

Serie 5Theoretische Aufgaben**1. Reihenfolge umkehren**Nicht rekursiv, Zeit $\Theta(n)$, einfach verkettet, n Elemente

```
LIST-REVERSE(L)
    point := NIL
    current := L.head
    WHILE current != nil
        next := current.point
        current.point := point
        point := current
        current := next
    L.head = point
```

//head wird zu tail werden und auf NIL zeigen
//aktuelle Stelle, Beginn bei Head

//Zwischenspeicher Zeiger von akt. El.
//Zeiger überschrieben (1. NIL)
// Zeiger von nächstem El. (akt. El.)
// auf nächstes Element
//Kopf = letztes El. Auf das gezeigt wurde

2. Warteschlangee.next ist ein Element oder NIL, e.key ist ein Wert
Nil.next und NIL.key geht nicht

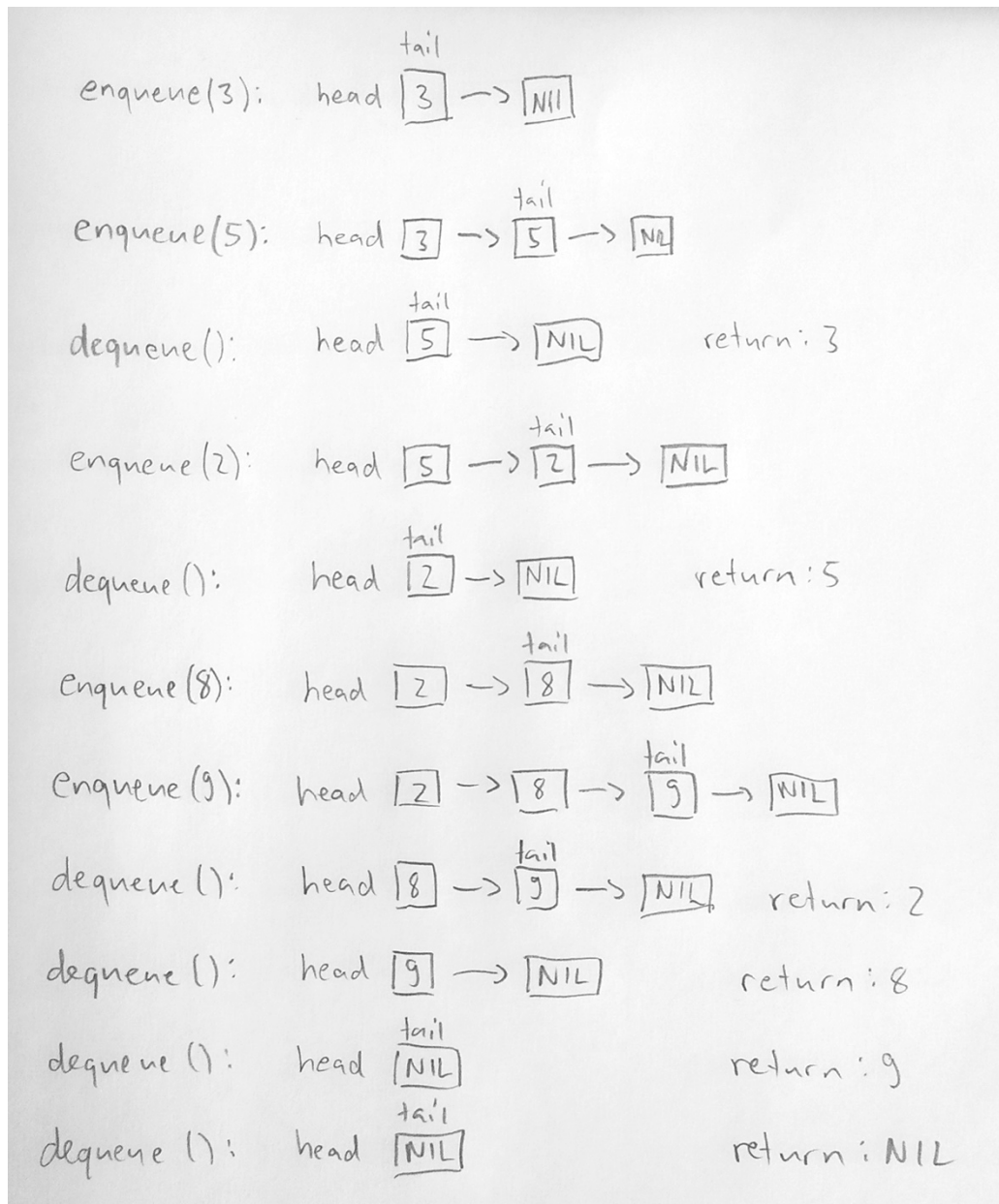
```
Enqueue(Queue Q, Element e)
    e.next := NIL
    If Q.head = NIL
        Q.head := e
    Else
        Q.tail.next := e
    Q.tail := e
```

//neues Element zeigt auf NIL
//e ist erstes Element

//aktueller tail zeigt auf e
//e wird tail


```
Deque(Queue Q)
    If Q.head = NIL
        Return NIL
    ELSE
        e := Q.head.key
        If Q.head.next = NIL
            Q.tail := NIL
        Q.head = Q.head.next
    return e
```

//Q.head.key geht nicht wenn Q.head = NIL
//Queue nicht leer
//Zwischenspeicher Ausgabe (e kein Element)
//letztes Element
//tail = NIL
//Element oder NIL



3. Schlüssel aller Knoten eines gerichteten Baums: rekursiv

```

GET-KEYS (n)
IF n == NIL
    RETURN
ELSE
    PRINT (n.key)
    GET-KEYS (n.left-child)
    GET-KEYS (n.right-sibling)

```

4. Schlüssel aller Knoten eines gerichteten Baums: nicht rekursiv

```
S := NEW STACK();
S.PUSH(root)

TRAVERSE_TREE(Node: root)
WHILE S != NIL
current := Node
    WHILE current != NIL
        S.PUSH(Node.left_child)
        current := current.left_child
    END

    last_node := S.POP()
    PRINT(last_node.key)
END
```

5. Merge

```
MERGE (L, R, M)

M.head := NIL

IF L.head != NIL | R.head != NIL

    IF (L.head.key <= R.head.key) | (R.head = NIL && L.head != NIL)
        M.head := L.head
        L.head := L.head.next
    ELSE
        M.head := R.head
        R.head := R.head.next

    M.tail := M.head.next

WHILE L.head != NIL | R.head != NIL
    IF (L.head.key <= R.head.key) | (R.head = NIL && L.head != NIL)
        M.tail := L.head
        M.tail := M.tail.next
        L.head := L.head.next
    ELSE
        M.tail := R.head
        M.tail := M.tail.next
        R.head := R.head.next

M.tail.next := M.head    //zyklisch
Return M
```