# R 4.5

Merge(A, B)

S <- Sequence

While !A.isEmpty() and !B.isEmpty() do

if A.first() = B.first() then

B.remove(B.first())

else if A.first() < B.first() then

if S.last() != A.first()

S.insert(A.fist())

A.remove(A.first())

else if A.first() > B.first() then

if S.last() != B.first()

S.insert(B.fist())

B.remove(B.first())

return S

# R 4.9

If the middle element of a sorted sequence ‘S’ is selected as a pivot then, size of both Lower Partition (L) and Greater partition (G) will be always at least S/4. So, the height of the quick-sort tree will be **log4/3n.** The running time for each depth is **O(n)**

Therefore,total running time of quick sort will be **O(nlog4/3n)🡪O(nlogn).**

# C 4.10

electionWinner(S)

D <- Dictionary<int, int>

for i=0 to S.size() do

if D.containsKey(S[i])

D(S[i])++

else

D.Add(S[i], 1)

maxV <- 0

maxId <- 0

foreach entry in D

if entry.value > maxV

maxV = entry.value

maxId = entry.key

return maxId

**Algorithm sortRBG(L)**

if L.size() <= 1 then return L // 1

p <- L.first() // 1

while p != L.last() /\ p.element() = R do // n

p <- L.after(p) // n

nextRed <- p // 1

while p != L.last() do // n

if p.element() = R then // n

L.swapElements(p, nextRed) // n

nextRed <- L.after(nextRed) // n

p <- L.after(p) // n

if p.element() = R then // 1

L.swapElements(p, nextRed) // 1

if ! L.isLast(nextRed) then // 1

nextRed <- L.after(nextRed) // 1

p <- nextRed // 1

nextGreen <- L.last() // 1

while p != nextGreen /\ nextGreen.element() = G do // n

nextGreen <- L.before(nextGreen) // n

while p != nextGreen do // n

if p.element() = G then // n

L.swapElements(p, nextGreen) // n

nextGreen <- L.before(nextGreen) // n

p <- L.after(p) // n