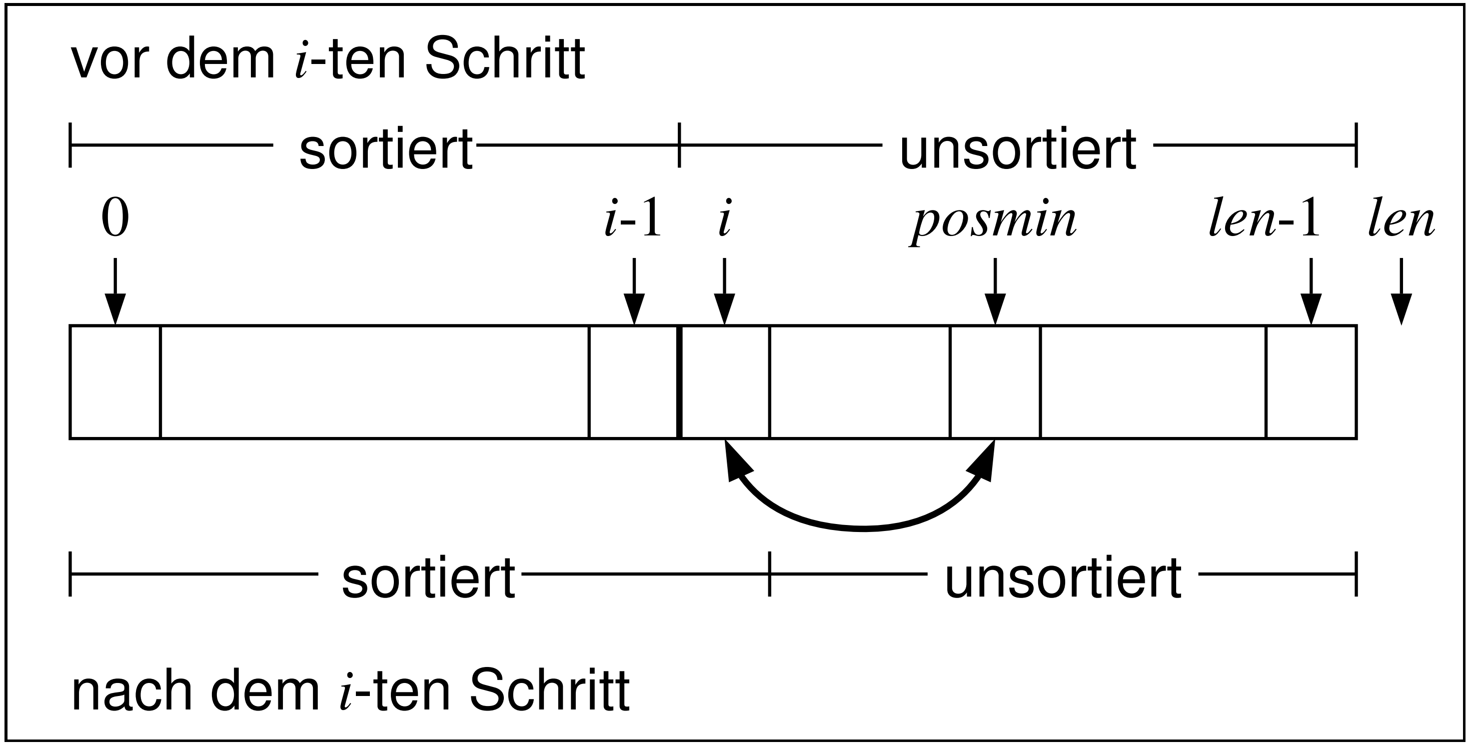
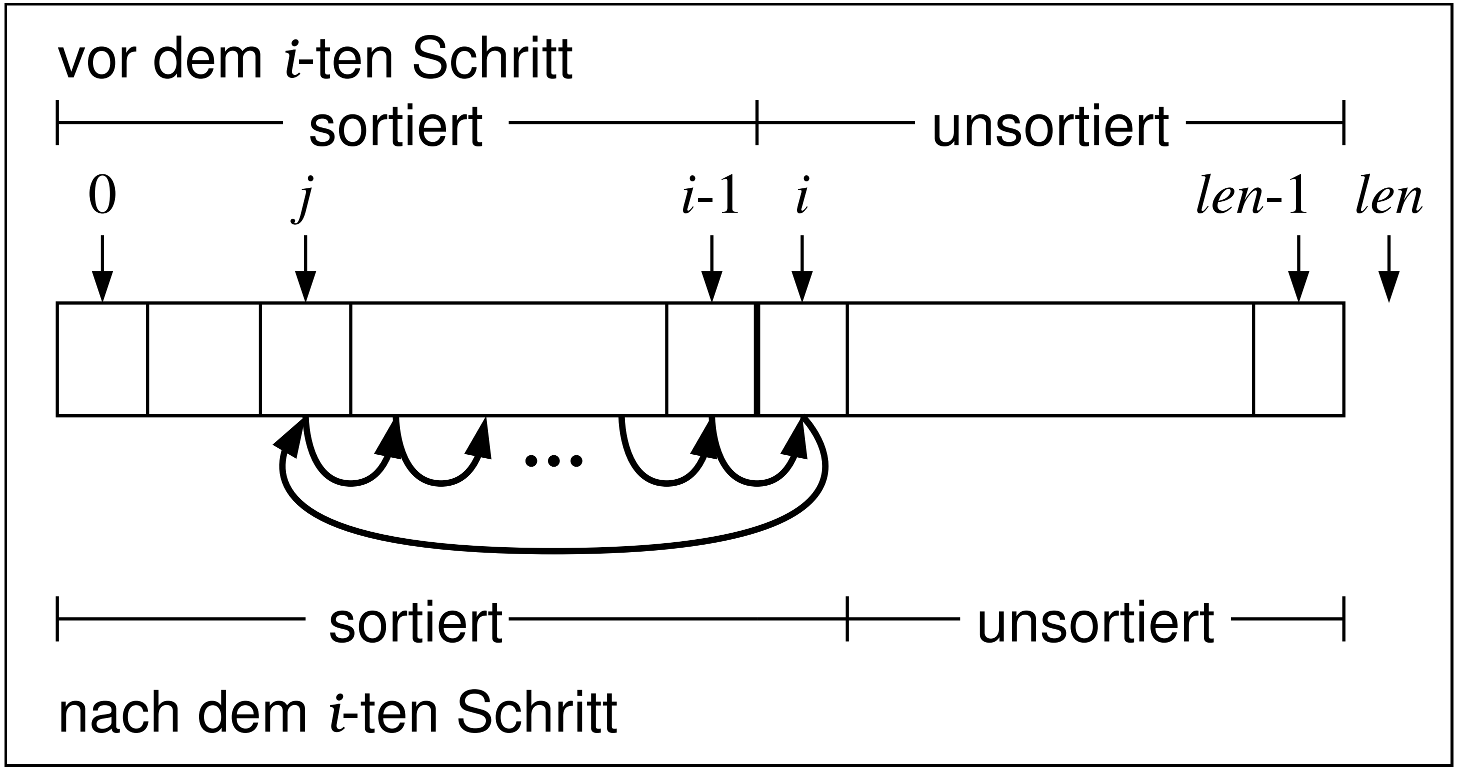
SelectionSort



Satz: Selectionsort benötigt zum Sortieren von n Elementen genau $\binom{n}{2}$ Vergleiche.

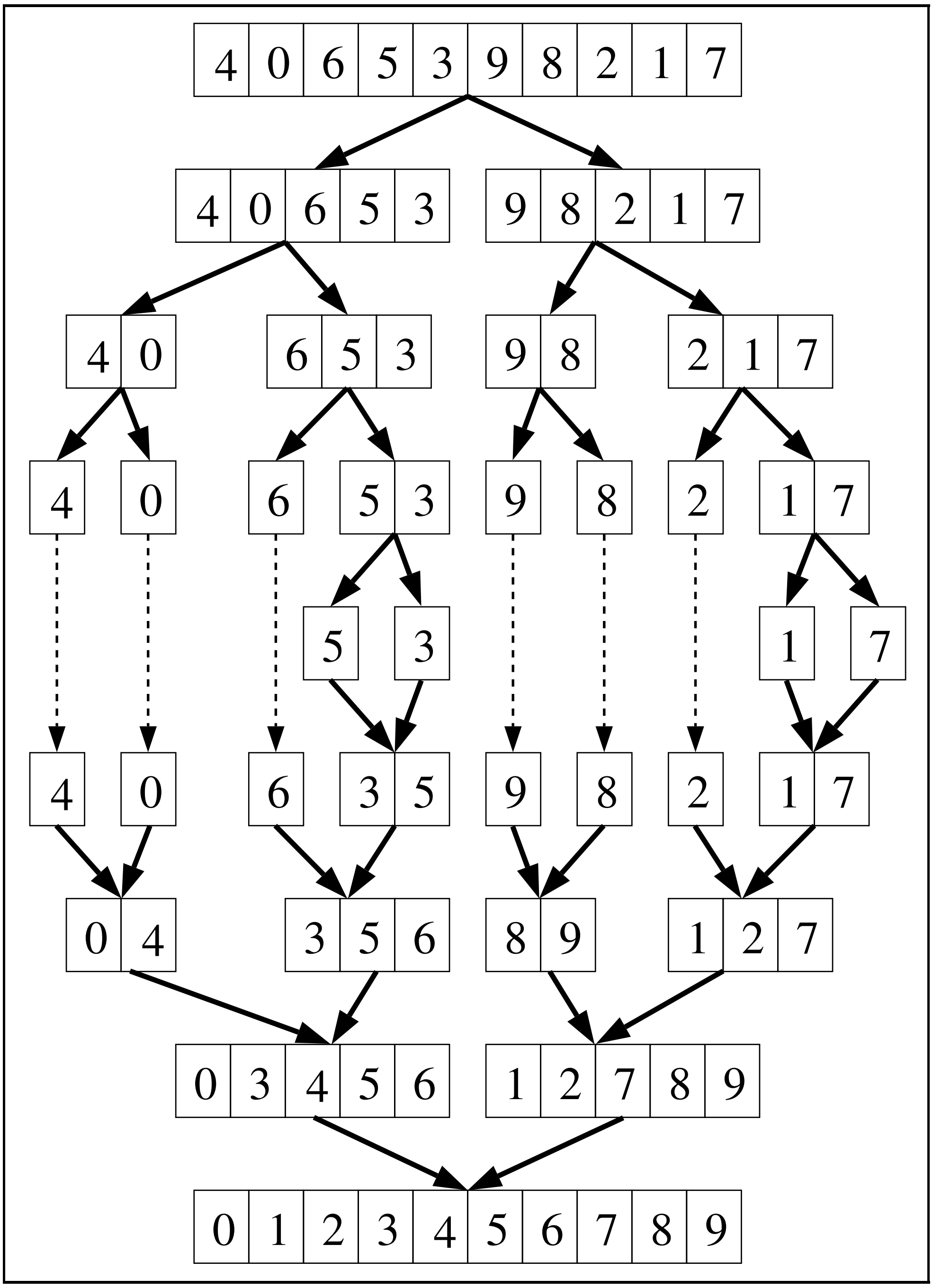
InsertionSort



Satz 1: Insertionsort benötigt zum Sortieren von n Elementen maximal $\binom{n}{2}$ Vergleiche.

Satz 2: Der auf der binären Suche basierende Insertionsort (beim Einsortieren des nächsten Elementes in den bereits sortierten Teil) benötigt zum Sortieren von n Elementen maximal $n\lceil\log(n+1)\rceil$ Vergleiche.

MergeSort



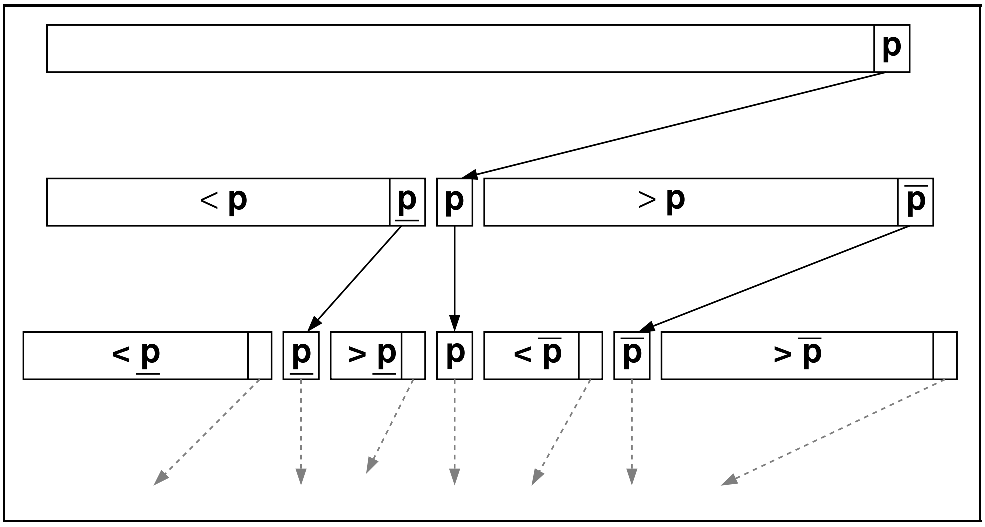
Ablauf des rekursiven Mergesort auf der Folge

(4,0,6,5,3,9,8,2,1,7)

Satz: Die rekursive Version von Mergesort sortiert ein Feld der Länge n mit maximal $n \cdot \lceil \log(n) \rceil$ Vergleichen.

Beweis: vgl DSI

QuickSort



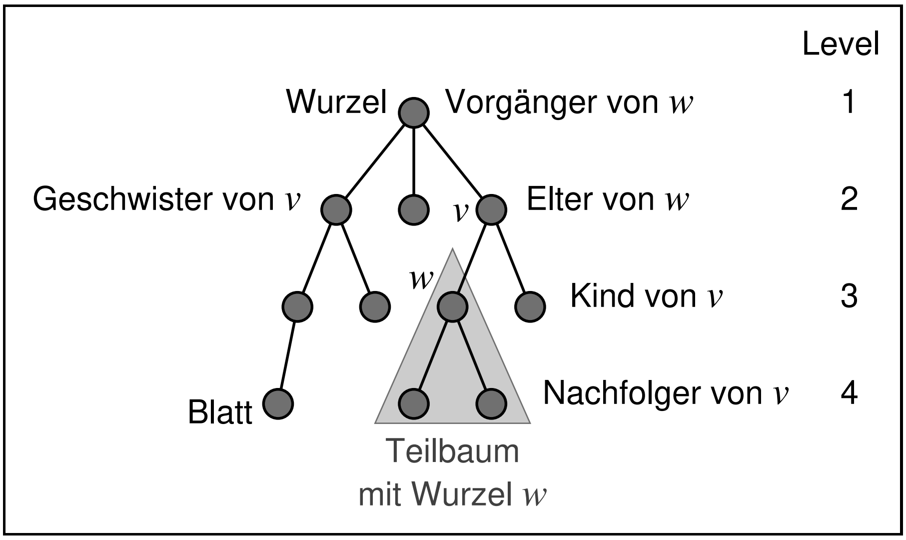
Satz: Quicksort benötigt zum Sortieren eines Feldes der Länge n maximal $\binom{n}{2}$ viele Vergleiche.

Satz: Quicksort benötigt zum Sortieren eines Feldes der Länge n mit *durchschnittlich* nur $2\ln(2)\,n\log(n)+O(n)$ viele Vergleiche.

Beweis: vgl DSII

HeapSort (1)

Notation:



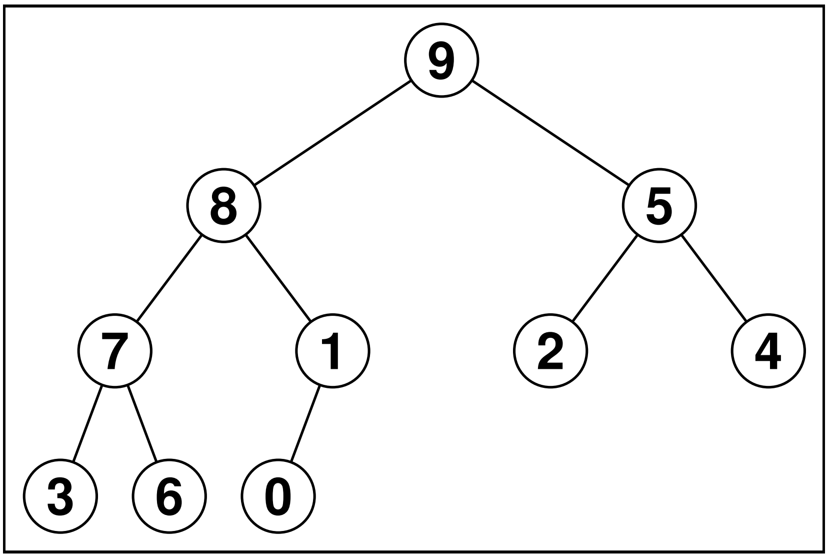
Definition eines *Heaps*:

– Alle inneren Knoten bis auf maximal einen haben genau zwei Kinder.

– Alle Knoten mit weniger als zwei Kindern (insbesondere also die Blätter) befinden sich auf den beiden größten Leveln;

– Die Blätter im größten Level des Baumes sind von links nach rechts aufgefüllt.

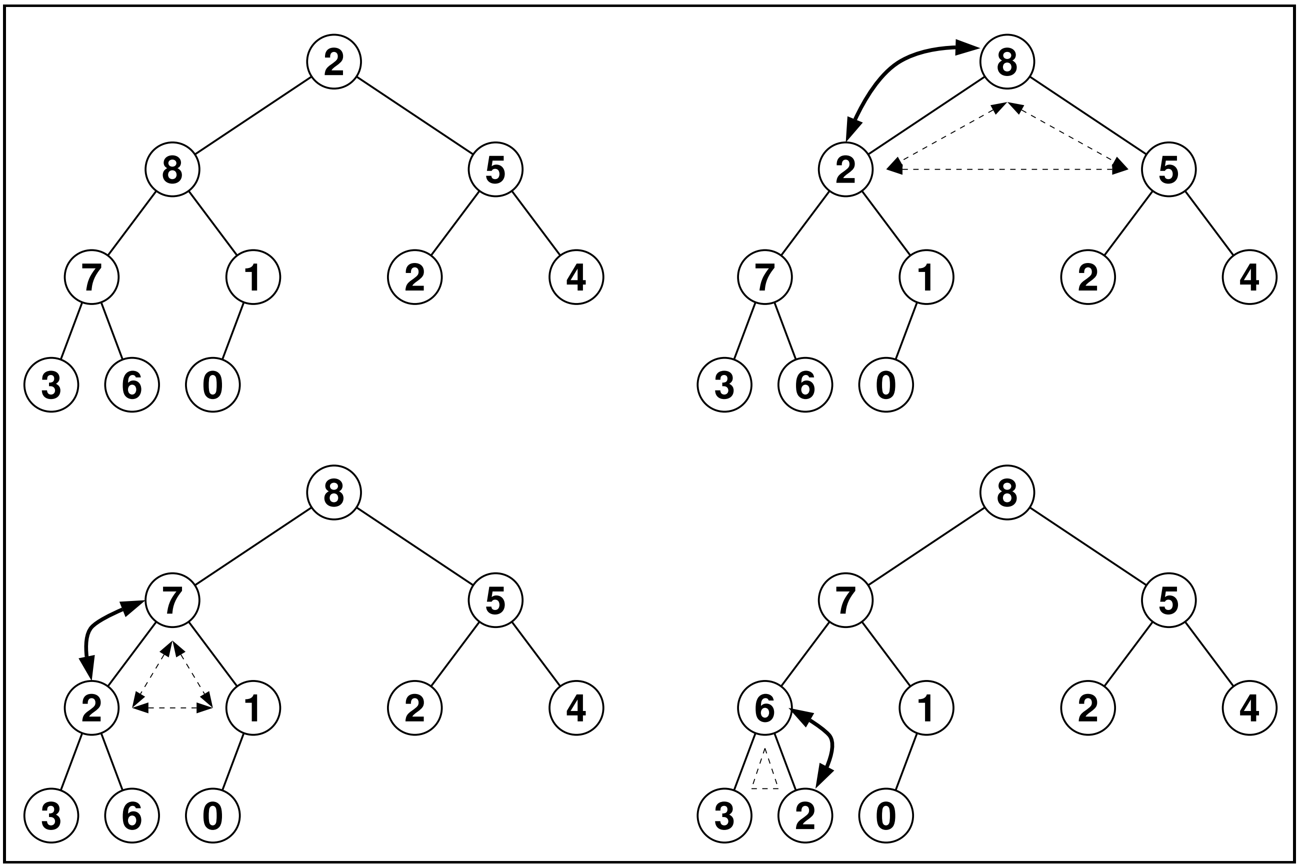
– Für jeden Knoten gilt: alle Nachfolger haben einen höchstens gleich großen Schlüssel.



HeapSort (2)

Algorithmus REHEAP(h) zum Einfügen der Wurzel in einen ansonsten korrekt sortierten Heap:

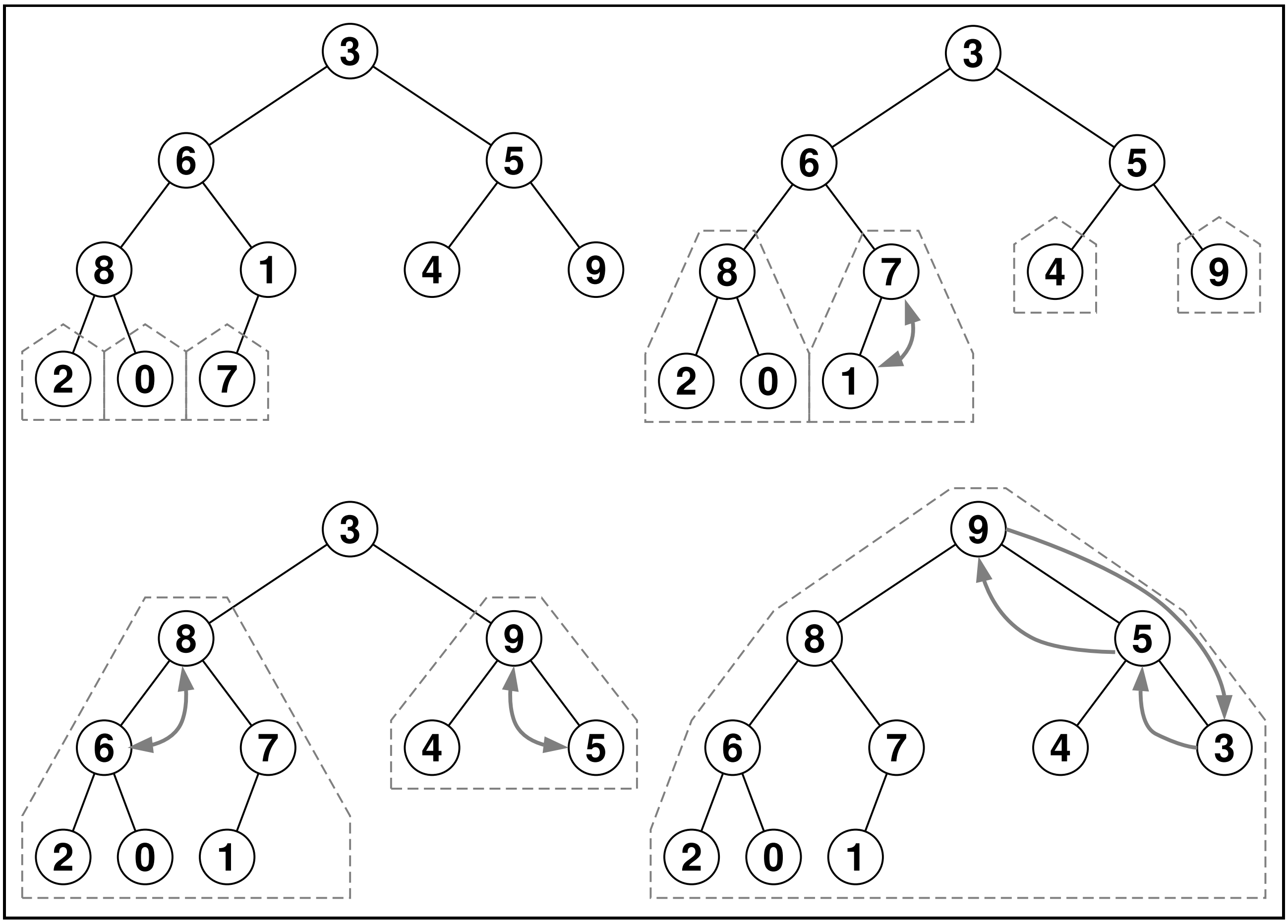
> Sei v die Wurzel des Heaps h;
>
> **while** (Heap-Eigenschaft in v nicht erfüllt)
>
> > Sei v^* das Kind mit dem größeren Schlüssel;
> >
> > Vertausche die Schlüssel in v und v^* und setzte $v = v^*$;



HeapSort (3)

Algorithmus CreateHeap(h) zum Erstellen eines Heaps für n beliebige Elemente:

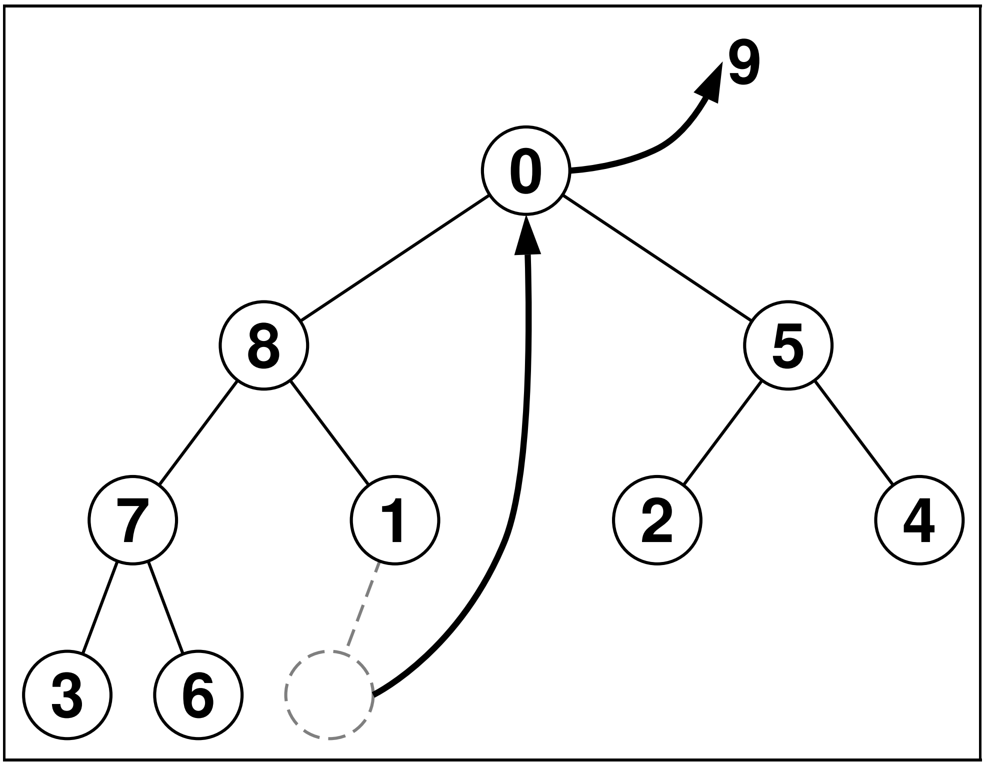
for ℓ := Tiefe des Baumes **down to** 1 **do**
    for each Knoten v auf Level ℓ **do**
        reheap(Baum H mit Wurzel v);



HeapSort (4)

Algorithmus DELETEMAX(h) zum Löschen der Wurzel:

> Sei r die Wurzel des Heaps h und sei k der in r gespeicherte Schlüssel;
> Sei ℓ das rechteste Blatt im untersten Level;
> Kopiere den Schlüssel in ℓ in die Wurzel r;
> Lösche das Blatt ℓ und dekrementiere $heap\_size$;
> reheap(h);



HeapSort (5)

Algorithmus HEAPSORT (sortiert absteigend):

> CreateHeap(h);
>
> **while** h nicht leer **do**
>
> > Gebe Schlüssel der Wurzel aus;
> >
> > DeleteMax(h);

Satz: Mittels Heapsort kann ein Feld der Länge n mit höchstens $2n\log(n) + 5/2n$ vielen Vergleichen sortiert werden.

Bemerkungen:

- Mit einer von Carlssons beschriebenen Variante von Heapsort kann man ein Feld der Länge n mit maximal $n\log(n) + \mathcal{O}(n\log\log(n))$ vielen Vergleichen sortieren.
- HeapSort ist ein sogenanntes in-situ Verfahren, d.h. es benötigt nur konstant viele zusätzliche Speicherplätze.