Eingabe: Array A von Datensätzen

Ausgabe: Sortiertes Array

SELECTIONSORT(A)

```
for i := 1 to n − 1 do
    ref := a_i; pos := i;
    for j := i + 1 to n do
        if a_j < ref then
        pos := j;
    end
    SWAP(i, pos)
end
```

Eingabe: Array A von Datensätzen
Ausgabe: Sortiertes Array
INSERTIONSORT(A)

for $i := 1$ **to** n **do**
    BINARYINSERT(A, a_i, 1, $i - 1$)
end

Eingabe: Array A, Element e, obere Grenze i, untere Grenze j
Ausgabe: Array A
BINARYINSERT(A, e, i, j)

if $i > j$ **then**
    INSERT(A, e, i)
    return A
end
INSERT(A, e, BINARYSEARCH(A, e, i, j))

Eingabe: Array A, Element e, obere Grenze i, untere Grenze j
Ausgabe: Index, vor dem e eingefügt werden soll
BINARYSEARCH(A, e, i, j)

$l := i$; $r := j$;
while $l < r$ **do**
    $p := \lfloor \frac{l+r}{2} \rfloor$;
    if $e \geq a_p$ **then** $l := p + 1$;
        else $r := p$;
end
return l

1 Heapsort

Eingabe: Heap H, v Knoten
Ausgabe: Heap H
REHEAP(H, v)

$w := v$;
while Heap-Eigenschaft in w nicht erfüllt **do**
    Sei v^* dasjenige Kind mit dem größeren Schlüssel;
    Vertausche w und v^*;
    $w := v^*$;
end

Eingabe: Heap H
Ausgabe: das größte Element von H
DELETEMAX(H)

1. Sei r die Wurzel von H;
2. Sei k der in r gespeicherte Schlüssel;
3. Sei l das rechteste Blatt im untersten Level;
4. Kopiere den Schlüssel in l nach r;
5. Lösche das Blatt l;
6. REHEAP(H, r);
7. **return** k

Eingabe: Array A
Ausgabe: Heap H
CREATEHEAP(A)

```
Konstruiere einen Baum H mit leeren Knoten;
Fülle jeden Knoten mit einem Schlüssel aus A;
for l := Höhe(H) to 1 by -1 do
    foreach v auf Level l do
        REHEAP(H, v);
    end
end
return H
```

Eingabe: Array A
Ausgabe: sortiertes Array
HeapSort(A)

```
H := CreateHeap(A);
for i := 1 to Länge(A) do
    output(DeleteMax(H))
end
```

2 Quicksort

Eingabe: Array A, Index i
Ausgabe: Arrays B und C mit $B_j < A_i$ und $A_i \leq C_k$ für alle j, k
PARTITION(A, i)

1. $\dots$

Eingabe: Array A mit n Elementen
Ausgabe: sortiertes Array
QUICKSORT(A)

1. (B, C) := PARTITION(A, n);
2. B' :=QUICKSORT(B);
3. C' :=QUICKSORT(C);
4. **return** APPEND((B', C'))

Eingabe: Array A mit $|A| = n$, Index $1 \leq p \leq n$
Ausgabe: Arrays B und C mit $B_j < A_p$ und $A_p \leq C_k$ für alle j, k
PARTITION(A, p)

```
i := 1; j := n − 1;
pivot := A_p;
SWAP(A_p, A_n);
while true do
    while i < j and A_i < pivot do i++;
    while i < j and A_j ≥ pivot do j−−;
    if i ≥ j then break;
    SWAP(A_i, A_j);
end
SWAP(A_i, A_n);
return (A_{1,...,i−1}, A_{i,...,n});
```

Eingabe: Array A mit n Elementen, Index k
Ausgabe: Element vom Rang k
QuickSelect(A, k)

$(B, C) :=$ Partition(A, n);
$r :=$ length(B) $+ 1$;
if $k = r$ **then return** C_1;
if $k < r$ **then return** QuickSelect(B,k);
else return QuickSelect(C,$k - r + 1$);