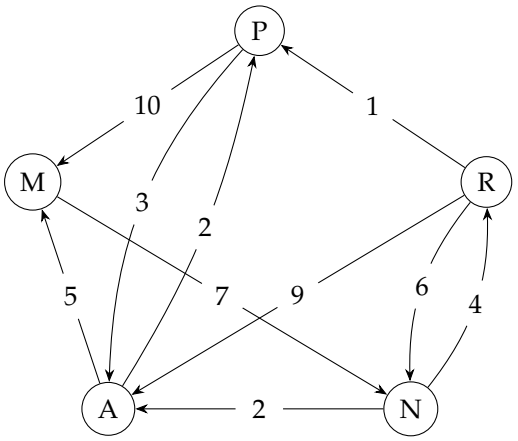
gerichteter Distanzgraph angegeben durch Adjazenzmatrix[1]

Ein gerichteter Distanzgraph sei durch seine Adjazenzmatrix gegeben (in einer Zeile stehen die Längen der von dem Zeilenkopf ausgehenden Wege.)[2]

$$\begin{array}{c} \\ M \\ A \\ P \\ R \\ N \end{array} \begin{array}{c} \begin{array}{ccccc} M & A & P & R & N \end{array} \\ \begin{pmatrix} - & 5 & 10 & - & - \\ - & - & 3 & 9 & 2 \\ - & 2 & - & 1 & - \\ - & - & - & - & 4 \\ 7 & - & - & 6 & - \end{pmatrix} \end{array}$$

(a) Stellen Sie den Graph in der üblichen Form dar.



(b) Bestimmen Sie mit dem Algorithmus von Dijkstra ausgehend von M die kürzeste Wege zu allen anderen Knoten.

Nach der Methode von Prof. Dr. Oliver Lazar

Schritt	Betrachteter Knoten	Kosten A	Kosten P	Kosten R	Kosten N
Initial	M				7
1	N	9		11	7
2	A	9	11	11	7
3	P	9	11	11	7
4	R	9	11	11	7

[1] **aud:ab:6**.
[2] **aud:ab:6**.

Nach der Methode aus der Vorlesung

Besuchte Knoten: M

Knoten-Name	M	A	P	R	N
Distanz	0	∞	∞	∞	∞
Vorgänger	null	null	null	null	null

Besuchte Knoten: M, N

Knoten-Name	M	A	P	R	N
Distanz	0	∞	∞	∞	7
Vorgänger	null	null	null	null	M

Besuchte Knoten: M, N, A

Knoten-Name	M	A	P	R	N
Distanz	0	9	∞	∞	7
Vorgänger	null	N	null	null	M

Besuchte Knoten: M, N, A, P

Knoten-Name	M	A	P	R	N
Distanz	0	9	11	∞	7
Vorgänger	null	N	A	null	M

Besuchte Knoten: M, N, A, P, R

Knoten-Name	M	A	P	R	N
Distanz	0	9	11	11	7
Vorgänger	null	N	A	N	M

Ergebnis

$M \to N = 7$
$M \to N \to A = 9$
$M \to N \to A \to P = 11$
$M \to N \to R = 11$

(c) Beschreiben Sie wie ein Heap als Prioritätswarteschlange in diesem Algorithmus verwendet werden kann.

Die Prioritätswarteschlange kann dazu verwendet werden, den Knoten mit der kürzesten Distanz schnell zu finden. Eine Prioritätswarteschlange kann zum Beispiel durch eine Min-Heap realisiert werden. Wenn eine Min-Heap aufgebaut wird, ist das Minium immer das Wurzelelement. Es kann sehr einfach und schnell entnommen werden. Der Aufbau einer Min-Heap geht mit linearem Zeitaufwand $\mathcal{O}(n)$ vonstatten. Die Entnahme des Miniums schlägt im schlechtes-

ten Fall mit einem Aufwand von $\mathcal{O}(\log n)$ zu Buche schlagen.

(d) Geben Sie die Operation *„Entfernen des Minimums“* für einen Heap an. Dazu gehört selbstverständlich die Restrukturierung des Heaps.

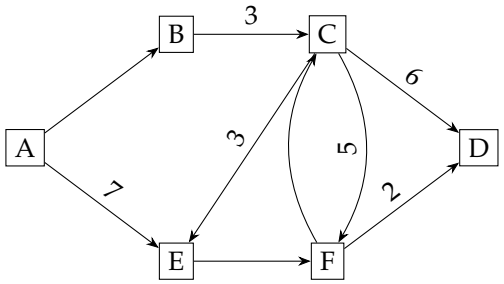
Bei der Entnahme des Minimums wird an dessen Stelle das am Ende der Halde sich befindende Element gesetzt.

Das neue Minimum verletzt unter Umständen die Heap-Eigenschaften, wenn eines oder beide seiner Kind-Knoten kleiner sind. Es muss mit dem kleinsten Kind-Knoten getauscht werden. Diese Prozedur wird rekursiv so lange ausgeführt, bis die Heap-Eigenschaften wieder hergestellt sind. Man nennt diesen Vorgang auch `heapify`.

Es kann aber auch sein das das verschobene Element kleiner ist. Dann muss die gegenteilige Operation von `heapify` ausführt werden, die `decrease` genannt wird.

Aufgabe 6[3]

Gegeben sei der folgende gerichtete Graph $G = (V, E, d)$ mit den angegebenen Kantengewichten.



(a) Geben Sie eine formale Beschreibung des abgebildeten Graphen G durch Auflistung von V, E und d an.

$G = (V, E, d)$

mit

$V = \{A, B, C, D, EF\}$

und

$E = \{(A, B), (A, E), (B, C), (C, D), (C, E), (C, F), (E, F), (F, C), (F, D), \}$

und

$d = \{1, 7, 3, 6, 3, 5, 1, 1, 2\}$

Als Adjazenzliste

[3] **46115:2012:03**.

A: $\rightarrow$ B $\xrightarrow{7}$ E

B: $\xrightarrow{3}$ C

C: $\xrightarrow{6}$ D $\xrightarrow{3}$ E $\xrightarrow{5}$ F

D:

E: $\rightarrow$ F

F: $\rightarrow$ C $\xrightarrow{2}$ D

(b) Erstellen Sie die Adjazenzmatrix A zum Graphen G.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} * & 1 & - & - & 7 & - \\ - & * & 3 & - & - & - \\ - & - & * & 6 & 3 & 5 \\ - & - & - & * & - & - \\ - & - & - & - & * & 1 \\ - & - & 1 & 2 & - & * \end{pmatrix} \end{array}$$

(c) Berechnen Sie unter Verwendung des Algorithmus nach Dijkstra - vom Knoten A beginnend - den kürzesten Weg, um alle Knoten zu besuchen. Die Restknoten werden in einer Halde (engl. Heap) gespeichert. Geben Sie zu jedem Arbeitsschritt den Inhalt dieser Halde an.

Nr.	**besucht**	**A**	**B**	**C**	**D**	**E**	**F**
0		0	∞	∞	∞	∞	∞
1	A	**0**	1	∞	∞	7	∞
2	B	|	**1**	4	∞	7	∞
3	C	|	|	**4**	10	7	9
4	E	|	|	|	10	**7**	8
5	F	|	|	|	10	|	**8**
6	D	|	|	|	**10**	|	|

nach	Entfernung	Reihenfolge	Pfad
A → A	0	1	
A → B	1	2	A → B
A → C	4	3	A → B → C
A → D	10	6	A → B → C → D
A → E	7	4	A → E
A → F	8	5	A → E → F

Aufgabe 7 Heap und binärer Suchbaum und AVL Baum[4]

Fügen Sie nacheinander die Zahlen 11, 1, 2, 13, 9, 10, 7, 5

(a) in einen leeren binären Suchbaum und zeichnen Sie den Suchbaum jeweils nach dem Einfügen von „9“ und „5“

Nach Einfügen von „9“:

Nach Einfügen von „5“:

(b) in einen leeren Min-Heap ein, der bzgl. „≤“ angeordnet ist und geben Sie den Heap nach „9“ und nach „5“ an

Nach dem Einfügen von „9“:

Nach dem Einfügen von „5“:

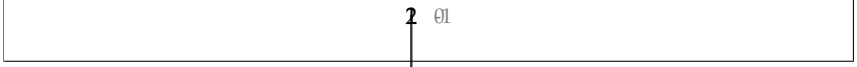
(c) in einen leeren AVL-Baum ein! Geben Sie den AVL Baum nach „2“ und „5“ an und beschreiben Sie die ggf. notwendigen Rotationen beim Einfügen dieser beiden Elemente!

Nach Einfügen von „2“:

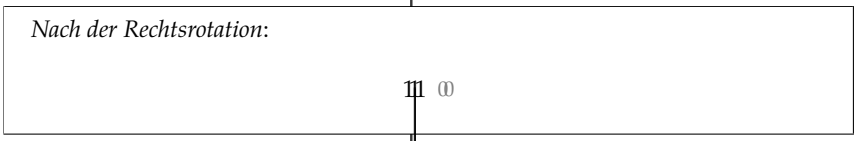
2 ~~0~~ 1

[4] **46115:2014:03**.

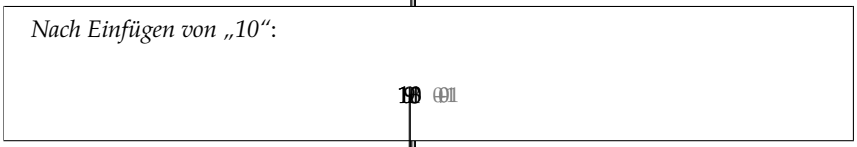
Nach der Linksrotation:



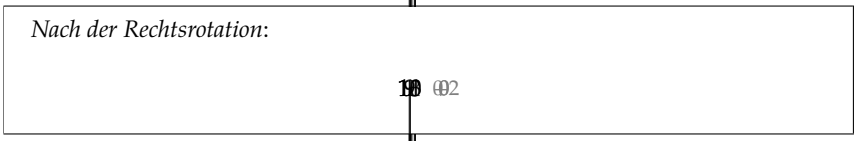
Nach der Rechtsrotation:



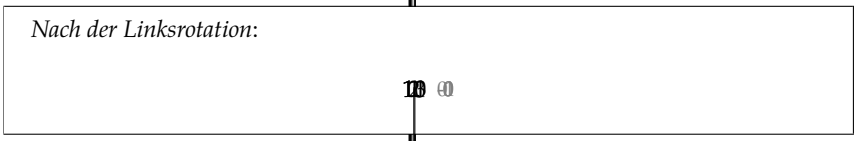
Nach Einfügen von „10“:



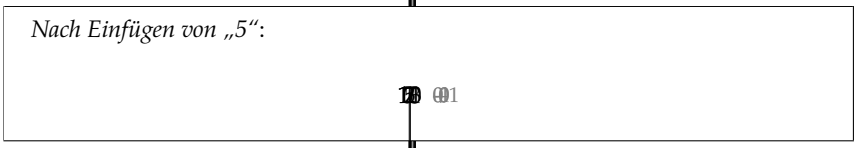
Nach der Rechtsrotation:



Nach der Linksrotation:



Nach Einfügen von „5“:



Aufgabe 6:[5]

Gegeben sei folgende Feld-Einbettung (Array-Darfstellung) einer Min-Halde:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0	2	3	7	6	5	4	8	9	10	11	12

(a) Stellen Sie die Halde graphisch als (links-vollständigen) Baum dar.

(b) Entfernen Sie das kleinste Element (die Wurzel 0) aus der obigen initialen Halde, stellen Sie die Haldeneigenschaft wieder her und geben Sie nur das Endergebnis in Felddarstellung an.

[5] **46115:2017:09**.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	6	3	7	10	5	4	8	9	12	11

(c) Fügen Sie nun den Wert 1 in die obige initiale Halde ein, stellen Sie die Haldeneigenschaft wieder her und geben Sie nur das Endergebnis in Felddarstellung an.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
0	2	1	7	6	3	4	8	9	10	11	12	5

Additum: zu b)

Entfernen von „0“:

Nach Vertauschen von „12“ und „2“:

Nach Vertauschen von „12“ und „6“:

Nach Vertauschen von „12“ und „10“:

Additum: zu c)

Nach dem Einfügen von „1“:

Nach Vertauschen von „1“ und „5“:

Nach Vertauschen von „1“ und „3“:

Aufgabe 7 (Heapify)[6]

Schreiben Sie in Pseudocode eine Methode `heapify(int[] a)`, welche im übergebenen Array der Länge n die Heapeigenschaft in $\mathcal{O}(n)$ Schritten herstellt. D. h. als Ergebnis soll in a gelten, dass $a[i] \leq a[2i+1]$ und $a[i] \leq a[i+2]$.

```java
import org.bschlangaul.helfer.Konsole;

/**
 * Nach Pseudocode nach
 * https://www.oreilly.com/library/view/algorithms-in-
 ↪  a/9780596516246/ch04s06.html
 */
public class Heapify {

  public static void buildHeap(int a[]) {
    int n = a.length;
    for (int i = n / 2 - 1; i >= 0; i--) {
      heapify(a, i, n);
    }
  }

  public static void heapify(int a[], int index, int max) {
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    int smallest;

    if (left < max && a[left] < a[index]) {
      smallest = left;
    } else {
      smallest = index;
    }

    if (right < max && a[right] < a[smallest]) {
      smallest = right;
    }

    if (smallest != index) {
      int tmp = a[index];
      a[index] = a[smallest];
      a[smallest] = tmp;
      heapify(a, smallest, max);
    }
  }

  public static void main(String[] args) {
    int[] a = new int[] { 5, 3, 16, 2, 10, 14 };
    buildHeap(a);
    Konsole.zeigeZahlenFeld(a); // 2 3 14 5 10 16
  }

```

[6] **46115:2019:09**.

```
47  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/Heapify.java`

Aufgabe 7[7]

Sei H ein Max-Heap, der n Elemente speichert. Für ein Element v in H sei $h(v)$ die Höhe von v, also die Länge eines längsten Pfades von v zu einem Blatt im Teilheap mit Wurzel v.

(a) Geben Sie eine rekursive Definition von $h(v)$ an, in der Sie sich auf die Höhen der Kinder v.left und v.right von v beziehen (falls v Kinder hat).

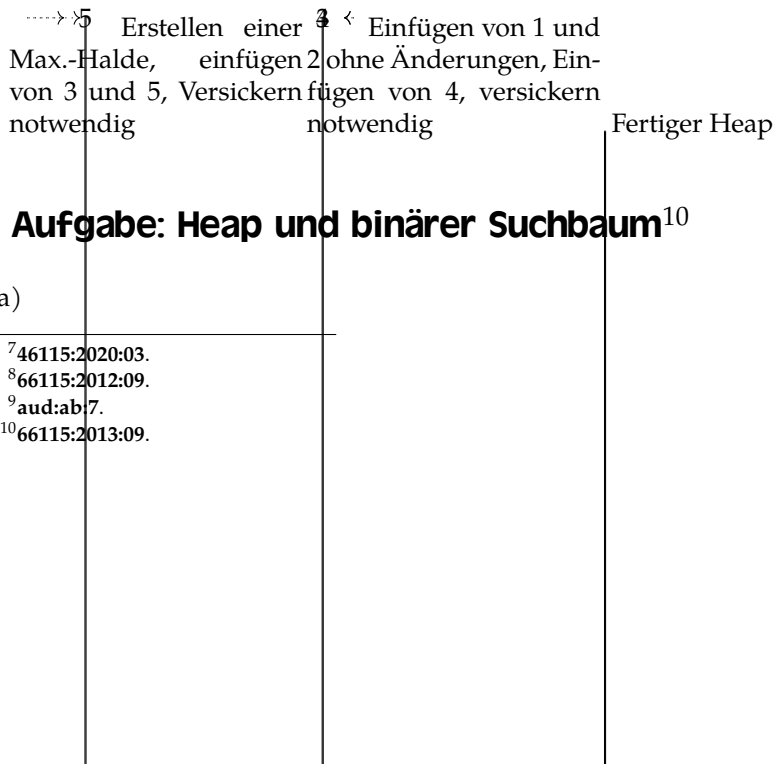
(b) Geben Sie eine möglichst niedrige obere asymptotische Schranke für die Summe der Höhen aller Elemente in H an, also für $\sum_{v \in H} h(v)$ und begründen Sie diese.

Tipp: Denken Sie daran, wie man aus einem beliebigen Feld einen Max-Heap macht.

(c) Sei H' ein Feld der Länge n. Geben Sie einen Algorithmus an, der in Linearzeit testet, ob H ein Max-Heap ist.

Aufgabe 7: Heap[8]

Fügen Sie nacheinander die Zahlen `3`, `5`, `1`, `2`, `4` in einen leeren Heap ein. Geben Sie die Ergebnisse an (Zeichnung).[9]



7. Aufgabe: Heap und binärer Suchbaum[10]

(a)

[7] **46115:2020:03**.
[8] **66115:2012:09**.
[9] **aud:ab:7**.
[10] **66115:2013:09**.

(i) Fügen Sie nacheinander die Zahlen 7, 1, 12, 8, 10, 3, 5 in einen leeren binären Suchbaum ein und zeichnen Sie den Suchbaum nach „8“ und nach „3“.

Nach Einfügen von „8“:

Nach Einfügen von „3“:

Nach Einfügen von „5“:

(ii) Löschen Sie die „1“ aus dem in (i) erstellten Suchbaum und zeichnen Sie den Suchbaum.

Nach Löschen von „1“:

(iii) Fügen Sie 7, 1, 12, 8, 10, 3, 5 in einen leeren MIN-Heap ein, der bzgl. „≤“ angeordnet ist. Geben Sie den Heap nach jedem Element an.

(b) Was ist die worst-case Laufzeit in O-Notation für das Einfügen eines Elements in einen Heap der Größe n? Begründen Sie ihre Antwort.

Aufgabe 6[11]

Gegeben sei ein einfacher Sortieralgorithmus, der ein gegebenes Feld A dadurch sortiert, dass er das *Minimum* m von A *findet*, dann das Minimum von A ohne das Element m usw.[12]

(a) Geben Sie den Algorithmus in Java an. Implementieren Sie den Algorithmus *in situ*, ðso, dass er außer dem Eingabefeld nur konstanten Extraspeicher benötigt. Es steht eine Testklasse zur Verfügung.

```
3  public class SortierungDurchAuswaehlen {
4    static void vertausche(int[] zahlen, int index1, int index2) {
5      int tmp = zahlen[index1];
6      zahlen[index1] = zahlen[index2];
7      zahlen[index2] = tmp;
8    }
9
10   static void sortiereDurchAuswählen(int[] zahlen) {
11     // Am Anfang ist die Markierung das erste Element im Zahlen-Array.
12     int markierung = 0;
13     while (markierung < zahlen.length) {
14       // Bestimme das kleinste Element.
```

[11] **aud:ab:3**.
[12] **examen:66115:2014:09**.

```
15        // 'min' ist der Index des kleinsten Elements.
16        // Am Anfang auf das letzte Element setzen.
17        int min = zahlen.length - 1;
18        // Wir müssen nicht bis letzten Index gehen, da wir 'min' auf
          ↪ das letzte Element
19        // setzen.
20        for (int i = markierung; i < zahlen.length - 1; i++) {
21          if (zahlen[i] < zahlen[min]) {
22            min = i;
23          }
24        }
25
26        // Tausche zahlen[markierung] mit gefundenem Element.
27        vertausche(zahlen, markierung, min);
28        // Die Markierung um eins nach hinten verlegen.
29        markierung++;
30      }
31    }
32
33    public static void main(String[] args) {
34      int[] zahlen = { 5, 2, 7, 1, 6, 3, 4 };
35      sortiereDurchAuswählen(zahlen);
36      for (int i = 0; i < zahlen.length; i++) {
37        System.out.print(zahlen[i] + " ");
38      }
39    }
40  }
```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/SortierungDurchAuswaehlen.java

(b) Analysieren Sie die Laufzeit Ihres Algorithmus.

Beim ersten Durchlauf des *Selectionsort*-Algorithmus muss $n-1$ mal das Minimum durch Vergleich ermittel werden, beim zweiten Mal $n-2$. Mit Hilfe der *Gaußschen Summenformel* kann die Komplexität gerechnet werden:

$$(n-1)+(n-2)+\cdots+3+2+1=\frac{(n-1)\cdot n}{2}=\frac{n^2}{2}-\frac{n}{2}\approx\frac{n^2}{2}\approx n^2$$

Da es bei der Berechnung des Komplexität um die Berechnung der asymptotischen oberen Grenze geht, können Konstanten und die Addition, Subtraktion, Multiplikation und Division mit Konstanten z. b. $\frac{n^2}{2}$ vernachlässigt werden.

Der *Selectionsort*-Algorithmus hat deshalb die Komplexität $\mathcal{O}(n^2)$, er ist von der Ordnung $\mathcal{O}(n^2)$.

(c) Geben Sie eine Datenstruktur an, mit der Sie Ihren Algorithmus beschleunigen können.

Der *Selectionsort*-Algorithmus kann mit einer Min- (in diesem Fall) bzw. einer Max-Heap beschleunigt werden. Mit Hilfe dieser Datenstruktur kann sehr schnell das Minimum gefunden werden. So kann

auf die vielen Vergleiche verzichtet werden. Die Komplexität ist dann $\mathcal{O}(n \log n)$.

Verständnis Suchbäume[13]

Wofür eignen sich die folgenden Baum-Datenstrukturen im Vergleich zu den anderen angeführten Baumstrukturen am besten, und warum. Sprechen Sie auch die Komplexität der wesentlichen Operationen und die Art der Speicherung an.

(a) Rot-Schwarz-Baum

Einfügen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Löschen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Suchen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall) [a]

[a] tutorialspoint.com

(b) AVL-Baum

Einfügen (Zeitkomplexität)
$\mathcal{O}(\log_2 n)$ (im Durchschnitt)
$\mathcal{O}(\log_2 n)$ (im schlechtesten Fall)

Löschen (Zeitkomplexität)
$\mathcal{O}(\log_2 n)$ (im Durchschnitt)
$\mathcal{O}(\log_2 n)$ (im schlechtesten Fall)

Suchen (Zeitkomplexität)
$\mathcal{O}(\log_2 n)$ (im Durchschnitt)
$\mathcal{O}(\log_2 n)$ (im schlechtesten Fall) [a]

[a] tutorialspoint.com

(c) Binärer-Heap

Verwendungszweck zum effizienten Sortieren von Elementen. [a]

Einfügen (Zeitkomplexität)
$\mathcal{O}(1)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Löschen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)

[13] **examen:66115:2016:03**.

$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Suchen (Zeitkomplexität)
$\mathcal{O}(n)$ (im Durchschnitt)
$\mathcal{O}(n)$ (im schlechtesten Fall) [b]

[a] deut. Wikipedia
[b] engl. Wikipedia

(d) B-Baum

Einfügen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Löschen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Suchen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall) [a]

[a] tutorialspoint.com

(e) R-Baum

Verwendungszweck Ein R-Baum erlaubt die schnelle Suche in mehrdimensionalen ausgedehnten Objekten. [a]

Suchen (Zeitkomplexität)
$\mathcal{O}(\log_M n)$ (im Durchschnitt) [b]
$\mathcal{O}(n)$ (im schlechtesten Fall) [c]

[a] deut. Wikipedia
[b] eng. Wikipedia
[c] Simon Fraser University, Burnaby, Kanada

Staatsexamenaufgabe zu Binärbaum, Halde, AVL[14]

(a) Fügen Sie die Zahlen 13, 12, 42, 3, 11 in der gegebenen Reihenfolge in einen zunächst leeren binären Suchbaum mit aufsteigender Sortierung ein. Stellen Sie nur das Endergebnis dar.[15]

(b) Löschen Sie den Wurzelknoten mit Wert 42 aus dem folgenden *binären* Suchbaum mit aufsteigender Sortierung und ersetzen Sie ihn dabei durch einen geeigneten Wert aus dem *rechten* Teilbaum. Lassen Sie möglichst viele Teilbäume unverändert und erhalten Sie die Suchbaumeigenschaft.[16]

[14] **examen:66115:2017:09**.
[15] **examen:66115:2017:09**.
[16] **examen:66115:2017:09**.

(c) Fügen Sie einen neuen Knoten mit dem Wert 13 in die folgende Min-Halde ein und stellen Sie anschließend die Halden-Eigenschaft vom neuen Blatt aus beginnend wieder her, wobei möglichst viele Knoten der Halde unverändert bleiben und die Halde zu jedem Zeitpunkt links-vollständig sein soll. Geben Sie nur das Endergebnis an.

(d) Geben Sie für die ursprüngliche Min-Halde aus Teilaufgabe c) (ohne den neu eingefügten Knoten mit dem Wert 13) die Feld-Einbettung (Array-Darstellung) an.

(e) Löschen Sie den Wurzelknoten mit Wert 42 aus der folgenden Max-Halde und stellen Sie anschließend die Halden-Eigenschaft ausgehend von einer neuen Wurzel wieder her, wobei möglichst viele Knoten der Halde unverändert bleiben und die Halde zu jedem Zeitpunkt links-vollständig sein soll. Geben Sie nur das Endergebnis an.

(f) Fügen Sie in jeden der folgenden AVL-Bäume mit aufsteigender Sortierung jeweils einen neuen Knoten mit dem Wert 13 ein und führen Sie anschließend bei Bedarf die erforderliche(n) Rotation(en) durch. Zeichnen Sie den Baum vor und nach den Rotationen.

   (i) AVL-Baum A

   (ii) AVL-Baum B

Aufgabe 3[17]

Es sei der folgende Min-Heap gegeben:

(a) Geben Sie obigen Min-Heap in der Darstellung eines Feldes an, wobei die Knoten in Level-Order abgelegt sind.

0	**1**	**2**	**3**	**4**	**5**
10	20	15	40	25	35

(b) Führen Sie wiederholt DeleteMin-Operationen auf dem gegebenen Heap aus, bis der Heap leer ist. Zeichnen Sie dafür den aktuellen Zustand des Heaps als Baum und als Feld nach jeder Änderung des Heaps, wobei Sie nur gültige Bäume zeichnen (d. h. solche die keine Lücken haben). Dokumentieren Sie, was in den einzelnen Schritten geschieht.

Löschen von 10

[17] **66115:2020:09**.

Nach dem Ersetzen von „10“ durch „35“:

0	**1**	**2**	**3**	**4**
35	20	15	40	25

Nach Vertauschen von „35“ und „15“:

0	**1**	**2**	**3**	**4**
15	20	35	40	25

Löschen von 15

Nach dem Ersetzen von „15“ mit „25“:

0	**1**	**2**	**3**
25	20	35	40

Nach Vertauschen von „25“ und „20“:

0	**1**	**2**	**3**
20	25	35	40

Löschen von 20

Nach dem Vertauschen von „20“ mit „40“:

0	**1**	**2**
40	25	35

Nach Vertauschen von „40“ und „25“:

0	**1**	**2**
25	40	35

Löschen von 25

Nach dem Ersetzen von „25“ durch „35“:

0	**1**
35	40

Löschen von 35

Nach dem Ersetzen von „35“ mit „40“:

0
40

Löschen von 40

Nach dem Löschen von „40“: