ETH zürich

Übungslektion 8 – Bäume & Heaps

Informatik II

8. / 9. April 2025

Heutiges Programm

Wiederholung von Kursinhalten

Lektionsübung

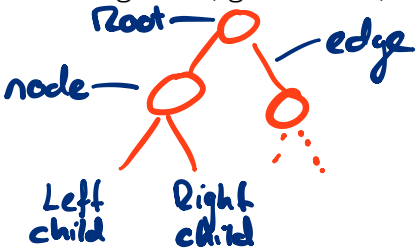
Hausaufgaben

1. Wiederholung von Kursinhalten

Bäume

Ein *Baum* ist

- Verallgemeinerte Liste: Knoten können mehrere Nachfolger haben.
- Spezieller Graph: Graphen bestehen aus Knoten und Kanten. Ein Baum ist ein zusammenhängender, gerichteter, azyklischer Graph.



Binäre Bäume

Ein *binärer Baum* ist

- entweder ein Blatt, d.h. ein leerer Baum,
- oder ein innerer Knoten mit zwei Bäumen T_l (linker Teilbaum) und T_r (rechter Teilbaum) als linken und rechten Nachfolger.

immer nur zwei Kinder

In jedem inneren Knoten **`v`** wird gespeichert

- ein Schlüssel **`v.key`** und
- zwei Zeiger **`v.left`** und **`v.right`** auf die Wurzeln der linken und rechten Teilbäume.

Ein Blatt wird durch den null-Zeiger repräsentiert. Um überfüllte Diagramme zu vermeiden, lassen wir beim Zeichnen von Bäumen manchmal die Kanten zu den Blättern weg.

Arten von Binären Bäumen

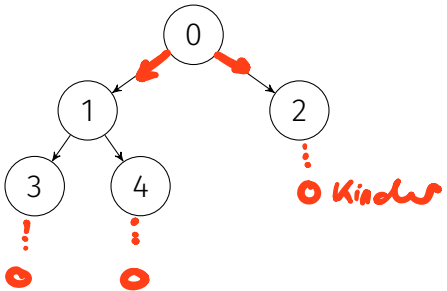
Es gibt drei Arten von Binärbäumen basierend auf der **Anzahl der Kinder**:

- Voller Binärbaum
- Vollständiger Binärbaum
- Perfekter Binärbaum

Hinweis: In der Literatur existieren verschiedene Bezeichnungen und Definitionen!

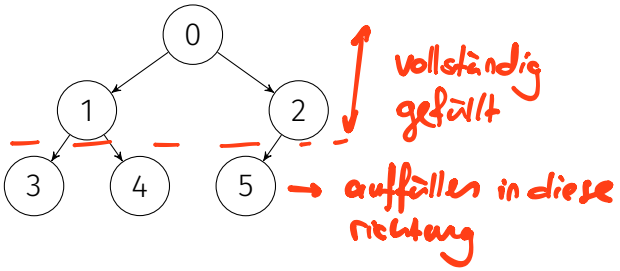
Voller Binärbaum

Ein Binärbaum ist ein voller Binärbaum, wenn jeder Knoten 0 oder 2 Kinder hat.



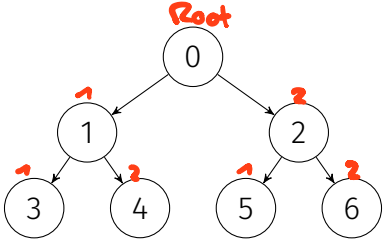
Vollständiger Binärbaum

Ein vollständiger Binärbaum ist ein Baum, in dem alle Ebenen vollständig gefüllt sind, ausser möglicherweise die letzte Ebene, die von links nach rechts gefüllt ist.



Perfekter Binärbaum

Ein Binärbaum ist ein Baum, bei dem alle inneren Knoten zwei Kinder haben und alle Blätter auf derselben Ebene sind.



Binäre Bäume: Quiz

zwei oder 0 Kinder

Kannst

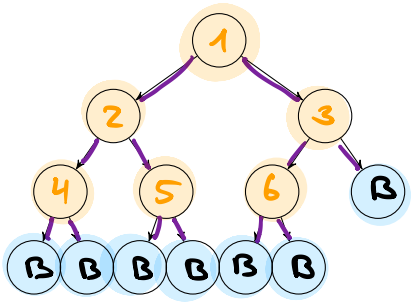
Wie viele Knoten enthält ein voller Binärbaum mit 6 Nicht-Blatt-Knoten höchstens:

A. 6 Knoten
B. 9 Knoten
C. 11 Knoten
D. 13 Knoten

Binäre Bäume: Quiz Antwort

Ein voller Binärbaum mit n Nicht-Blatt-Knoten enthält $2n + 1$ Knoten. Also: $2 \times 6 + 1 = 13$.

Knoten: Wurzel, innere Knoten, Blätter } 6



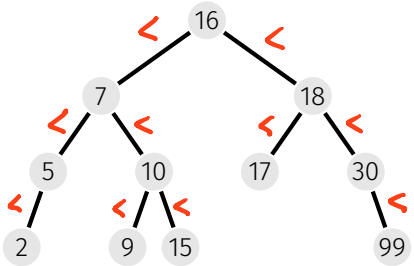
n - nicht Blatt: 2n+1 Knoten

6 + 7 = 13

Binäre Suchbäume

Ein *binärer Suchbaum* ist ein binärer Baum, der die **Suchbaumeigenschaft** erfüllt:

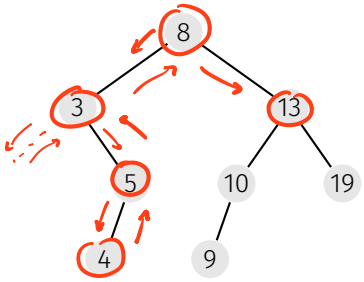
- Jeder Knoten `v` speichert einen Schlüssel
- Schlüssel im linken Teilbaum `v.left` kleiner als `v.key`
- Schlüssel im rechten Teilbaum `v.right` grösser als `v.key`



Traversierungsarten

- *Hauptreihenfolge* (*preorder*):
  v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.

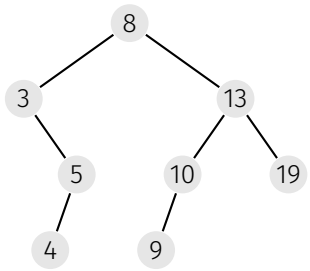
1 2 3



8 3 5 4 13 10 9 19

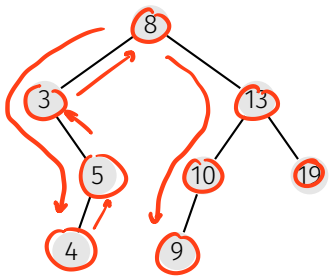
Traversierungsarten

- *Hauptreihenfolge* (*preorder*):
  v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19



Traversierungsarten

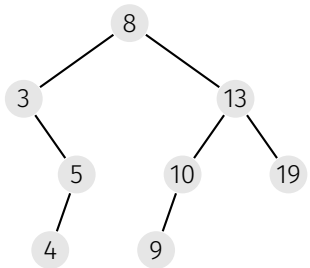
- *Hauptreihenfolge* (*preorder*):
  v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- *Nebenreihenfolge* (*postorder*):
  $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.



4 5 3 8 9 10 13 19

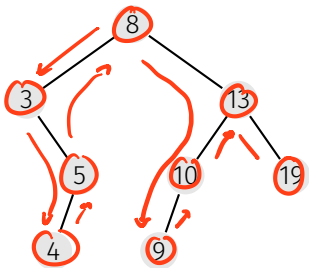
Traversierungsarten

- *Hauptreihenfolge* (*preorder*):
  v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- *Nebenreihenfolge* (*postorder*):
  $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
  4, 5, 3, 9, 10, 19, 13, 8



Traversierungsarten

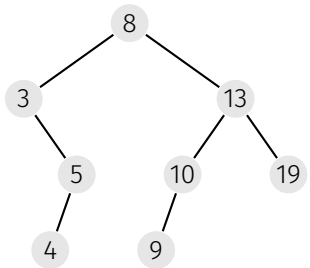
- *Hauptreihenfolge* (*preorder*):
  v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- *Nebenreihenfolge* (*postorder*):
  $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
  4, 5, 3, 9, 10, 19, 13, 8
- *Symmetrische Reihenfolge* (*inorder*):
  $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.



3 4 5 8 9 10 13 19

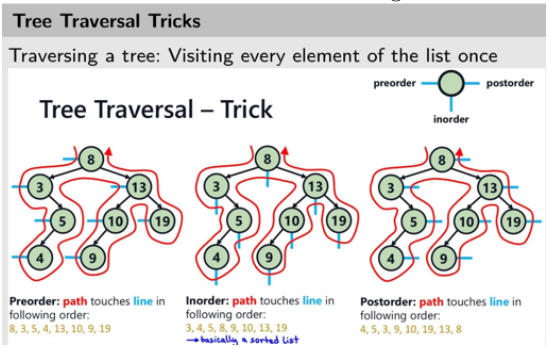
Traversierungsarten

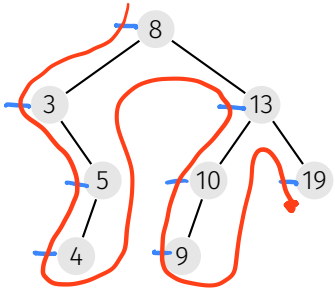
- *Hauptreihenfolge* (*preorder*):
  v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- *Nebenreihenfolge* (*postorder*):
  $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
  4, 5, 3, 9, 10, 19, 13, 8
- *Symmetrische Reihenfolge* (*inorder*):
  $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.
  3, 4, 5, 8, 9, 10, 13, 19



Traversierungsarten – Trick

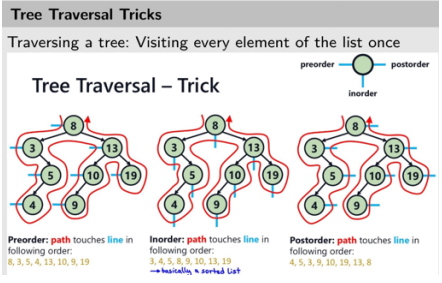
- *Hauptreihenfolge* (*preorder*):
  v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.

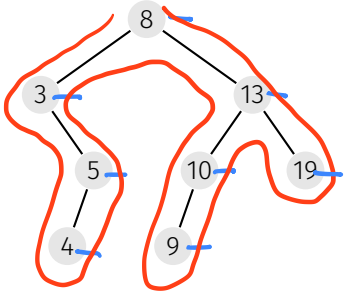




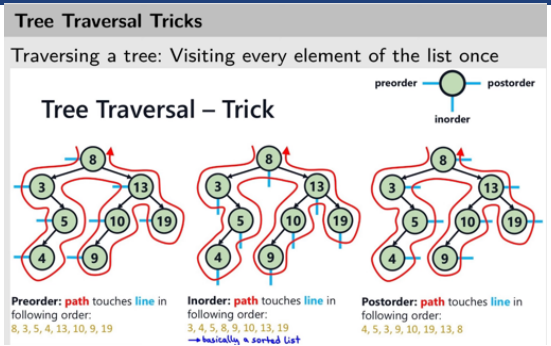
Traversierungsarten

- *Hauptreihenfolge* (*preorder*):
  v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- *Nebenreihenfolge* (*postorder*):
  $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.

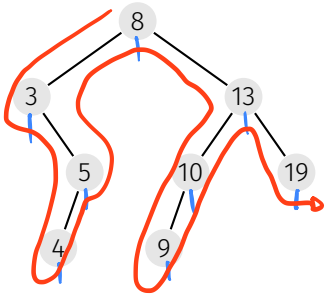




Traversierungsarten



- *Symmetrische Reihenfolge* (*inorder*): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.



Traversierungsarten: Quiz

Left: kleiner
Right: Grösser

Zeichnen Sie jeweils einen binären Suchbaum, der die folgenden Traversierungen erzeugt. Ist der Baum eindeutig?

Symmetrische Reihenfolge (inorder)	1 2 3 4 5 6 7 8
Hauptreihenfolge (preorder)	4 3 1 2 8 6 5 7
Nebenreihenfolge (postorder)	1 3 2 5 6 8 7 4

Geben Sie zu jeder Reihenfolge eine Zahlensequenz aus $\{1, \ldots, 4\}$, die nicht aus einem gültigen binären Suchbaum stammen kann.

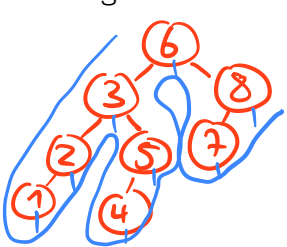
Traversierungsarten: Quiz Antwort

1 2 3 4 5 6 7 8

⇒ inorder gibt immer sorted list!

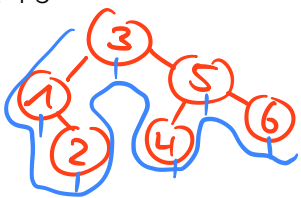
Wo ist root?

- Jeder Suchbaum mit den Zahlen $\{1, \ldots, 8\}$ ist gültig.
- Der Baum ist nicht eindeutig.
- Es gibt keinen Suchbaum, welcher nicht die aufsteigend sortierte Sequenz ausgeben würde. Gegenbeispiel: 1 2 4 3



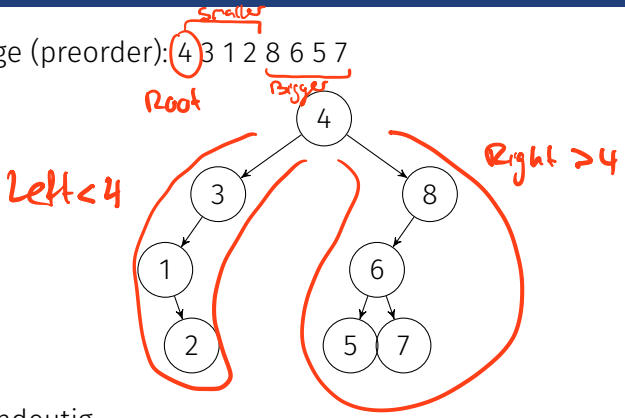
≙

same output



Traversierungsarten: Quiz Antwort

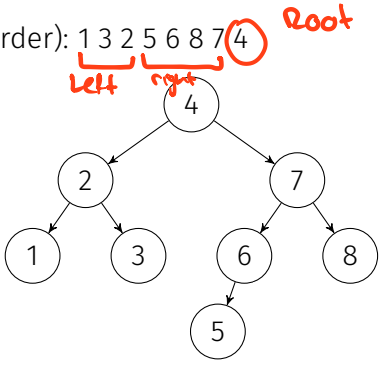
- Hauptreihenfolge (preorder): 4 3 1 2 8 6 5 7



- Der Baum ist eindeutig.
- Es muss rekursiv gelten, dass zuerst eine Gruppe Zahlen kleiner und danach grösser als der erste Wert kommen. Gegenbeispiel: 3 1 4 2

Traversierungsarten: Quiz Antwort

- Nebenreihenfolge (postorder): 1 3 2 5 6 8 7 4



- Der Baum ist eindeutig.
- Konstruktion hier: `https://www.techiedelight.com/build-binary-search-tree-from-postorder-sequence/`, Ähnliches Argument wie vorher, nur von hinten nach vorne. Gegenbeispiel 4 2 1 3

Heaps

Ein *Heap* ist eine baumbasierte Datenstruktur wobei:

- Der Baum ein **vollständiger Binärbaum** ist. → alle Ebenen gefüllt ausser letzte
- Ein Heap mit **N** Knoten eine Höhe von **log N** hat (Eigenschaft eines vollständigen Binärbaums).
- Es ist für die schnelle Extraktion von Minimum oder Maximum und für das Sortieren optimiert.

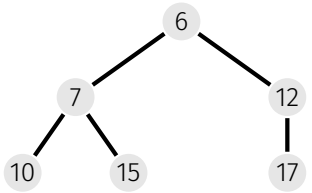
Es gibt zwei Arten von Heap-Implementierungen:

- Min-Heaps → Root = Min
- Max-Heaps → Root = Max

Min-Heaps

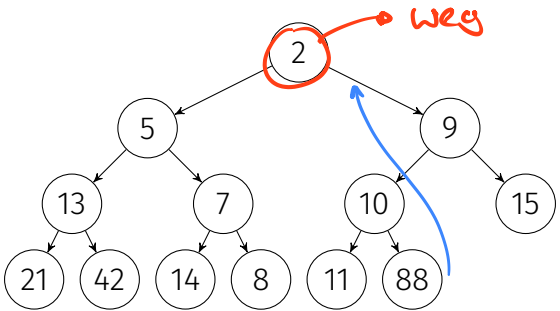
Ein *Min-Heap* ist ein binärer Baum wo:

- Der am Wurzelknoten vorhandene Schlüssel ist immer **kleiner oder gleich** den Schlüsseln aller seiner Kinder. 6 kleinstes
- Dieselbe Eigenschaft muss für alle Teilbäume in diesem Binärbaum rekursiv wahr sein. : 7 < 10, 15 | 12 < 17
- Das **kleinste** Schlüsselelement ist daher immer an der Wurzel vorhanden.



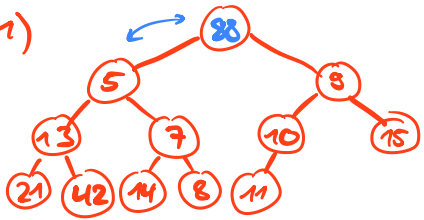
Min-Heaps: Quiz

Führen Sie auf folgendem Min-Heap eine Extract-Min (entferne den kleinsten Schlüssel) Operation aus, wie in der Vorlesung vorgestellt, einschliesslich der Wiederherstellung der Heap-Bedingung. Wie sieht der Heap nach der Operation aus?

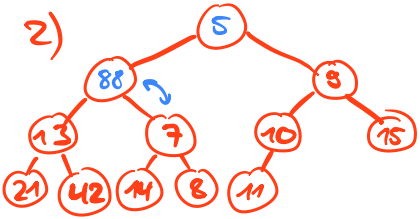


Min-Heaps: Quiz

1)

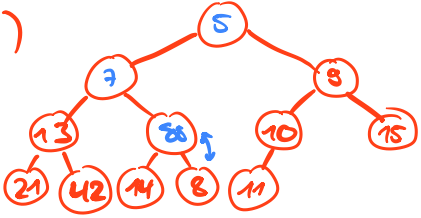


2)

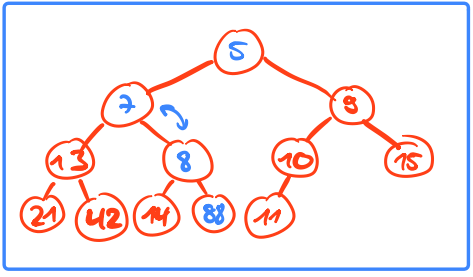


min heap: sift down
direction smallest element

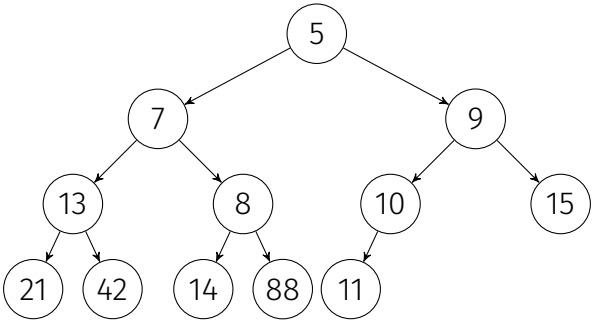
3)



4)



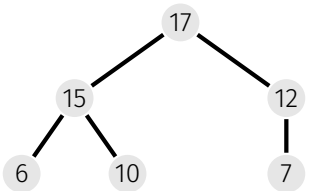
Min-Heaps: Quiz Antwort



Max-Heaps

Ein *Max-Heap* ist ein binärer Baum wo:

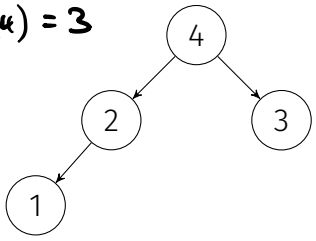
- Der am Wurzelknoten vorhandene Schlüssel ist immer **größer oder gleich** den Schlüsseln aller seiner Kinder.
- Dieselbe Eigenschaft muss für alle Teilbäume in diesem Binärbaum rekursiv wahr sein.
- Das **größte** Schlüsselelement ist daher immer an der Wurzel vorhanden.

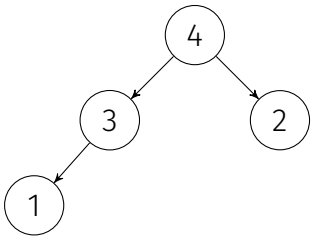


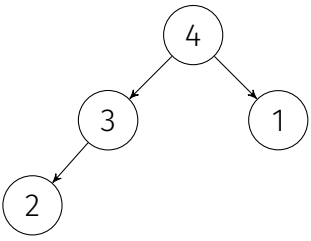
Anzahl Max-Heaps

Sei $N(n)$ die Anzahl verschiedener Max-Heaps, welche aus allen Schlüsseln $1, 2, \ldots, n$ gebildet werden können. Beispielsweise ist $N(1) = 1,\ N(2) = 1,\ N(3) = 2,\ N(4) = 3$ und $N(5) = 8$.
Finde die Werte $N(6)$ und $N(7)$.

$N(4) = 3$







$\binom{n}{k} = \frac{n!}{k!(n-k)!}$

$N(1) = 1$

$N(2) =$ 

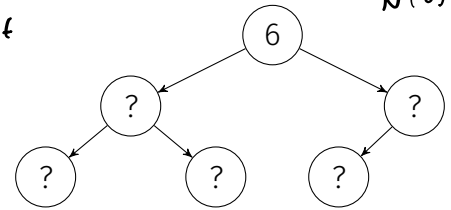
$N(3) =$ 

Anzahl Max-Heaps

$$N(n) = \binom{n-1}{l} \cdot N(l) \cdot N(r)$$

tipp: $\binom{5}{3} = 10$, $\binom{6}{3} = 20$

Ein die Elemente $1, 2, 3, 4, 5, 6$ enthaltender Max-Heap sieht so aus:



l = # nodes left

$r = n - 1 - l$ = # nodes right

$l = \lfloor 2^{h-1} - 1 + \min(p, 2^{h-1}) \rfloor$

h = height $= \lfloor \log_2(n) \rfloor$ ← abrunden

$p = n - (2^h - 1)$

$N(6)$: $h = \log_2(6) = 2$

$p = 6 - (2^2 - 1) = 3$

$l = 2^{2-1} - 1 + \min(3, 2^{2-1})$

$l = 1 + 2 = 3$

$r = 6 - 1 - 3 = 2$

$N(6) = \binom{6-1}{3} \cdot N(3) \cdot N(2)$

$N(6) = \binom{5}{3} \cdot N(3) \cdot N(2) =$

$= 10 \cdot 2 \cdot 1 = 20$

$N(7)$: $h = \log_2(7) = 2$

$p = 7 - (2^2 - 1) = 4$

$l = 2^{2-1} - 1 + \min(4, 2^{2-1})$

$l = 1 + 2 = 3$

$r = 7 - 1 - 3 = 3$

$N(7) = \binom{6}{3} \cdot N(3) \cdot N(3) = 20 \cdot 2 \cdot 2 = 80$

Anzahl Max-Heaps

Möglichkeiten, Elemente des linken Teilbaums zu wählen: $\binom{5}{3}$.

$$\Rightarrow N(6) = \binom{5}{3} \cdot N(3) \cdot N(2) = 10 \cdot 2 \cdot 1 = 20.$$

$$\text{und } N(7) = \binom{6}{3} \cdot N(3) \cdot N(3) = 20 \cdot 2 \cdot 2 = 80.$$

Komplexitätsanalyse von Min-Heap und Max-Heap

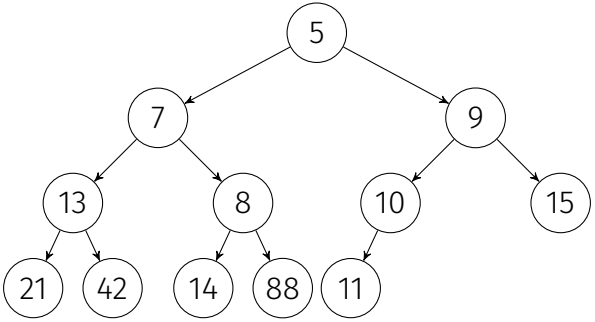
- Erhalten des größten oder kleinsten Elements: **O(1)** → ganz oben
- Element in Max-Heap oder Min-Heap einfügen: **O(log N)**
- Größte- oder Kleinste-Element entfernen: **O(log N)** — sift up/down

Heaps: hinzufügen von Elementen immer bei Blättern von links → rechts
↳ danach Heap bedingung wiederherstellen! (sift up)

~~Min-Heaps: Quiz Antwort~~

Heaps als Listen dargestellt: wie sieht dieser Heap aus?

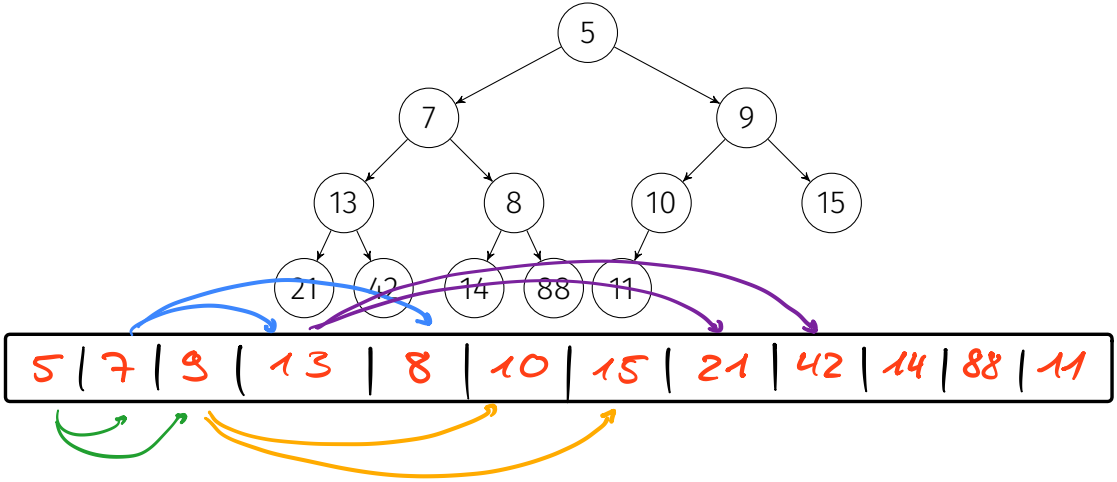
Kahoot



~~Min-Heaps: Quiz Antwort~~

Heaps als Listen dargestellt: wie sieht dieser Heap aus?

Kahoot



Schlüssel Einfügen

Binärer Suchbaum

- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.

Min-Heap

- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).

Schlüssel Einfügen

Binärer Suchbaum

- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.

Min-Heap

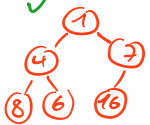
- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).

min heap:
sift up, swap mit kleinstem elem

Aufgabe: Einfügen von $4, 8, 16, 1, 6, 7$ in leeren Baum/Heap.

nach und nach

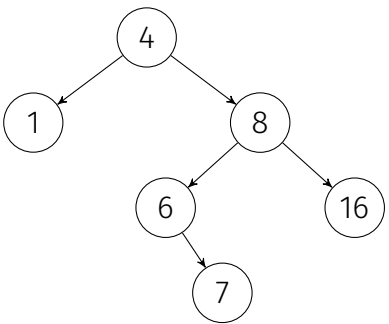
→ Blätter auffüllen, Bedingung checken:



Schlüssel Einfügen

Binärer Suchbaum

- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.



Min-Heap

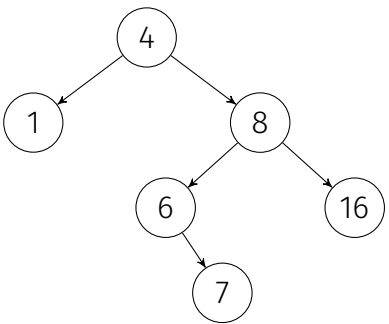
- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).

$\log$

Schlüssel Einfügen

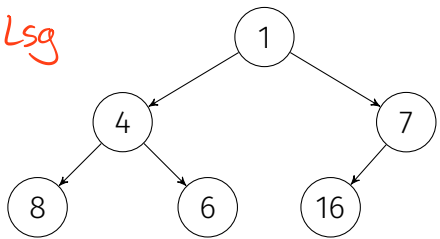
Binärer Suchbaum

- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.



Min-Heap

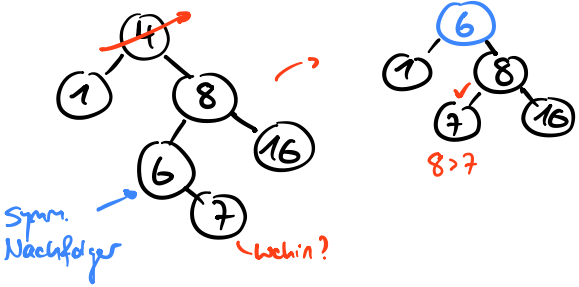
- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).



Schlüssel Löschen

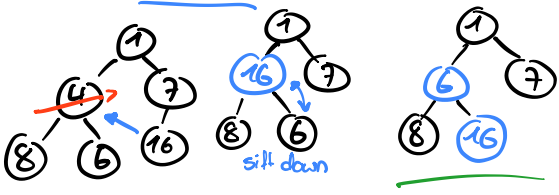
Binärer Suchbaum

- Schlüssel k durch symm. Nachfolger n ersetzen.
- Achtung: Wohin mit rechtem Kind von n?



Min-Heap

- Schlüssel durch hinterstes Arrayelement ersetzen.
- Heap-Bedingung wiederherstellen: `siftDown` *oder* `siftUp`.



Schlüssel Löschen

Binärer Suchbaum

- Schlüssel k durch symm. Nachfolger n ersetzen.
- Achtung: Wohin mit rechtem Kind von n?

Min-Heap

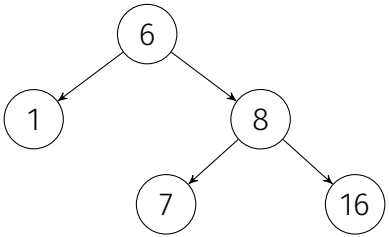
- Schlüssel durch hinterstes Arrayelement ersetzen.
- Heap-Bedingung wiederherstellen: `siftDown` *oder* `siftUp`.

Aufgabe: Löschen von 4 in Beispiel-Baum/Heap.

Schlüssel Löschen

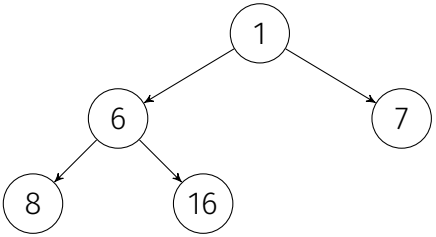
Binärer Suchbaum

- Schlüssel k durch symm. Nachfolger n ersetzen.
- Achtung: Wohin mit rechtem Kind von n?



Min-Heap

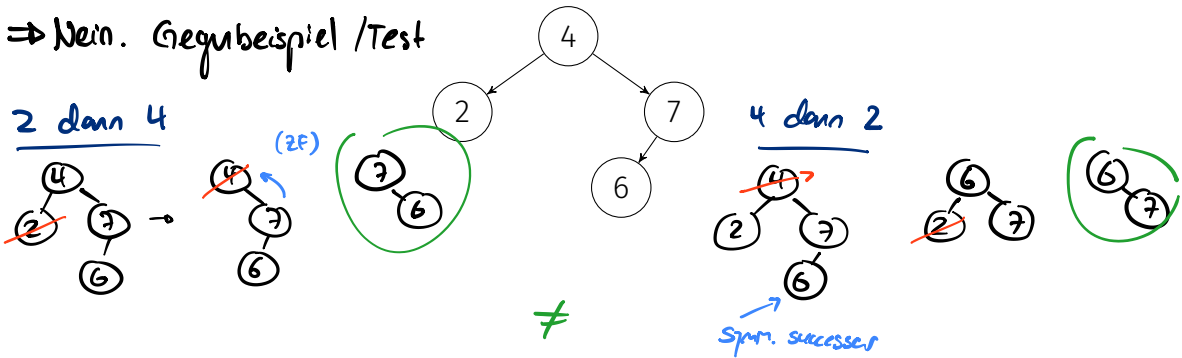
- Schlüssel durch hinterstes Arrayelement ersetzen.
- Heap-Bedingung wiederherstellen: `siftDown` *oder* `siftUp`.



Schlüssel Löschen: Quiz

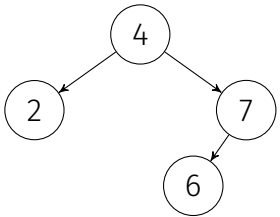
Wenn Sie zwei Schlüssel aus einem Binären Suchbum löschen wollen, spielt es dann eine Rolle, in welcher Reihenfolge Sie dies tun? Mit anderen Worten, ist das Löschen kommutativ?

⇒ Nein. Gegenbeispiel / Test



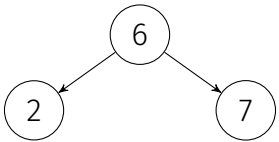
Schlüssel Löschen: Quiz Antwort

Ja, die Reihenfolge kann wichtig sein. Betrachten wir ein Gegenbeispiel. Angenommen, wir löschen Schlüssel 4 aus diesem Baum.



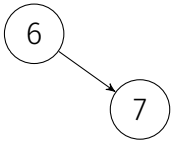
Schlüssel Löschen: Quiz Antwort

Der Schlüssel 6 ist jetzt die neue Wurzel. Löschen wir nun Schlüssel 2.



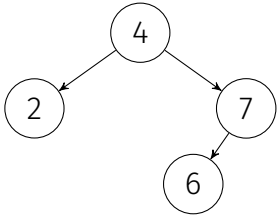
Schlüssel Löschen: Quiz Antwort

Schlüssel 2 wurde gelöscht. Dies ist der Binäre Suchbaum, den wir erhalten, nachdem wir Schlüssel 4 und dann Schlüssel 2 gelöscht haben.



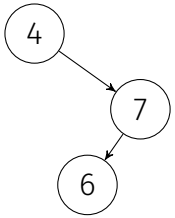
Schlüssel Löschen: Quiz Antwort

Kehren wir zum ersten Baum zurück. Angenommen, wir löschen zuerst Schlüssel 2.



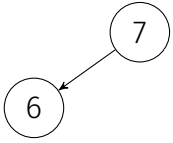
Schlüssel Löschen: Quiz Antwort

Löschen wir nun Schlüssel 4.



Schlüssel Löschen: Quiz Antwort

Schlüssel 7 ist nun die Wurzel des Baumes, nicht Schlüssel 6 wie zuvor. Wir haben gezeigt, dass das Löschen von Schlüsseln nicht immer kommutativ ist.



2. Lektionsübung

Lektionsübung: Alte Prüfungsaufgabe

In der folgenden Tabelle ist ein Min-Heap in seiner üblichen Form gespeichert. Wie sieht die Tabelle aus, nachdem die Zahl 17 eingefügt wurde?

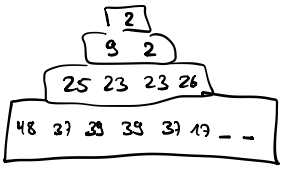
1	2	3	4	5	6	7	8	9	10	11	12
2	9	7	25	23	32	26	48	37	39	39	37

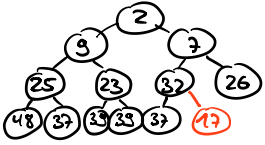
1	2	3	4	5	6	7	8	9	10	11	12	13

Lektionsübung: Alte Prüfungsaufgabe

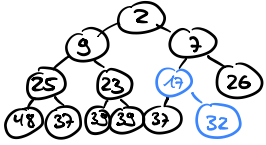
1	2	3	4	5	6	7	8	9	10	11	12	
2	9	7	25	23	32	26	48	37	39	39	37	17

Min Heap: Add 17





sift up



Liste: → 2 | 9 | 7 | 25 | 23 | 17 | 26 | 48 | 37 | 39 | 39 | 32

Lektionsübung: Alte Prüfungsaufgabe Lösung

In der folgenden Tabelle ist ein Min-Heap in seiner üblichen Form gespeichert. Wie sieht die Tabelle aus, nachdem die Zahl 17 eingefügt wurde?

1	2	3	4	5	6	7	8	9	10	11	12
2	9	7	25	23	32	26	48	37	39	39	37

1	2	3	4	5	6	7	8	9	10	11	12	13
2	9	7	25	23	17	26	48	37	39	39	37	32

3. Hausaufgaben

Übung 7: Trees

Auf `https://expert.ethz.ch/enrolled/SS25/mavt2/exercises`

Exercise 7: Trees

- Binary Search Trees and Heaps
- Implementing a Binary Search Tree
- Concatenating Heaps
- Heapsort

Abgabedatum: Montag 14.04.2025, 20:00 MEZ

KEINE HARDCODIERUNG

Fragen?