Aufgabe 5

Geben Sie Pseudocode für eine Merge Methode an, die zwei sortierte einfach verkettete zyklische Listen als Parameter annimmt und diese in linearer Zeit zu einer einzelnen sortierten Liste zusammenfügt. Die ursprünglichen Listen dürfen dabei zerstört werden und es soll nur konstant viel zusätzlicher Speicher verwendet werden.
Wieso ist die Zeitkomplexität quadratisch statt linear, wenn der Merge Pseudocode aus Kapitel 2, Seite 32 im Buch direkt verwendet wird, die Felder A,L,R aber durch verkettete Listen ersetzt werden?

```
merge (list1, list2)
        listNew

        // Randfälle: eine Liste leer
        if(list1.head == NIL) return list2; endif
        if(list2.head == NIL) return list1; endif

        //initialize list: Idee - immer das letzte Element der listNew haben und bei diesem dann die
        neuen Elemente einfügen —> dazu muss die listNew erst einmal ein Element enthalten
        if(list1.head <= list2.head)
                listNew.head = list1.head
                deque(list1)
                //für zyklische liste
                listNew.head.next = listNew.head.
        else
                listNew.head = list2.head
                deque(list2)
                //für zyklische liste
                listNew.head.next = listNew.head.
        endif

        // tail von listNew tracken, um in O(1) anfügen zu können
        listNewTail = listNew.head
        while (list1.head ≠ NIL && list2.head ≠ NIL) //Handhabung wie im Buch, head = NIL => leer
                if (list1.head <= list2.head)
                        //Das element list1.head wird in die newList verlinkt
                        listNewTail.next = list1.head
                        DEQUEUE(list1)
                        // zyklische Liste. Bemerkung:  list1.head darf nicht verändert werden vor
                        dequeue, sonst zerstört man list1 und dequeue schlägt fehlt
                        listNewTail.next.next = listNew.head
                        listNewTail = listNewTail.next; // track tail.
                 else
                        //Das element list2.head wird in die newList verlinkt
                        listNewTail.next = list22.head
                        DEQUEUE(list2)
                        // zyklische Liste. Bemerkung:  list1.head darf nicht verändert werden vor
                        dequeue, sonst zerstört man list1 und dequeue schlägt fehlt
                        listNewTail.next.next = listNew.head
                        listNewTail = listNewTail.next; // track tail.
                endif
        endwhile
        while (list1.head ≠ NIL)
                //Das element list1.head wird in die newList verlinkt
```

```
        listNewTail.next = list1.head
        DEQUEUE(list1)
        // zyklische Liste. Bemerkung:  list1.head darf nicht verändert werden vor
        dequeue, sonst zerstört man list1 und dequeue schlägt fehlt
        listNewTail.next.next = listNew.head
        listNewTail = listNewTail.next // track tail.

endwhile
while (list2.head ≠ NIL)
        //Das element list2.head wird in die newList verlinkt
        listNewTail.next = list22.head
        DEQUEUE(list2)
        // zyklische Liste. Bemerkung:  list1.head darf nicht verändert werden vor
        dequeue, sonst zerstört man list1 und dequeue schlägt fehlt
        listNewTail.next.next = listNew.head
        listNewTail = listNewTail.next; // track tail.

endwhile
```