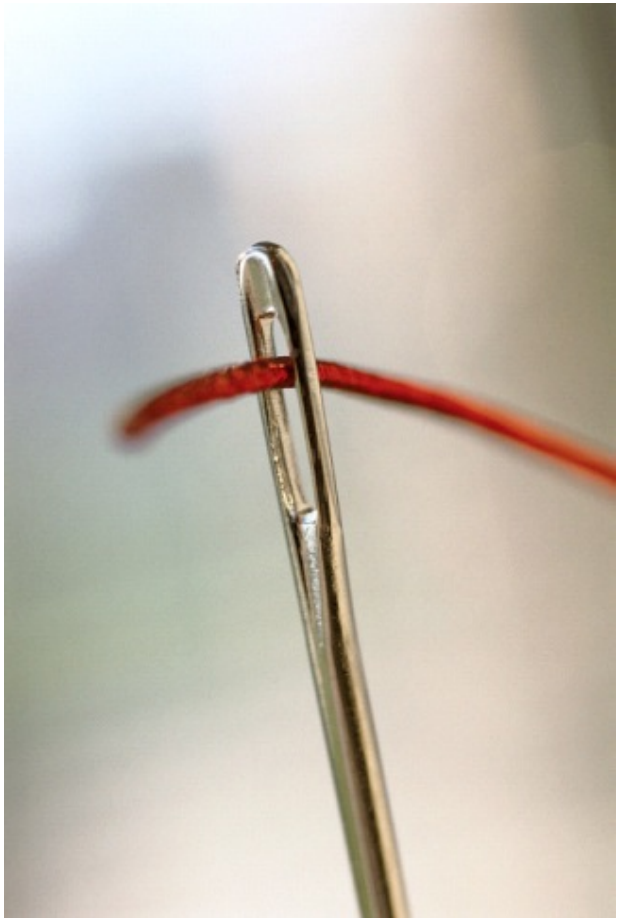
10. Kapitel – Teil 3

SORTIEREN MIT BÄUMEN - HEAPSORT

Algorithmen & Datenstrukturen

Prof. Dr. Wolfgang Schramm

Übersicht



0. Einführung
1. Algorithmen
2. Eigenschaften von Programmiersprachen
3. Algorithmenparadigmen
4. Suchen & Sortieren
5. Hashing
6. Komplexität von Algorithmen
7. Abstrakte Datentypen (ADT)
8. Listen
9. **Bäume**
10. Graphen

Lernziele des Kapitels

- Verstehen, wie ein Heap aufgebaut ist ?
- Kennenlernen wie ein Heap aufgebaut wird.
- Kennenlernen wie ein Heap sortiert wird.
- 2 Heap-Implementierungen kennenlernen.

Inhalt

- Übersicht
- Die Datenstruktur Heap
- Das Sortieren
- Beispiel
- Komplexität
- Vor- und Nachteile
- Hinweise zur Implementierung

HeapSort

Sortierverfahren 1964 entwickelt von J.W.J Williams (von Floyd „verbessert“).

Sortieren in 2 Schritten:

1. In bestimmte Datenstruktur bringen → HEAP

2. vorstrukturierte Daten sortieren → SORT

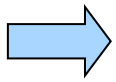
Die Datenstruktur HEAP

Einsatzbereich: Häufig bei Vorrangswarteschlangen oder sog. Greedy Algorithmen (z.B. Dijkstra).

Arten von Heaps:

- Fibonacci-Heap
- Binomial-Heap
- Treap
- Binärer Heap

→ unterschiedliche Laufzeitverhalten für die unterschiedlichen Operationen.



Hier nur binärer Heap.

Heap-Eigenschaften

Es gibt für den **binären Heap** zwei Implementierungsvarianten :

MaxHeap und **MinHeap**

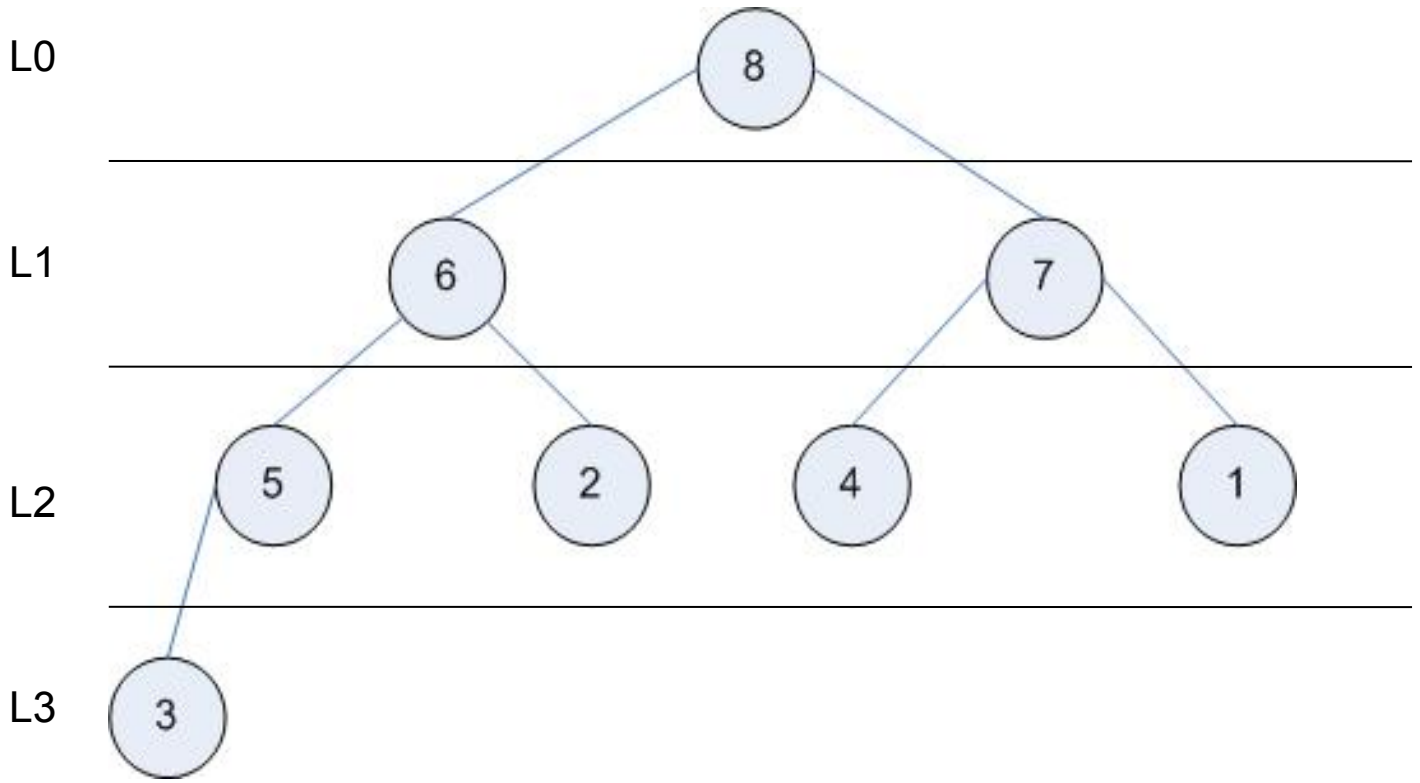
Eigenschaften:

- Ein binärer **MaxHeap** besteht aus einem Binärbaum bei dem bis auf das letzte Level alle vollständig ausgefüllt sind. Das letzte Level muss von links aufgefüllt sein.
- Für alle inneren Knoten muss gelten:

  $\forall\ x_1 \in L,\ x_2 \in L+1$ (x_1 ist Elternknoten von x_2): $x_1 \geq x_2$

Oder: Ein Binärbaum stellt einen MaxHeap dar, wenn alle Elternknoten größer oder gleich ihren Kinderknoten sind.

Heap: Beispiel



Heap aufbauen

2 Verfahren

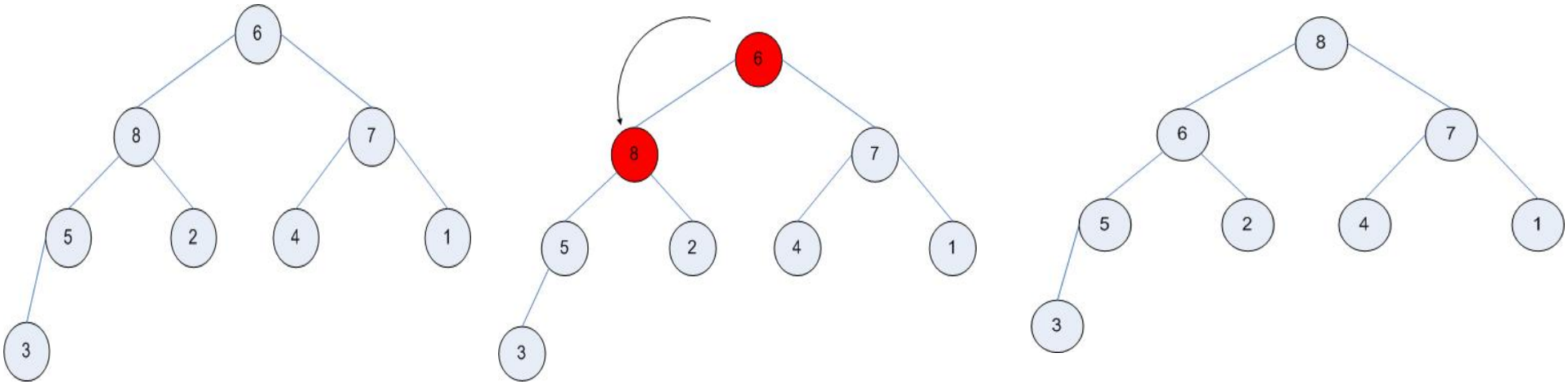
- von J.W.J. Williams mit einer Komplexität von O(n*log(n)),
- von R.W. Floyd mit einer Komplexität O(n).

Hierfür benötigt: Methode zum **Versickern** eines Knotens.

- Wird sowohl für das Erstellen des Heaps benötigt als auch für die Sortierung.
- Wird das Wurzelelement entfernt und durch ein neues Element ersetzt, ist nicht mehr gewährleistet, ob die Heap-Bedingungen noch erfüllt sind.
- Man nutzt hierzu die Eigenschaft aus das die beiden Teilbäume noch die Heapeigenschaft besitzen und lässt das neue Element versickern.

Heap: Versickern

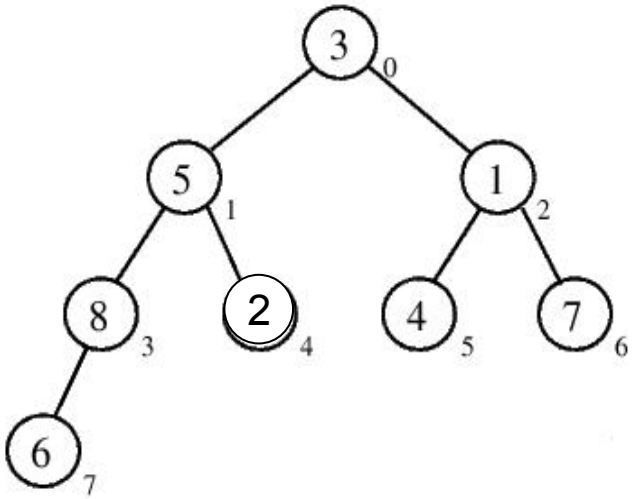
1. Prüfen ob Heapeigenschaft verletzt.
2. Prüfen ob das Element einen oder zwei Söhne besitzt .
3. Wählen des größeren Sohnes und vertauschen, also zum neuen Vergleichselement machen.



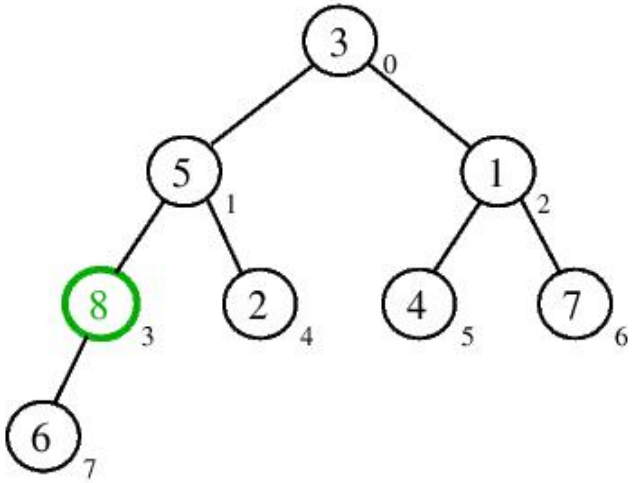
Heap: Erstellen

- Der Baum hat n Elemente.
- Baue zuerst Heaps der Höhe 2 auf; beginne dazu bei Element n/2 – 1.
- Danach werden Heaps der nächst größeren Höhe aufgebaut.
- Führe für dieses Element die Funktion Versickern aus, falls Heapbedingung für dieses nicht erfüllt.
- Wähle danach die Elemente n/2 -2, n/2-3, . . . , 0.
- **Erklärung:**
- Auf dem letzten Level hat ein binärer Baum max. (n+1)/2 Elemente.
- Erstellt man nun Heaps der Höhe 2, so nimmt man die direkt über den Blättern liegenden Knoten. Also Level n-1 Das letzte Element hier hat die Position n/2 – 1.
- Diese Knoten lässt man nun nach und nach versickern bis man bei der Wurzel angekommen ist.

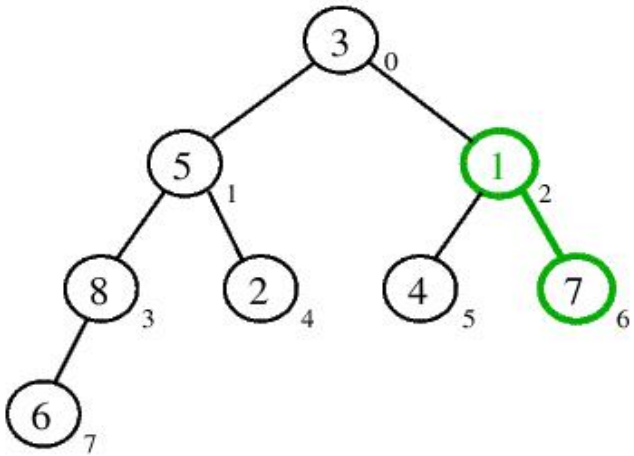
Heap: Beispiel



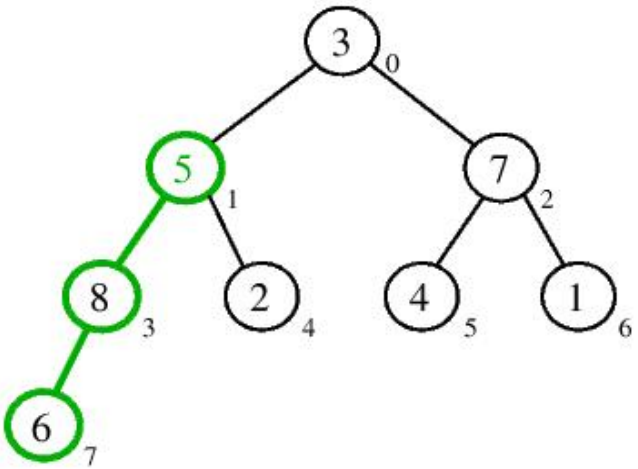
Heap: Beispiel



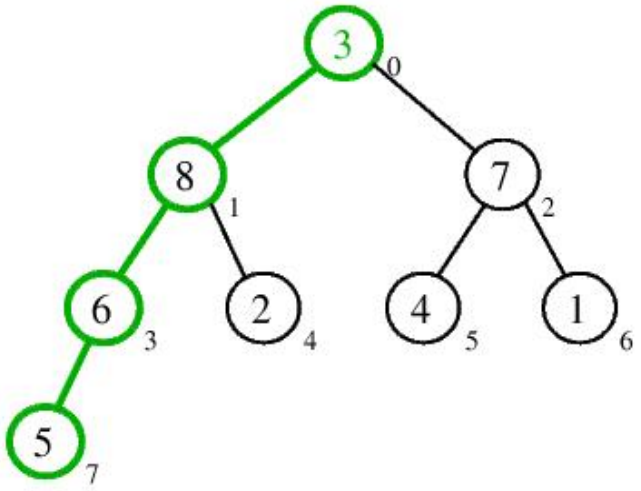
Heap: Beispiel



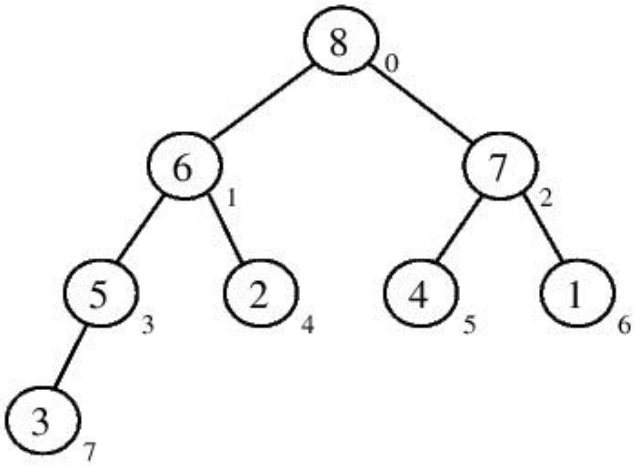
Heap: Beispiel



Heap: Beispiel



Heap: Beispiel



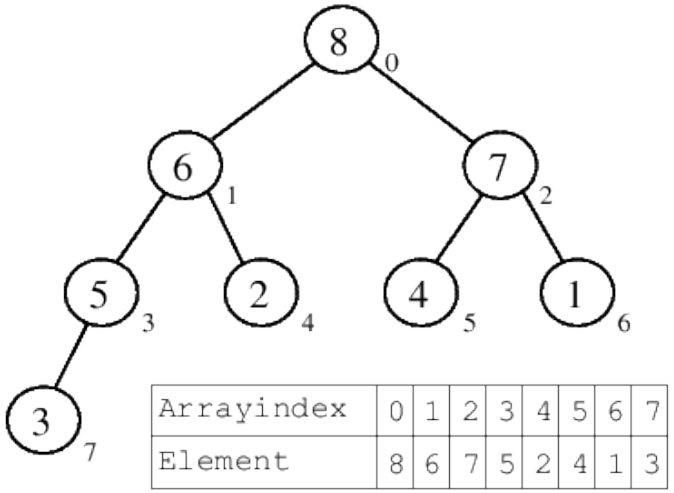
Heap: Sortieren

Aus dem nun vorliegenden (max)Heap und der Methode des Versickern lässt sich leicht ein Sortierverfahren konstruieren.

Vorgehen:

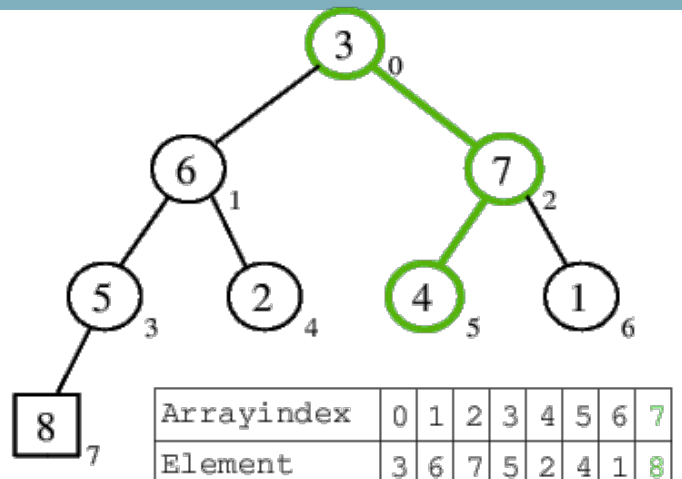
1) Vertausche das Wurzelelement mit dem letzten Element der Blattebene.
2) Lass dieses Element nun versickern (bis n-1) falls Heapbedingungen verletzt.
3) Betrachte nun wie beim Selectionsort nur noch die Menge n-1.
4) Beginne wieder bei 1 bis die zu betrachtende Menge 0 ist.

Heap: Sortieren – Beispiel (MinHeap)



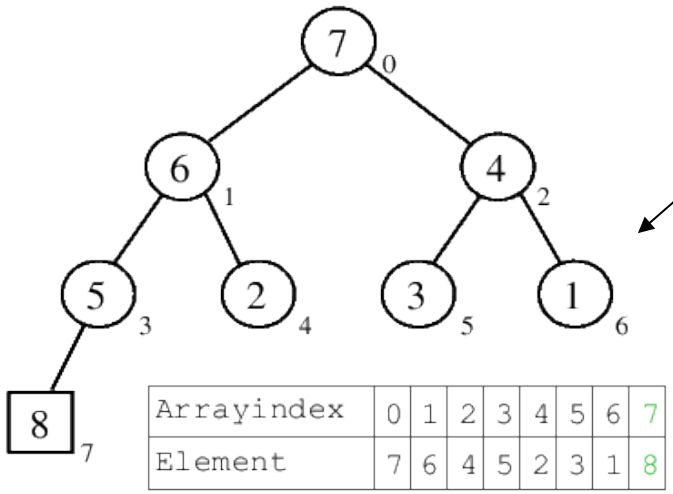
Arrayindex	0	1	2	3	4	5	6	7
Element	8	6	7	5	2	4	1	3

1. Ausgangsheap



Arrayindex	0	1	2	3	4	5	6	7
Element	3	6	7	5	2	4	1	8

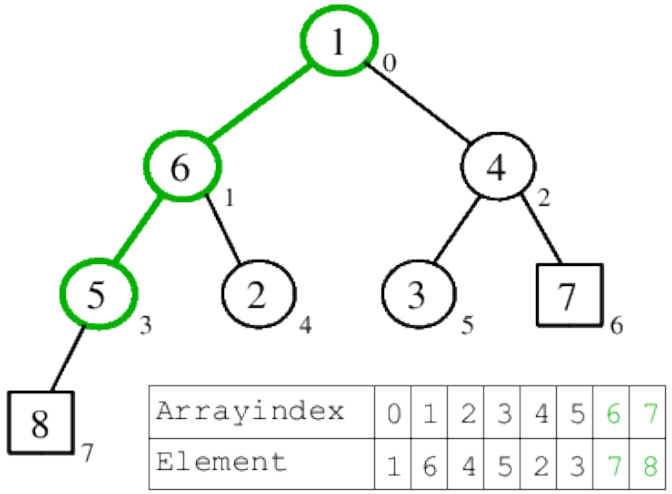
2.1 Vertauschen / versickern



Arrayindex	0	1	2	3	4	5	6	7
Element	7	6	4	5	2	3	1	8

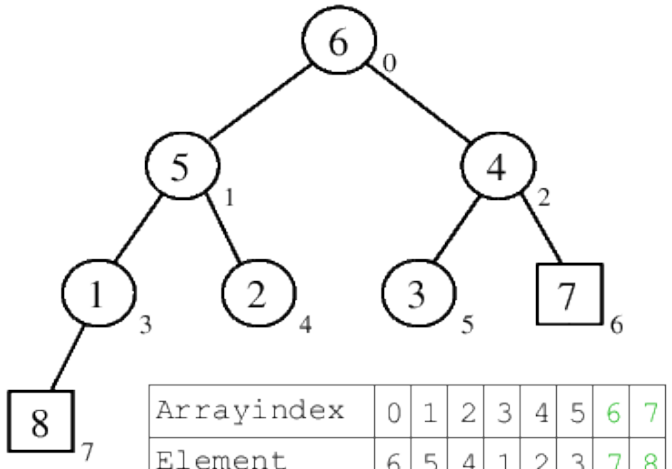
2.2 Wurzel: 7

Heap: Sortieren - Beispiel



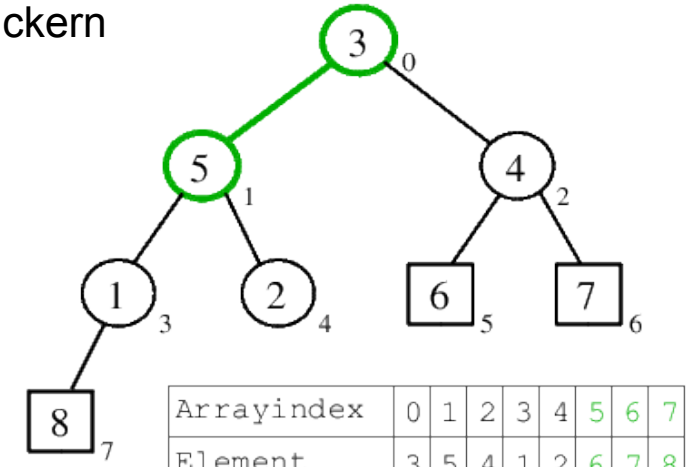
Arrayindex	0	1	2	3	4	5	6	7
Element	1	6	4	5	2	3	7	8

3.1 Vertauschen / versickern



Arrayindex	0	1	2	3	4	5	6	7
Element	6	5	4	1	2	3	7	8

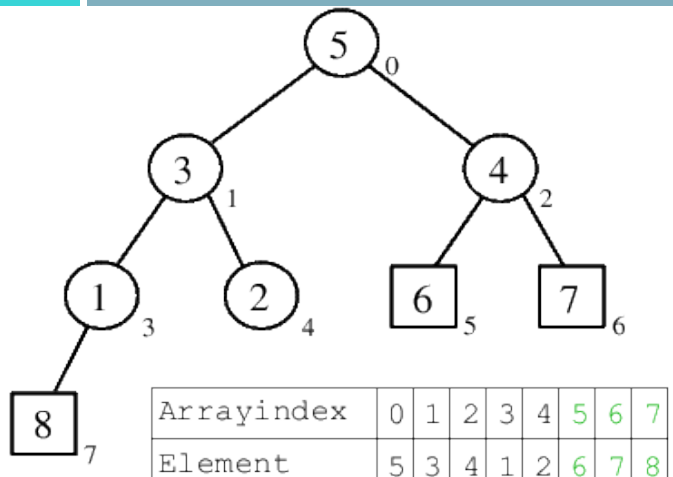
3.2 neue Wurzel: 6



Arrayindex	0	1	2	3	4	5	6	7
Element	3	5	4	1	2	6	7	8

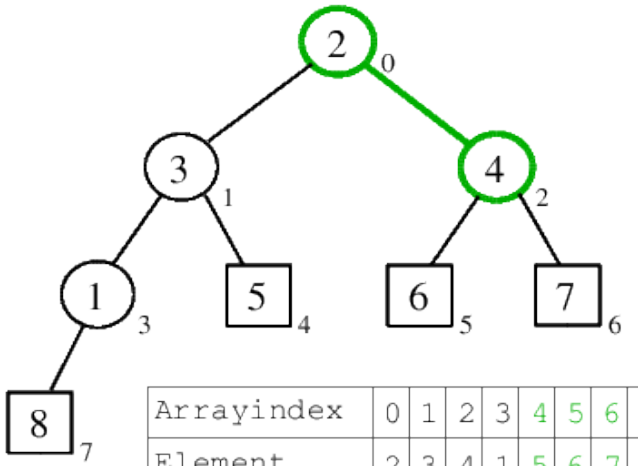
4.1 Vertauschen und versickern

Heap: Sortieren - Beispiel



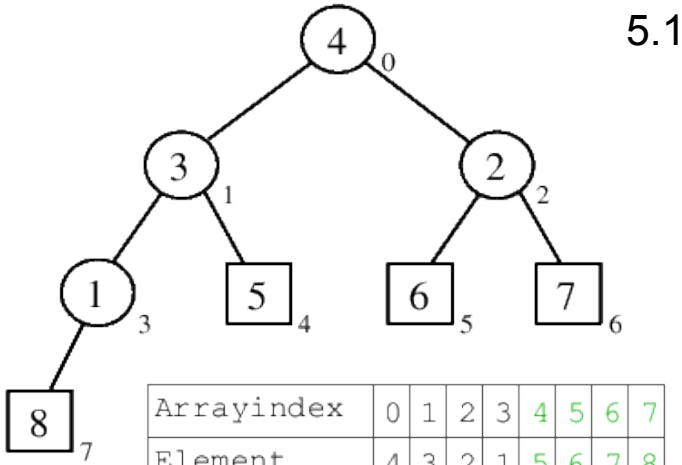
Arrayindex	0	1	2	3	4	5	6	7
Element	5	3	4	1	2	6	7	8

4.2 Wurzel: 5



Arrayindex	0	1	2	3	4	5	6	7
Element	2	3	4	1	5	6	7	8

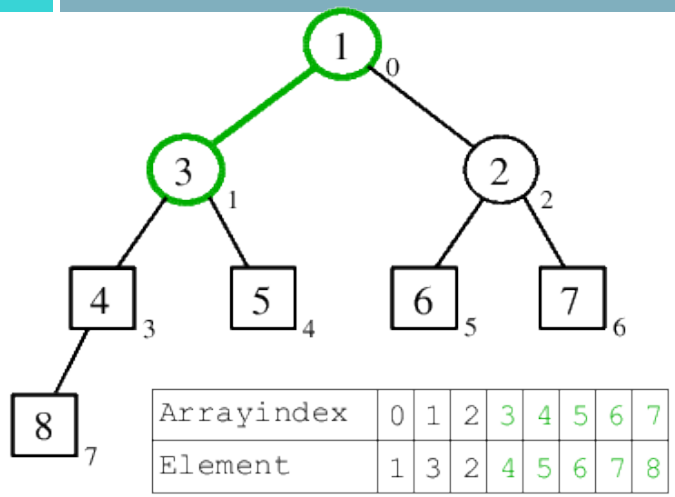
5.1 vertauschen und versickern



Arrayindex	0	1	2	3	4	5	6	7
Element	4	3	2	1	5	6	7	8

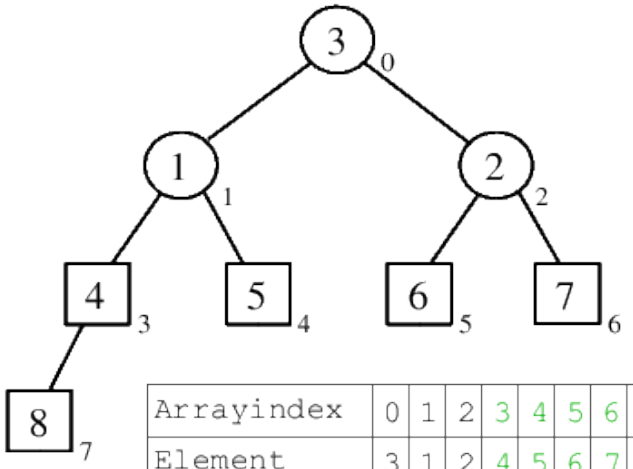
5.2 neue Wurzel: 4

Heap: Sortieren - Beispiel



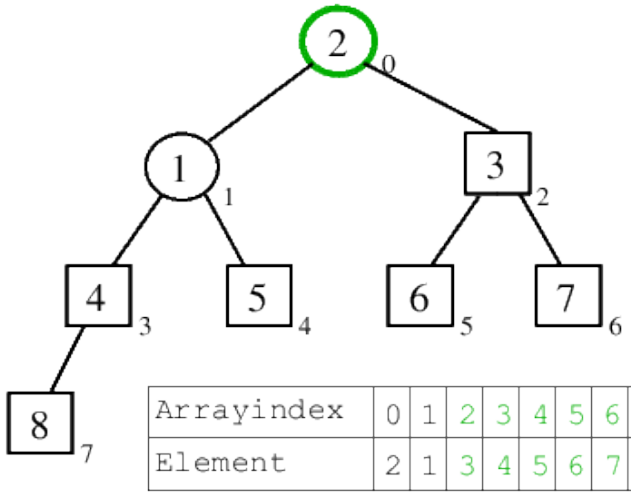
Arrayindex	0	1	2	3	4	5	6	7
Element	1	3	2	4	5	6	7	8

6.1 vertauschen und versickern



Arrayindex	0	1	2	3	4	5	6	7
Element	3	1	2	4	5	6	7	8

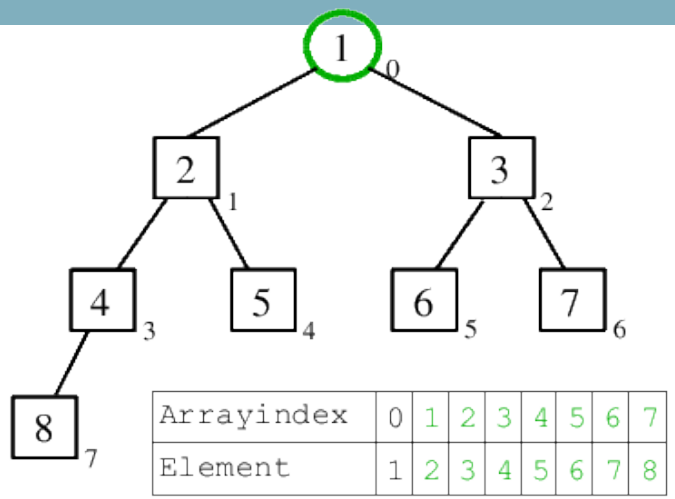
6.2 neue Wurzel: 3



Arrayindex	0	1	2	3	4	5	6	7
Element	2	1	3	4	5	6	7	8

7. vertauschen (kein versickern)

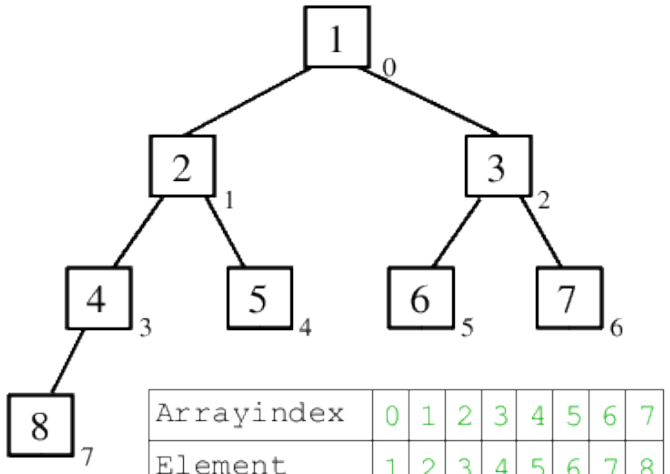
Heap: Sortieren - Beispiel



Arrayindex	0	1	2	3	4	5	6	7
Element	1	2	3	4	5	6	7	8

8. Heap besteht nur noch aus einem Element

→ kein versickern



Arrayindex	0	1	2	3	4	5	6	7
Element	1	2	3	4	5	6	7	8

Fertig

Komplexität

Aufbau des Heaps benötigt O(n).

Versickern benötigt O(log(n)) und dies n-mal.

GESAMT

→ O(n)+n*O(log(n)) = *O(n*log(n))*

Vor- / Nachteile

+ worst case immer noch O(n*log(n))

+ internes Sortierverfahren (in-place)

+ Implementierung mit Array → wenig Code

- Im Vergleich zum Quicksort nur im worst case besser, der selten eintritt

- nicht stabil

Array-Darstellung

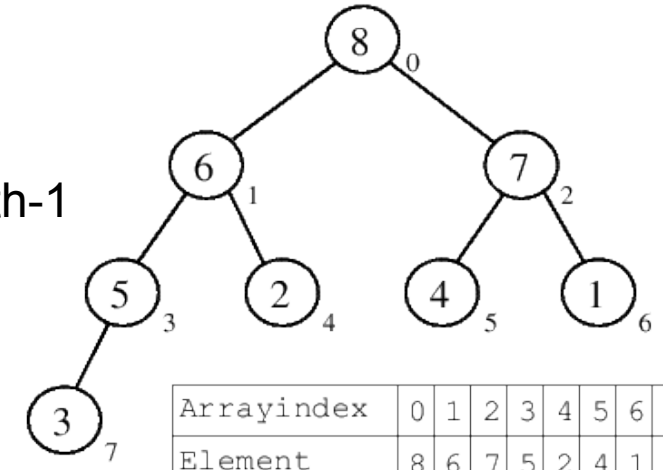
Versickern:

Index eines Knotens: i

Index der Söhne: 2i+1 <= n-1 und 2i+2 <= n-1

Heapsort

in jedem Durchlauf: aktuelle array.length-1



Arrayindex	0	1	2	3	4	5	6	7
Element	8	6	7	5	2	4	1	3

Literatur

Quellen:

[Wilj64] J. W. J. Williams. Algorithm 232 (heapsort). Communications of the ACM, 7:347-348, 1964

Thomas Ottmann und Peter Widmayer: "Algorithmen und Datenstrukturen". 3. Auflage. 1996. Spektrum Akademischer Verlag, Heidelberg

Für die Beispiele:

http://www.linux-related.de/index.html?/coding/sort/sort_heap.htm