4.5

```
Algorithm joinListNoDups(A, B)
    S := new Sequence()
O1  if !A.isEmpty and !B.isEmpty then
O1      if a.first().element() < b.first().element
O1          s.insertLast(a.first())   a.remove(a.first)
O1      else s.insertLast(b.first())  b.remove(B.first)
O1  else if !A.isEmpty then
O1      s.insertLast(a.first())       a.remove(a.first)
O1  else s.insertLast(B.first())      B.remove(b.first)
On  return merge(A, B, S)

On Algorithm merge(A, B, S)
On  if(A.isEmpty and B.isEmpty) return s
On  if(!A.isEmpty and !B.isEmpty) then
On      if(A.first.element < B.first.elemen then
On          helperDelete(A, S)
On      else helperDelete(B, S)
On  else if !A.isEmpty then
On      helperDelete(A, S)
On  else helperDelete(B, S)
On  merge(A, B, S)
O1 Algorithm helperDelete(S1, S2)
O1      if S1.first.element() != S2.last().elemen()
O1          S2.insertLast(S1.first())
O1      S1.remove(S1.first())
```

Algorithm getWiner (S, C)

sorted := mergeSort(S, C)

lastCandidate := null

lastCandidateVotes := 0

currentId := null

currentVotes := 0

iterator

while (sorted.elements().hasNext())

candidateId := iterator.next()

if (currentId == null) then

currentId := candidateId

~~currentVotes := currentVotes + 1~~.

if (currentID == candidateId) then

currentVotes := currentVotes + 1

else if lastCandidateVotes < currentVotes then

lastCandidateVotes := currentVotes

lastCandidateId := currentId

currentId = candidateID

currentVotes = 1

return lastCandidateID;

4.9 . In QuickSort if we change the pivot in the middle of the list

Take always → $O(n \log n)$

```
Algorithm InPlaceQuickSort(S, lower, higuest)
    If lower < higuest then
        (P1, P2) := InPlacePartition(S, lower, higuest)
        inPlaceQuickSort(S, lower, P1-1)
        inPlaceQuickSort(S, P2+1, higuest)

Algorithm inPlacePartition(S, lower, max)
    p : random Number between Lower and Max
    pivot := S.elementAtRank(p)
    maxP := max
    minP := max
    S.swapElements(S.atRank(max), S.atRank(p))
    init := lower and last := max-1
    while init ≤ last do
        while last ≥ init ∧ S.elemAtRank(last) ≥ pivot do
            last := last - 1
        while init ≤ K ∧ S.elemAtRank(init) < pivot do
            init := init + 1
        while last ≥ 0 && init < minP && (S.elementAtRank(init) == pivot
        or S.elementAtRank == pivot) do
            if S.elemAtRank(init) == pivot then
                minP := minP - 1 and maxP := maxP + 1
                S.insertAfter(S.atRank(minP), S.atRank(init))
                continue;
            if S.elementAtRank(last) == pivot then
                minP := minP - 1 and maxP := maxP + 1
                S.insertAfter(S.atRank(minP), S.atRank(last))
                continue
            if j < k then
                S.swapElements(S.atRank(init), S.atRank(last))
    return(minP, maxP)
```

```
Algorithm sortRBG(L)
    curP := L.first()
    nextNotRed := L.first()
    last := L.last().
    while (curP != last) then
        if curP.element() != redObject
            curP := L.after(curP)
        else
            L.swapElement(curP, nextNotRed)
            nextNotRed := L.after(nextNotRed)

    if curP == redObject then
        L.swap(curP, nextNotRed)
        nextNotRed := L.after(nextNotRed)
    curP := nextNotRed
    nextGreen := curP

    while curP != last then
        if cur.element() != greenObject
            curP := L.after(curP)
        else L.swapElement(curP, nextGreen)
            nextNotBlue := L.after(nextNotBlue)
```