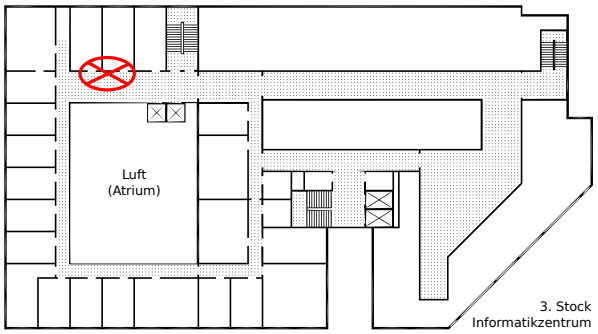
Algorithmen und Datenstrukturen **Winter 2019/20**
Prof. Dr. Sándor P. Fekete **Abgabe:** 13.01.2020
Arne Schmidt **Rückgabe:** 20.–24.01.2020

Übungsblatt 4

Abgabe der Lösungen bis zum 13.01.2020 um 10:00 Uhr im Hausaufgabenschrank bei Raum IZ 337 (siehe Skizze rechts). Es werden nur mit einem dokumentenechten Stift (kein Rot!) geschriebene Lösungen gewertet. **Bitte die Blätter zusammenheften und vorne deutlich mit eigenem Namen, Matrikel- und Gruppennummer, sowie Studiengang versehen!**



Beachte: Bei der Bearbeitung der Hausaufgaben gelten folgenden Richtlinien:
`https://www.ibr.cs.tu-bs.de/courses/ws1920/aud/HA-Hinweise.pdf`

Dieses Blatt besteht aus einer Präsenzaufgabe, die in der kleinen Übung besprochen wird, sowie aus zwei Hausaufgaben, die abgegeben werden müssen und bewertet werden.

Präsenzaufgabe: (Besprechung 20.-24.01.2020)
BUBBLESORT (siehe Algorithmus 1) ist ein Sortieralgorithmus, welcher ein Array $n-1$ mal durchläuft und dabei benachbarte Felde vertauscht, falls die Elemente falsch geordnet sind.

a) Führe BUBBLESORT auf folgendem Array aus.

$$A = [7, 2, 4, 1, 6, 5, 8, 3]$$

b) Begründe kurz die Korrektheit von BUBBLESORT.

c) Zeige: Jeder beliebige Sortieralgorithmus, welcher nur benachbarte Felder vertauschen darf, besitzt eine Laufzeit von $\Omega(n^2)$.

```
1: function BUBBLESORT(A)
2:     for i = n downto 2 do
3:         for j = 1 to i − 1 do
4:             if A[j] > A[j + 1] then
5:                 Vertausche A[j]und A[j + 1]
```

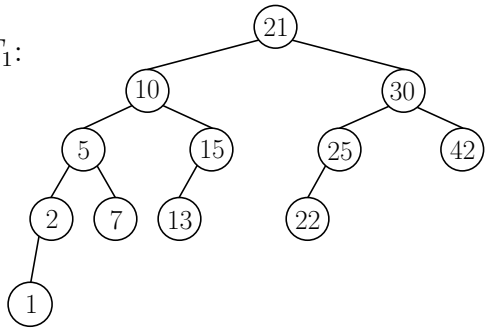
Algorithmus 1: Der BUBBLESORT-Algorithmus.

Hausaufgabe 1 (AVL-Bäume): **(3+2+2+2+4 Punkte)**

Betrachte in den Aufgabenteilen a) bis d) den Baum, der in der jeweiligen Abbildung dargestellt ist. Führe die Operation des jeweiligen Aufgabenteils und die damit verbundenen Restrukturierungsmaßnahmen zum Erhalt der AVL-Eigenschaft auf dem entsprechenden Baum aus. Zeichne dabei das Resultat nach jeder einzelnen ausgeführten Operation INSERT, DELETE und RESTRUCTURE in einen separaten Baum:

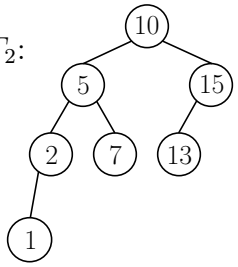
a) DELETE($T_1, 42$)

T_1:



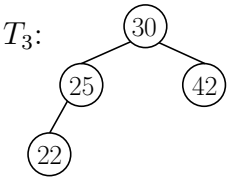
b) INSERT($T_2, 12$)

T_2:



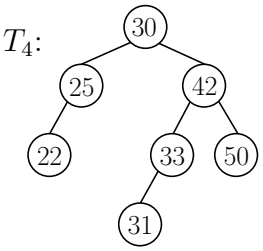
c) INSERT($T_3, 23$)

T_3:



d) INSERT($T_4, 32$)

T_4:



e) Zeige mit vollständiger Induktion, dass ein AVL-Baum der Höhe h mindestens $F_{h+2} - 1$ Knoten enthält.
(Hinweis: F_n beschreibt die n-te Fibonacci-Zahl mit $F_0 = 0$, $F_1 = 1$ und $F_n = F_{n-1} + F_{n-2}$.)

Hausaufgabe 2 (Max-Heaps): **(7 Punkte)**

Ein *Max-Heap* ist ein binärer Baum mit folgenden Eigenschaften:

- Jeder Knoten besitzt einen Schlüssel
- Ebene $i < h$ besitzt 2^{i-1} Knoten, wobei h die Höhe des Baumes ist.
- Auf Ebene h sind die linken $n - 2^{h-1} + 1$ Positionen besetzt.
- Der Schlüssel jedes Knotens ist mindestens so groß wie die seiner beiden Kinder.

Ein Max-Heap kann sowohl als Array, als auch in einer Baumstruktur angegeben werden (siehe Abbildung 1).
(Hinweis: Max-Heaps werden auch in der Vorlesung am 07.01.2020 vorgestellt.)

Betrachte das Array $A = [2, 11, 1, 3, 4, 9, 10, 8, 7, 5, 6, 12]$. Führe BUILD-MAXHEAP (siehe Algorithmus 2 links) auf A aus und gib A als Array oder in Baumstruktur nach **jeder** Tauschoperation zweier Elemente an.
(Hinweis: Das Element im Feld i besitzt als Kinder die Elemente in den Feldern $2i$ und $2i + 1$.)

$B = [12, 10, 11, 9, 6, 8, 1, 4, 7, 2, 5, 3]$

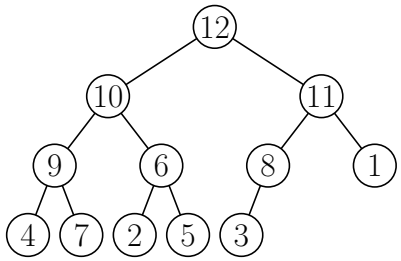


Abbildung 1: Links: Max-Heap als Array. Rechts: Derselbe Max-Heap in Baumstruktur.

```
1: function BUILD-MAX-HEAP(A)
2:     heap-größe[A] := länge[A]
3:     for i = ⌊länge[A]/2⌋ down to 1 do
4:         MAX-HEAPIFY(A, i)
```

```
1: function MAX-HEAPIFY(A, i)
2:     ℓ := links(i), r := rechts(i)
3:     if ℓ ≤ heap-größe[A] und A[ℓ] > A[i] then
4:         max := ℓ
5:     else
6:         max := i
7:     if r ≤ heap-größe[A] und A[r] > A[max] then
8:         max := r
9:     if max ≠ i then
10:        Vertausche A[max]und A[i]
11:        MAX-HEAPIFY(A, max)
```

Algorithmus 2: Links: Algorithmus zur Konstruktion eines Max-Heaps aus einem Array. Rechts: Subroutine, um einen Knoten nach unten zu tauschen, falls dieser größere Kinder besitzt. Dabei ist links$(i) := 2i$ und rechts$(i) := 2i + 1$.