R-4.2

Algorithm mergeSort(S, C)

Input sequence S with n elements, comparator C

Output sequence S sorted according to C

if S.size() > 1 then

(S1, S2) ← partition(S, n/2)

mergeSort(S1, C)

mergeSort(S2, C)

S ← merge(S1, S2, C)

Algorithm merge(A, B, C)

Input sequences A and B with n/2 elements each, comparator C

Output sorted sequence of A and B

S ← empty sequence

while ¬A.isEmpty() ∧ ¬B.isEmpty() do

if C.isLessThan(B.first().element(), A.first().element()) then

S.insertLast(B.remove(B.first()))

else

S.insertLast(A.remove(A.first()))

while ¬A.isEmpty() do

S.insertLast(A.remove(A.first()))

While ¬B.isEmpty() do

S.insertLast(B.remove(B.first()))

return S

R-4.5

Algorithm specialMerge(A, B, C)

Input sequences A and B with n/2 elements each, comparator C

Output sorted sequence of A and B

S ← empty sequence

while ¬A.isEmpty() ∧ ¬B.isEmpty() do

if C.isLessThan(B.first().element(), A.first().element()) then

S.insertLast(B.remove(B.first()))

Else if C.isEqual(B.first().element(), A.first().element()) then

S.insertLast(B.remove(B.first())

A.remove(A.first())

else

S.insertLast(A.remove(A.first()))

while ¬A.isEmpty() do

S.insertLast(A.remove(A.first()))

While ¬B.isEmpty() do

S.insertLast(B.remove(B.first()))

return removeRepeated(s)

Algorithm removeRepeated(s)

For i←0 to s.size()-2 do

If s.elemAtRank(i) = s.elemAtRank(i+1) then

s.removeAtRank(i)

i ← i-1

return s

R-4.9

Since the list is already sorted then best running time will be the case, which is O(n log(n))

C-4.10

1. Sort the sequence S using Heap-Sort. The running time should be O(n log n)
2. Initialize two variables currentCount and maxCount
3. Iterate on the sorted sequence, and increment the currentCount until the ID changes, then compare currentCount with maxCount. If currentCount is greater than set maxCount as currentCount. The running time for the operation is O(n)

The total running time is O(n log n)