O(logn) //dividing array
into two
 mergeSortHelper(S, mid+1, hi, Temp) ……………O(logn)
 merge(S, lo, mid, hi, Temp) ……………………..O(nlogn)
 return
```

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 u B (with no duplicates).

```
 Algorithm removeDupUnion(A, B)
 Input: sequence of A and B with n element.
 output: sorted sequence of A U B.
```

```
S <-- empty sequence

while !A.isEmpty() and !B.isEmpty() do
if A.first().element() < B.first().element()
s.insertLast(A.remove(A.first()))
else if B.first().element() < A.first().element()
s.insertLast(B.remove(B.first()))
else
s.insertLast(A.remove(A.first()))
B.remove(B.first())

if A.size() != 0 then
p<--A
else if B.size() !=0 then
p <-- B
else
return s
while !p.isEmpty() do
s.insertLast(p.remove(p.first()))
return s
```

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?

Answer

O(nlogn) because originally quick sort have the running time of O(n^2) if the element is sorted and we decrease the array size by half the it will be O(n log n) running times because the pivot is selected at the middle of the array.

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.

```
Algorithm electionWinner(S)
    Input: n element sequence.
     Output: Winner ID.
    maxCount <-- 0 ......................................................1
    winner <-- null ...................................................1
        for r <-- 0 to s.size()-1 do .......................................n
            count <-- 0 ................................................1
            for j <-- 0 to s.size()-1 ........................................n^2
                if S.elementAtRank(r)==
            S.elementAtRank(j) .......................n^2
                count <-- count+1 ........................................n^2
            if maxCount < count ...............................................n
            maxCount <-- count ............................................n
             winner <-- S.elementAtRank(r) .................................n
    return winner .........................................................1
```

The running time of this algorithm is O(n^2)

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 getKey(e)
        if color(e) = RED then
              return 0
       else if color(e) = BLUE then
        return 1
        else
              return 2



Algorithm sortRBG(L)
        Q <- new Priority Queue
       Q.insertItem(getKey(p), p.element())
        while ! L.isEmpty() do
       e <- L.remove(L.first())
       Q.insertItem(getKey(e), e)
       while ! Q.isEmpty() do
       L.insertLast(Q.removeMin())
```