Prof. Dr. Ir. Joost-Pieter Katoen

Sebastian Junges, Benjamin Kaminski, David Korzeniewski, Tim Quatmann

Präsenzübung Datenstrukturen und Algorithmen SS 2018

Vorname: _____

Nachname: _____

Matrikelnummer: _____

Studiengang (bitte **genau** einen markieren):

- ○ Informatik Bachelor
- ○ Informatik Lehramt (Bachelor)
- ○ Sonstiges: _____
- ○ Mathematik Bachelor
- ○ CES Bachelor

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	**15**	
Aufgabe 2	**15**	
Aufgabe 3	**15**	
Aufgabe 4	**18**	
Aufgabe 5	**12**	
Aufgabe 6	**15**	
Summe	**90**	

Hinweise:

- Auf **alle Blätter** (inklusive Zusatzblätter) müssen Sie **Ihre Matrikelnummer** schreiben.
- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- Schreiben Sie mit **dokumentenechten** Stiften, nicht mit roten oder grünen Stiften und nicht mit Bleistiften.
- Bitte beantworten Sie die Aufgaben auf den Aufgabenblättern.
- Geben Sie für jede Aufgabe **maximal eine** Lösung an. Streichen Sie alles andere durch. Andernfalls werden alle Lösungen der Aufgabe mit **0 Punkten** bewertet.
- Werden **Täuschungsversuche** beobachtet, so wird die Übung mit **0 Punkten** bewertet.
- Geben Sie am Ende der Übung **alle Blätter zusammen mit den Aufgabenblättern ab**.

Aufgabe 1 ($\mathcal{O}$-Notation): (3 + 7 + 5 = 15 Punkte)

Rufen Sie sich die Quasiordnung $\sqsubseteq$, definiert als

$$f \sqsubseteq g \quad \text{genau dann wenn} \quad f \in \mathcal{O}(g) \ ,$$

in Erinnerung.

a) Beweisen oder widerlegen Sie, dass $\sqsubseteq$ anti-symmetrisch ist.

b) Sortieren Sie die folgenden 15 Funktionen

$$\log(n^n), \quad \sum_{i=0}^{n} \frac{2i^3}{1+i}, \quad 2^n, \quad n^2, \quad 3^n, \quad 2^{9000 \cdot n}, \quad n^2 \cdot \log(n), \quad n^n,$$

$$n \cdot \sqrt[3]{n}, \quad n \cdot \log(n), \quad n!, \quad \log(n^2), \quad 0, \quad n \cdot \sqrt{n}, \quad \frac{n^2}{2000}$$

in aufsteigender Reihenfolge bezüglich der Quasiordnung $\sqsubseteq$. Zeigen Sie auch durch $f \approx g$ an, wenn sowohl $f \sqsubseteq g$ als auch $g \sqsubseteq f$ gilt. Schreiben Sie also beispielsweise

$$f \sqsubseteq g \approx h \sqsubseteq i \qquad \text{falls} \qquad f \in \mathcal{O}(g) \ , \quad g \in \mathcal{O}(h) \text{ und } h \in \mathcal{O}(g) \ , \quad \text{und} \quad h \in \mathcal{O}(i) \ .$$

c) Beweisen oder widerlegen Sie: $o(f) \cap \Theta(f) = \emptyset$.

Aufgabe 2 (Sortieren): (4 + 5 + 6 = 15 Punkte)

a) Sortieren Sie das folgende Array mithilfe von Mergesort. Geben Sie dazu das Array nach jeder Merge-Operation an. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

5	23	7	2	8	17	9	3

b) Quicksort ist im allgemeinen einer der effizientesten Sortieralgorithmen. Leider benötigt er aber quadratische Laufzeit, wenn immer das größte oder kleinste Element als Pivot ausgewählt wird. Wie kann Quicksort so angepasst werden, dass der Algorithmus auf fast sortierten Eingaben dennoch effizient ist? Begründen Sie ihre Antwort.

Wir betrachten eine Eingabe als fast sortiert, wenn nur sehr wenige Elemente an zufälliger Position nicht korrekt einsortiert sind.

c) Ein naiver Algorithmus um das häufigste Element in einem Array der Länge $n \geq 1$ zu finden, benötigt $\Theta(n^2)$ Zeit und konstant viele Array-Indizes als Speicherplatz. Der naive Algorithmus zählt für jedes Element im Array, wie häufig es vorkommt und speichert sich jeweils die Position des bisher häufigsten Elements und seine Häufigkeit.

Beschreiben Sie einen Algorithmus, der das häufigste Element in einem Array der Länge $n > 1$ findet und dessen asymptotische Laufzeit in $O(n \log n)$ liegt. Außerdem darf der Algorithmus nur konstant mehr Speicher benötigen als ein naiver Algorithmus. Begründen Sie ihre Antwort!

Aufgabe 3 (Sortieren): (4 + 5 + 6 = 15 Punkte)

a) Sortieren Sie das folgende Array mit Hilfe von Insertionsort. Geben Sie dazu das Array nach jeder Iteration der äußeren Schleife an. Gehen Sie strikt nach dem Verfahren aus der Vorlesung vor. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

5	4	0	8	2	4	7

b) Geben Sie ein Array der Länge 4 an, sodass der Quicksort Algorithmus aus der Vorlesung diese Eingabe nicht stabil sortiert.

c) Es sei `sortHeap(int E[])` ein vergleichsbasierter Sortieralgorithmus, der als Eingabe einen Max-Heap (repräsentiert als Array `E`) bekommt und daraufhin das Eingabearray aufsteigend sortiert.

Bestimmen Sie eine asymptotische Worst-Case Laufzeit $f(n)$ in Abhängigkeit der Eingabelänge $n = \texttt{E.length}$, die ein solcher Algorithmus haben kann, sodass es

(i) **eine** Implementierung von `sortHeap(int E[])` gibt, dessen Worst-Case Laufzeit in $\mathcal{O}(f(n))$ liegt und

(ii) **keine** Implementierung von `sortHeap(int E[])` gibt, dessen Worst-Case Laufzeit in $o(f(n))$ liegt.

Begründen Sie informell, warum (i) und (ii) für Ihre angegebene Funktion gelten.

$f(n) =$ _____

Begründung zu (i):

Begründung zu (ii):

Aufgabe 4 (Rekursionsgleichungen): (4 + 4 + 5 + 5 = 18 Punkte)

Gegeben sei die Rekursionsgleichung

$$\begin{aligned} T(1) &= 1 \,, \\ T(n) &= 2T\left(\frac{n}{3}\right) + n^2 \,. \end{aligned}$$

a) Zeichnen Sie einen Rekursionsbaum für die Rekursionsgleichung T. Dieser soll mindestens die obersten 3 Ebenen enthalten, sowie eine Ebene für die Blätter.

b) Stellen Sie anhand des Rekursionsbaumes eine Summenformel für die sich ergebenden Gesamtkosten auf. Bestimmen Sie dazu die asymptotische Tiefe des Baumes, die asymptotische Breite der Ebene der Blätter, die akkumulierten Kosten einer beliebigen Ebene i, die nicht der Blätterebene entspricht, und letztendlich die sich ergebenden Gesamtkosten.

c) Bestimmen Sie aus der obigen Summenformel eine geschlossene Form für die asymptotischen Gesamtkosten. Zeigen Sie mit Hilfe einer beliebigen, Ihnen aus Vorlesung oder Übung bekannten, Methode, dass Ihre gefundene Schranke tatsächlich eine obere Schranke für eine Lösung der Rekursionsgleichung ist.

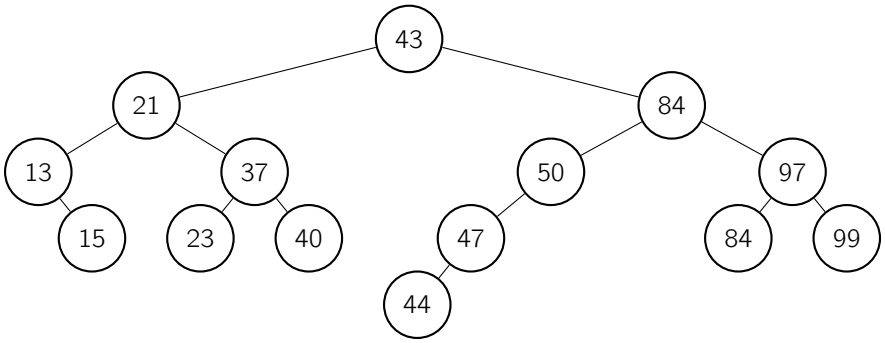
d) Lösen Sie die folgende Rekursionsgleichung

$$\begin{aligned} T(1) &= 1 \,, \\ T(n) &= 3T\left(\frac{n}{2}\right) + n^2 \,. \end{aligned}$$

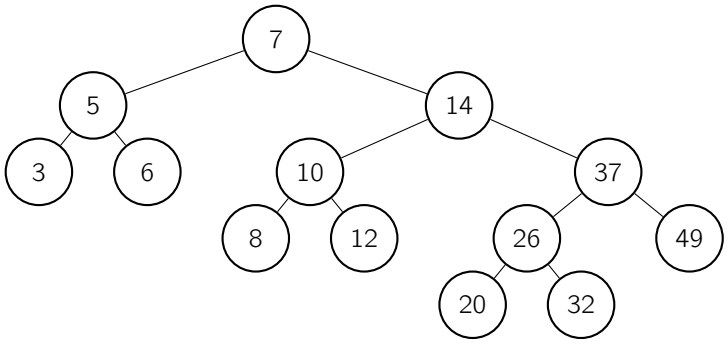
mit Hilfe des Mastertheorems.

Aufgabe 5 (Suchbäume): (2 + 3 + 4 + 3 = 12 Punkte)

a) Fügen Sie den Knoten mit Schlüssel 20 in den folgenden Binären Suchbaum (BST) ein. Nutzen Sie dazu das entsprechende Verfahren aus der Vorlesung.



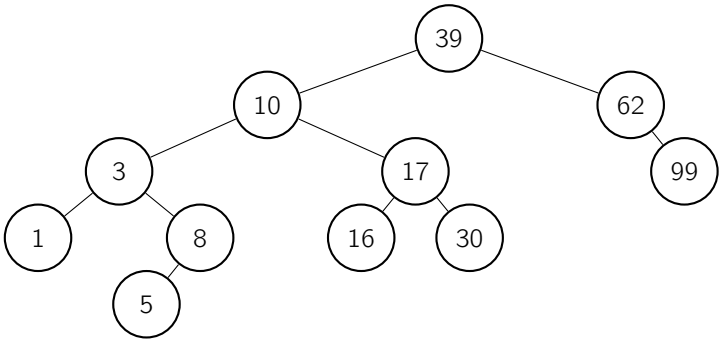
b) Löschen Sie den Knoten mit Schlüssel 14 aus dem folgenden Binären Suchbaum (BST). Nutzen Sie dazu das entsprechende Verfahren aus der Vorlesung.



c) Erstellen Sie einen AVL-Baum, der die folgenden Schlüssel enthält.

37	72	90	61	6	18	64	5

d) Führen Sie genau eine Rotation aus, um den folgenden BST in einen (balancierten) AVL-Baum zu überführen. Geben Sie den resultierenden AVL-Baum an sowie den Knoten auf dem die Rotation ausgeführt wurde und ob es sich um eine Links- oder Rechtsrotation handelt.



Aufgabe 6 (Laufzeit-Analyse): (3 + 3 + 3 + 3 + 3 = 15 Punkte)

Bestimmen Sie von folgenden Funktionen die Worst-Case Laufzeit. Geben Sie dazu eine geschlossene form für die asymptotische Anzahl von Print-Aufrufen an.

a)

```
A(n): Integer -> void

for i in 1..n:
    for j in 1..n:
        for k in 1..n:
            if i = k & j > i:
                print(..)
```

b)

```
B(n): Integer -> void

m = n
c = 0
while m > 1:
    m = m/2
    c = c + 1
    print(..)
while c > 0:
    c = c - 1
    for i in 1..n:
        print(..)
```

c)

```
C(n): Integer -> void

for i in 1..n:
    for j in 1..10:
        h = j*j
        for k in 1..h^j:
            print(..)
```

d)

```
D(n): Integer -> void

for i in 1..n:
    print(..)
    for j in 1..n:
        if i < 25:
            if j < i:
                print(..)
```

e) Hinweis: In dieser Teilaufgabe brauchen Sie keine geschlossene Form anzugeben. Eine Summe reicht!

```
PrimTest(n): Integer -> void

for i in 1..n:
    if i*i < n:
        t = n
        while t > 0:
            t = t - i
            print(Vielleicht)
        if t = 0:
            print(Ja)
        else:
            print(Nein)
```

Dies ist die letzte Seite.