

Sortieren mit Halde

Fachdidaktik Informatik
Referent: Hien Nguyen

Fachdidaktik DO 18.6

AMT MO 15.6

HCI 1 SO 14.6

HCI 2 SO 14.6

AB MO 15.6


SE-Projektmeeting FR 12.6

SE Hibernate SQLQuerys Arbeit MI 17.6

SE Hibernate SQLQuerys Meeting FR. 19.6

Stochastik Prog SO 14.6

Stochastik Theo MI 16.6



Fachdidaktik DO 18.6
AMT MO 15.6
HCI 1 SO 14.6
HCI 2 SO 14.6
AB MO 15.6
SE-Projektmeeting FR 12.6
SE Hibernate SQLQuerys Arbeit MI 17.6
SE Hibernate SQLQuerys Meeting FR. 19.6
Stochastik Prog SO 14.6
Stochastik Theo MI 16.6



SE-Projektmeeting FR 12.6
HCI 1 SO 14.6
HCI 2 SO 14.6
Stochastik Prog SO 14.6
AMT MO 15.6
AB MO 15.6
Stochastik Theo MI 16.6
SE Hibernate SQLQuerys Arbeit MI 17.6
Fachdidaktik DO 18.6
SE Hibernate SQLQuerys Meeting FR. 19.6

Wiederholung & Datenstruktur

- ▶ Datenstruktur: **Heap (Halde)**, die einen binären Baum mit bestimmten Eigenschaften bezeichnet:



Wiederholung & Datenstruktur

- ▶ Datenstruktur: **Heap (Halde)**, die einen binären Baum mit bestimmten Eigenschaften bezeichnet:
 - ▶ Baum ist vollständig, d.h. jede Blattebene ist von links nach rechts gefüllt

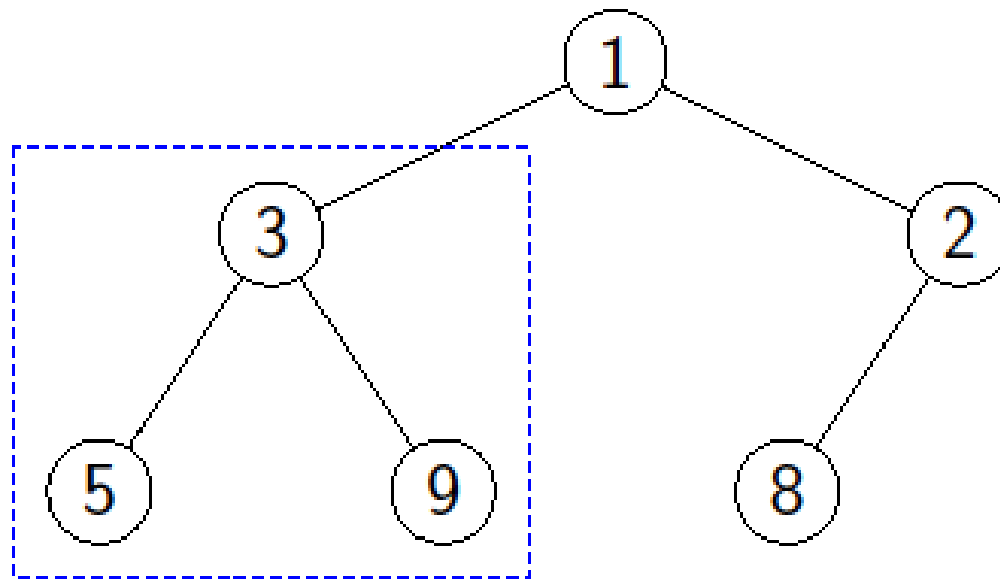


Wiederholung & Datenstruktur

- ▶ Datenstruktur: **Heap (Halde)**, die einen binären Baum mit bestimmten Eigenschaften bezeichnet:
 - ▶ Baum ist vollständig, d.h. jede Blattebene ist von links nach rechts gefüllt
 - ▶ Schlüssel eines Elternknoten ist kleiner oder gleich (größer oder gleich) dem Schlüssel seiner 2 Kinder
 - ▶ \leq : **Min-Heap**, d.h. $arr[parent(i)] \leq arr[i]$
 - ▶ \geq : **Max-Heap**, d.h. $arr[parent(i)] \geq arr[i]$
 - ▶ Wird als **Heap-Eigenschaft** bezeichnet

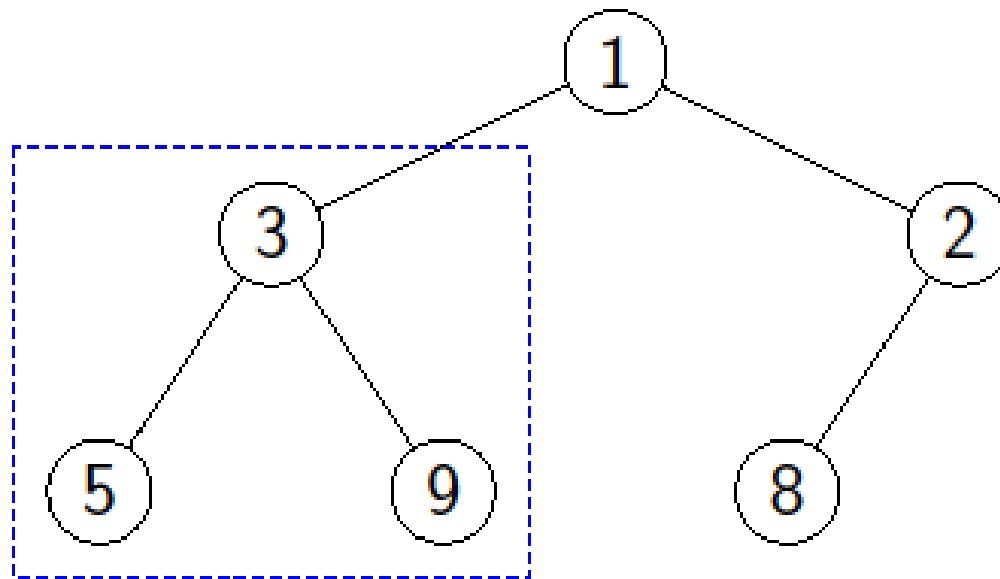


Beispiele

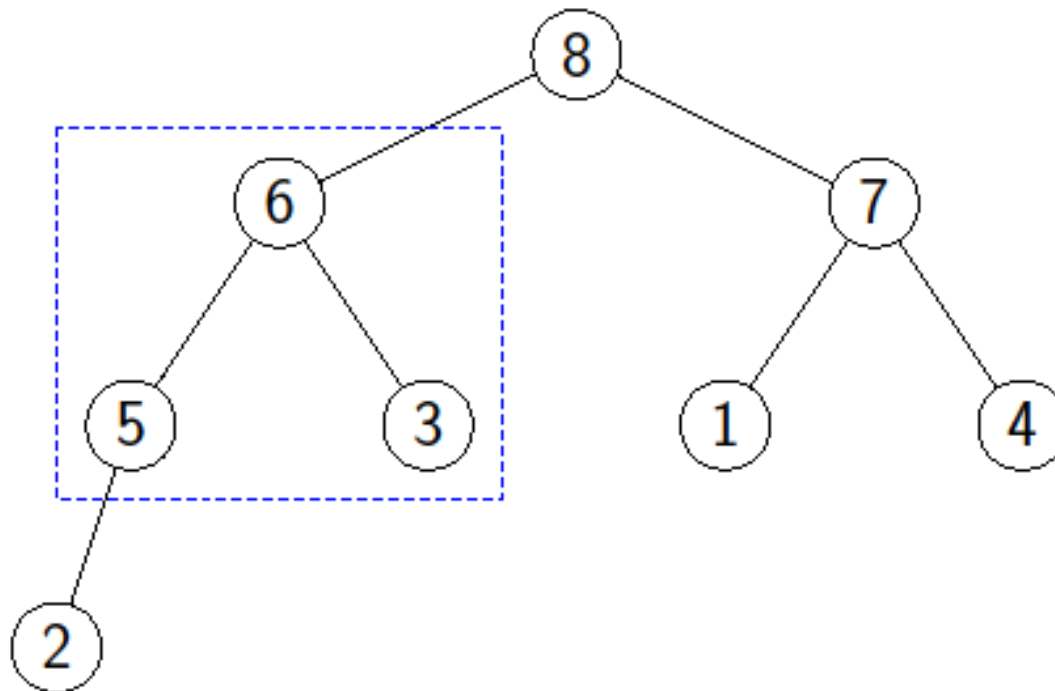


Beispiele

Min - Heap

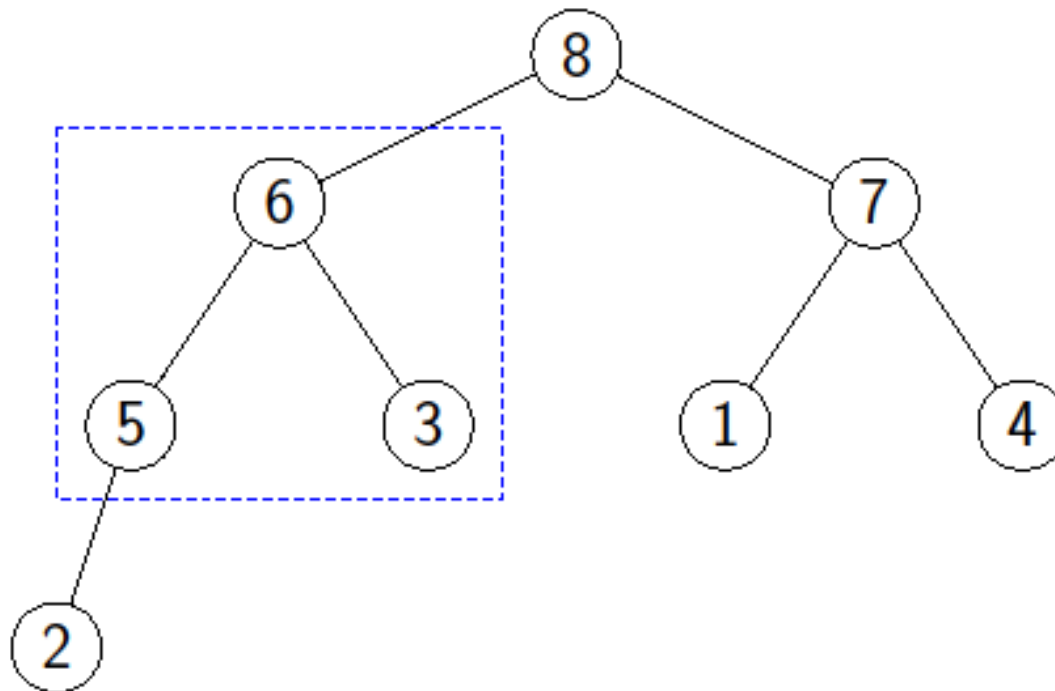


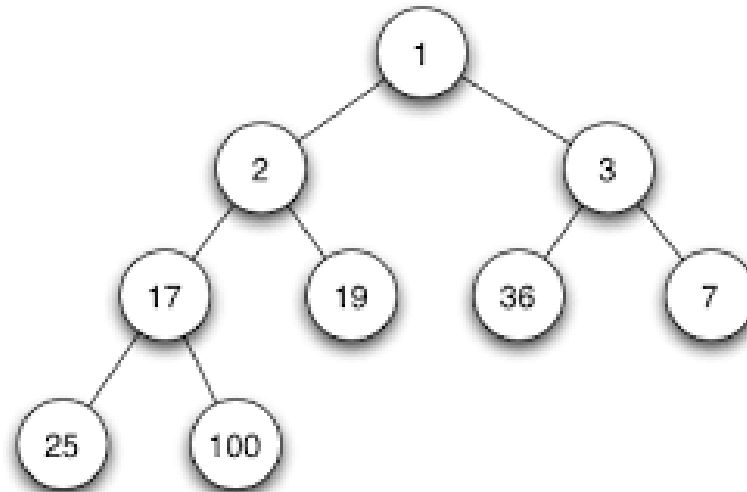
Beispiele



Beispiele

Max - Heap





Sortieren mit Halde

Fachdidaktik Informatik
Referent: Hien Nguyen

Anwendung und Eigenschaften

- ▶ Das **kleinste bzw. größte** Element befindet sich immer in der Wurzel
- ▶ **Anwendung:**
 - ▶ Priority Queue (Warteschlange),
 - ▶ Sortierung Heapsort



Grundidee von Heapsort

- ▶ Beim Entfernen der Wurzel wird beim:
 - ▶ Max-Heap: größte Element
 - ▶ Min-Heap: kleinste Element



Entfernen der Wurzel

- ▶ **Problem:** Durch Entfernen der Wurzel entsteht eine „Lücke“ im Baum – die Heap-Eigenschaft wird verletzt
- ▶ **Lösung:**
 - ▶ Bewege letzte Element des Heaps zur Wurzel
 - ▶ Vergleiche mit Nachfolgeelementen und „versickere (percolate)“ durch Tauschen der Nachfolger bis Heap-Eigenschaft wiederhergestellt ist



Grundidee von Heapsort

- ▶ Beim Entfernen der Wurzel wird beim:
 - ▶ Max-Heap: größte Element
 - ▶ Min-Heap: kleinste Element
- aus dem Heap genommen und durch **Versickern** die Heap-Eigenschaft wiederhergestellt.



Grundidee von Heapsort

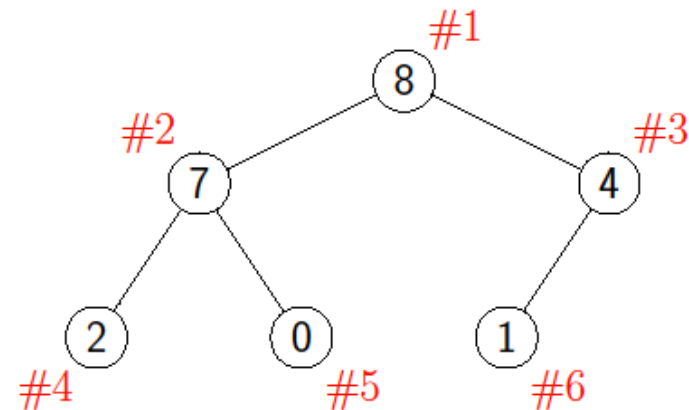
- ▶ Beim Entfernen der Wurzel wird beim:
 - ▶ Max-Heap: größte Element
 - ▶ Min-Heap: kleinste Elementaus dem Heap genommen und durch **Versickern** die Heap-Eigenschaft wiederhergestellt.
- ▶ Versickerung immer in den Teilbaum mit der kleineren Wurzel
- ▶ Wiederholung auf Rest-Heap bis sortierte Reihenfolge vorliegt



Struktur und Eigenschaften

- Die **Heap-Struktur** kann **lückenlos** in der Implementierung leicht durch ein **Array** repräsentiert werden, weil ein binärer Heap ein vollständiger binärer Baum ist.

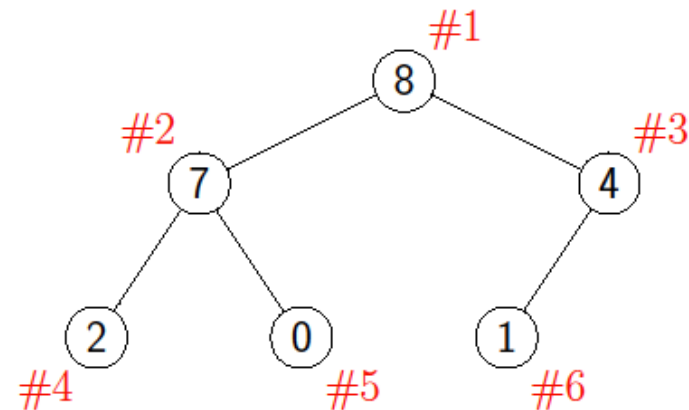
Index	1	2	3	4	5	6
Key	8	7	4	2	0	1



Struktur und Eigenschaften

- Die **Heap-Struktur** kann **lückenlos** in der Implementierung leicht durch ein **Array** repräsentiert werden, weil ein binärer Heap ein vollständiger binärer Baum ist.

Index	1	2	3	4	5	6
Key	8	7	4	2	0	1



- Nummerierung der Knoten des Heaps:
 - Kinder des k -ten Knotens auf Position
 - Links: $2k$
 - Rechts: $2k + 1$
 - Eltern-Knoten auf $\left\lfloor \frac{k}{2} \right\rfloor$

Eigenschaft auf Arrays

- ▶ Das Array $arr[1, \dots, n]$ erfüllt die **Heap-Eigenschaft**, wenn:
 - ▶ $arr[k] \geq arr[2k]$
 - ▶ $arr[k] \geq arr[2k + 1]$



Implementierung

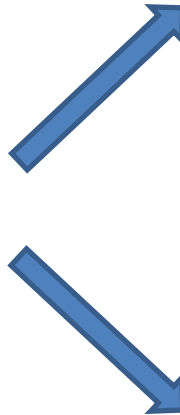
- ▶ Implementierung in Java
- ▶ Zwei Hilfsmethoden
 - ▶ Swap
 - ▶ Vertauscht zwei Elemente im Array



Implementierung

Heapsort

Eingabe: arr[0, ..., n-1]	
generateHeap(arr)	
sortHeap(arr)	
Ausgabe: sortiertes arr[0, ..., n-1]	



generateHeap()

Eingabe: arr[0, ..., n-1]	
wiederhole für i=arr.laenge / 2 bis 0	
percolate(arr, i, arr.laenge)	

sortHeap()

Eingabe: arr[0, ..., n-1]	
wiederhole für i=arr.laenge - 1 bis 1	
swap(arr, 0, i)	
percolate(arr, 0, i)	

Implementierung

percolate()

Eingabe: arr[0, ..., n-1]

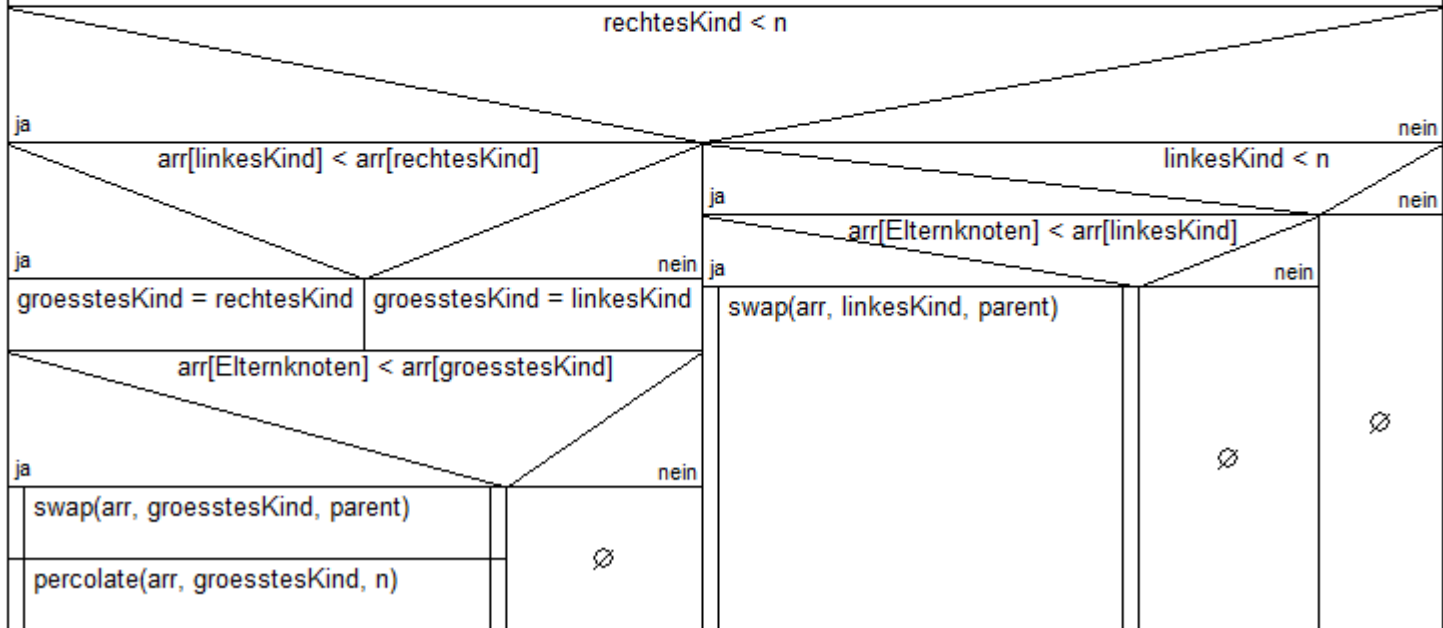
Eingabe: int Elternknoten

Eingabe: int groesseRestTeilbaum

linkesKind = 2 * parent

rechtesKind = 2 * parent + 1

int groesstesKind



Analyse der Laufzeit

- ▶ 2 Bestandteile

- ▶ generateHeap

- ▶ Percolate:

- $O(h) \rightarrow O(\log n)$

- ▶ Insgesamt: $\frac{n}{2} * \log n \rightarrow O(n * \log n)$

- ▶ Heapsortierung

- ▶ Swap: $O(1)$

- ▶ Percolate

- ▶ Insgesamt: $(n - 1) * 1 * \log n \rightarrow O(n * \log n)$

- ▶ Gesamtlaufzeit: $O(n \log n)$



Mögliche GFS-Themen

- ▶ Bottom-Up-Heapsort
- ▶ Smoothsort
- ▶ Ternäre Heaps & n-äre Heaps

