

Name: Ulges Vorname: Adrian

Matrikelnummer: \_\_\_\_\_ Unterschrift: \_\_\_\_\_

*Mit meiner Unterschrift bestätige ich, dass ich die Anmerkungen unten zur Kenntnis genommen, die Aufgaben eigenständig gelöst, sowie nur die zugelassenen Hilfsmittel verwendet habe.*

- Die Klausurdauer beträgt **90 Minuten**.
- Bitte legen Sie **Studierendenausweis** und **Lichtbildausweis** auf Ihren Tisch.
- Bitte schreiben Sie **deutlich**. Unleserliche Lösungen werden nicht gewertet. Die **Bindung** der Blätter dieser Klausur **darf nicht entfernt** werden. Sie dürfen auch die **Rückseiten** der Blätter verwenden (weiteres Schmierpapier befindet sich am Ende).
- Lesen Sie die Aufgabenstellungen **vollständig**. Sollten während der Klausur Unklarheiten bestehen, ist es möglich kurze **Fragen** zu stellen.
- Es sind **keine Hilfsmittel** zugelassen. Entfernen Sie Mobiltelefone, Vorlesungsmitschriften, sonstige lose Blätter und Bücher von Ihrem Tisch.
- Täuschungsversuche aller Art werden mit der **Note 5** geahndet.
- Beachten Sie insbesondere, dass **elektronische Geräte** (z.B. Mobiltelefone, Smartwatches oder Kameras) **unerlaubte Hilfsmittel** sind! Bereits das **Berühren** eines nicht erlaubten Hilfsmittels während der Prüfung stellt einen Täuschungsversuch dar.
- **Toilettengänge** während der Prüfung kosten Ihre Zeit und schaffen für alle Unruhe. Erledigen Sie sie möglichst vor der Prüfung. Wenn es trotzdem sein muss: Es darf immer nur einer gleichzeitig. Melden Sie sich bei der Aufsicht an und warten Sie auf das OK.

Viel Erfolg!

[illegible]

Matrikelnummer: \_\_\_\_\_

**Aufgabe 1 (2+4+7 = 13 Punkte)**

```
# Eingabe
# a: ein Array von Zahlen
# n: die Länge des Arrays
1  s = 0;
2  for (int i=0; i<n; i+=1) {
3      s += a[i];
4  }
5  avg = s/n;
6  s = 0;
7  for (int i=0; i<n; i+=1) {
8      if (a[i] <= a[0]) {
9          s += 2 * avg;
10     } else {
11         s += 2 * a[i];
12     }
13 }
14 return s;
```

} immer n Feldzugriffe (a)

} immer 2·n Feldzugriffe (b)

} (\*)

- a) Der obige Algorithmus wird auf einem Array  $a$  der Länge  $n \geq 1$  ausgeführt. Terminiert der Algorithmus für alle möglichen Eingabe-Arrays? Ist der Algorithmus deterministisch? Begründen Sie jeweils kurz.

- Terminiert? Ja! In beiden Schleifen (2,7) erreicht die Variable  $i$  nach  $n$  Durchläufen den Wert  $n$ , und die Schleife wird verlassen.
- Deterministisch? Ja! Wird dasselbe Array  $a$  übergeben, werden exakt dieselben Schritte durchlaufen.

- b) Berechnen Sie nachvollziehbar die genaue Anzahl der Feldzugriffe in Abhängigkeit von  $n$ , und zwar für den Best Case und den Worst Case. Geben Sie abschließend auch noch für beide Fälle die zugehörige Aufwandsklasse in O-Notation an.

- Immer: Schleife (a)  $\rightarrow n$  Feldzugriffe  
" " (b)  $\rightarrow 2n$  "
- Best Case: else (\*) wird nur für  $i = 0$  ausgeführt  $\rightarrow +1$  Feldzugriff
- Worst Case: else (\*) wird  $n$ -mal ausgeführt  $\rightarrow +n$  " e

$\Rightarrow$  Best Case =  $3n + 1 \in O(n)$ , Worst Case:  $4n \in O(n)$

- c) Füllen Sie die folgende Tabelle aus: Tragen Sie in die leeren Felder jeweils eine Funktion  $a_n$  oder  $b_n$  ein, so dass die Bedingungen in den drei rechten Spalten erfüllt sind. Sollte keine solche Funktion existieren, tragen Sie einen Strich ein.  
*Hinweis: Es ist keine Herleitung erforderlich.*

$a_n$	$b_n$	$a_n \in O(b_n)$	$b_n \in \Theta(a_n)$	$a_n \in \Omega(b_n)$
$2 \cdot n^2 + 100$	/	ja	ja	nein ⚡
$3n^3 + 4$	$n^4$	ja	nein	nein
$\log(n)$	1	nein	nein	ja
$3n^2$	$\frac{n^3+3}{n+5}$	ja	ja	ja
/	$\log(n) \cdot 2^n$	nein	nein	nein
$n^4 + 4^n$	$5^n$	ja	nein	nein
$n^2 \cdot 2^n$	$3^n$	ja	nein	nein
$n^3 - n^2$	/	ja	nein	ja
$n \cdot \log(n) - n$	$n$	nein	nein	ja

Matrikelnummer: \_\_\_\_\_

## Aufgabe 2 (3+3+3+3+3 = 15 Punkte)

Sind die folgenden Behauptungen korrekt? Kreuzen Sie an. Geben Sie (falls ja) eine knappe Begründung oder (falls nein) ein Gegenbeispiel an.

- a) Gilt  $a_n \in O(b_n)$ , gilt auch  $a_n \in \Theta(b_n)$ .

☐ gilt ☒ gilt nicht

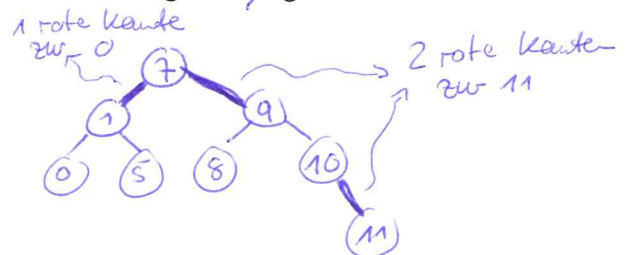
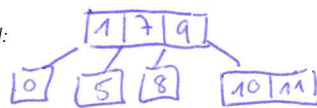
Begründung/Gegenbeispiel:

$a_n = n$ ,  $b_n = n^2$  (wächst nicht gleich schnell)

- b) In einem Red-Black-Tree ist die Anzahl der roten Kanten von der Wurzel zu allen Blättern gleich.

☐ gilt ☒ gilt nicht

Begründung/Gegenbeispiel:



- c) Bei gleicher Tabellengröße und Eingabedaten benötigt Hashing mit Verkettung immer mindestens so viel Speicher wie Hashing mit Sondierung.

☒ gilt ☐ gilt nicht

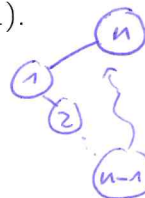
Begründung/Gegenbeispiel:

Es wird zusätzlicher Platz für Listen benötigt.

- d) Das Entfernen der Wurzel eines Suchbaums mit  $n$  Elementen hat immer den Aufwand  $O(1)$ .

☐ gilt ☒ gilt nicht

Begründung/Gegenbeispiel:



$O(n)$ , denn Element  $n-1$  muss gefunden und bewegt werden.

- e) Das Quicksort-Verfahren besitzt für alle Arrays der Länge  $n$  einen Aufwand von  $O(n \cdot \log(n))$ .

☐ gilt ☒ gilt nicht

Begründung/Gegenbeispiel:



Daten aufsteigend sortiert  
 $\rightarrow \Theta(n^2)$

Matrikelnummer: \_\_\_\_\_

**Aufgabe 3 (6+3+6 = 15 Punkte)**

- a) Sortieren Sie das folgende Array aufsteigend mit **RadixExchangeSort**. Führen Sie die Vertauschungen gemäß dem Schema aus der Veranstaltung durch. Überführen Sie die Zahlen hierzu zunächst in **Binärdarstellung (4 Bit)**. Behandeln Sie in **jeder Zeile** eine Binärziffer (ein Bit).

2	9	6	11	3	1
---	---	---	----	---	---

Binärdarstellung:

0010	1001	0110	1011	0011	0001
------	------	------	------	------	------

1. Bit

0010	0001	0110	0011	1011	1001
------	------	------	------	------	------

2. Bit

0010	0001	0011	0110	1011	1001
------	------	------	------	------	------

3. Bit

0001	0010	0011	0110	1001	1011
------	------	------	------	------	------

4. Bit

0001	0010	0011	0110	1001	1011
------	------	------	------	------	------

Endergebnis:

1	2	3	6	9	11
---	---	---	---	---	----

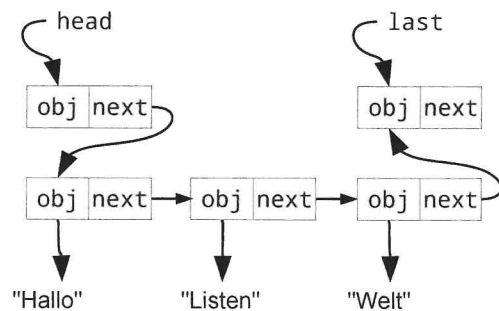




Matrikelnummer: \_\_\_\_\_

#### Aufgabe 4 (12+4 = 16 Punkte)

```
class LinkedList<T> {  
    private Node head;  
    private Node last;  
  
    private class Node {  
        T obj;  
        Node next;  
    }  
    ...  
}
```



Verkettete Listen bestehen aus Knoten (Nodes) und besitzen für Anfang und Ende separate Knoten head und last. Jeder Knoten referenziert seinen Nachfolger (next).

- a) Implementieren Sie eine Methode `isLongerThan(LinkedList<T> other)`, die eine Liste mit einer anderen Liste `other` vergleicht und genau dann `true` zurückliefert, wenn die Liste `this` mehr Knoten enthält als `other`. Vermeiden Sie es wenn möglich, die Länge beider Listen zu berechnen.

```
boolean isLongerThan(LinkedList<T> other) {
```

```
    Node n1 = this.head;
```

```
    "    n2 = other.head;
```

```
    while (n1 != null && n2 != null) {
```

```
        n1 = n1.next;
```

```
        n2 = n2.next;
```

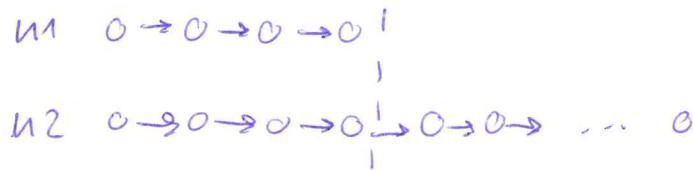
```
    }
```

```
    return (n1 != null);
```

// n2 muss null sein  
→ Ende von other erreicht!

```
}
```

- b) Geben Sie die Aufwandsklasse Ihrer Methode `isLongerThan()` in Abhängigkeit der Listenlängen  $n_1$  (this) und  $n_2$  (other) an. Begründen Sie Ihre Antwort.



↑  
Hier wird abgebrochen  
(Ende der kürzeren Liste erreicht)

- # Schleifendurchläufe:  $\min(n_1, n_2)$
- pro Durchlauf =  $O(1)$

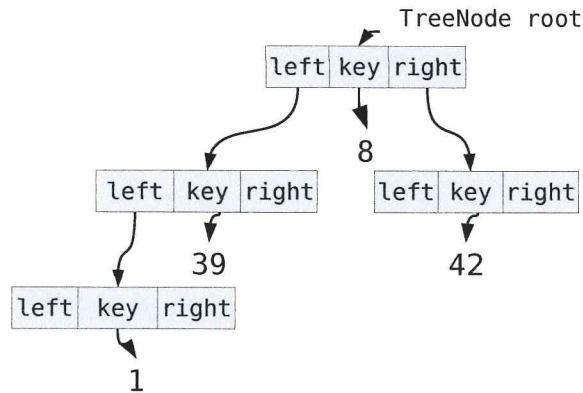
Insgesamt:  $O(\min(n_1, n_2))$



Matrikelnummer: \_\_\_\_\_

### Aufgabe 5 (9+9 = 18 Punkte)

```
class TreeNode {  
    int key;  
    TreeNode left;  
    TreeNode right;  
  
    int maxKey();  
  
    ...  
}
```



Binäre Bäume seien Knoten (Nodes) mit int-Schlüsseln und Referenzen auf ein linkes und rechtes Kind (siehe oben).

- a) Schreiben Sie eine rekursive Funktion `maxKey()`, die beim Aufruf auf einem Wurzelknoten (z.B. `root.maxKey()`) den größten key des gesamten Baums (incl. root) zurückliefert.

*Hinweis: `maxKey()` soll für beliebige binäre Bäume funktionieren, nicht nur für Suchbäume!*

```
int maxKey() {
```

```
    int result = this.key();  
    if (left != null) {  
        int l = left.maxKey();  
        result = l > result ? l : result;  
    }  
    if (right != null) {  
        int r = right.maxKey();  
        result = r > result ? r : result;  
    }  
    return result;  
}
```

- b) Fügen Sie zur Klasse Node eine rekursive Funktion `isSearchTree()` hinzu, die prüft, ob ein Baum ein Suchbaum ist. Genau dann soll `true` zurückgeliefert werden.

*Hinweis: Sie können die Funktion `maxKey()` aus Aufgabe (a) verwenden. Sie können auch eine analoge Funktion `minKey()` verwenden, die den minimalen Schlüssel zurückliefert. Sie müssen `minKey()` nicht implementieren.*

```
boolean isSearchTree() {
```

```
    boolean leftOk = ( left == null ||  
                      left.isSearchTree() &&  
                      left.maxKey() < this.key );  
    boolean rightOk = ( right == null ||  
                       right.isSearchTree() &&  
                       right.mininKey() > this.key );  
    return leftOk && rightOk;
```

```
}
```

### Aufgabe 6 (9+5 = 14 Punkte)

- a) Gegeben eine Hash-Tabelle  $T$  ( $N = 9$ ), führen Sie ein Hashing mit quadratischer Sondierung durch. Fügen Sie nacheinander die Zahlen 11, 2, 21, 38, 3 und 29 ein. Notieren Sie nach jedem Einfügen den Status der Hash-Tabelle (siehe unten) sowie die Anzahl der benötigten Sondierungen.

Füge 11 ein.

0		
1		
2	11	0
3		
4		
5		
6		
7		
8		

$i$        $T[i]$       # Sondierungen

Füge 2 ein.

0		
1		
2	11	0
3	2	1
4		
5		
6		
7		
8		

Füge 21 ein.

0		
1		
2	11	0
3	2	1
4	21	1
5		
6		
7		
8		

Füge 38 ein.

0		
1		
2	11	0
3	2	1
4	21	1
5		
6	38	2
7		
8		

Füge 3 ein.

0		
1		
2	11	0
3	2	1
4	21	1
5		
6	38	2
7	3	2
8		

Füge 29 ein.

0	29	4
1	<del>29</del>	
2	11	0
3	2	1
4	21	1
5		
6	38	2
7	2	2
8		

$x$	11	2	21	38	3	29
$h(x)$	2	2	3	2	3	4

- b) Ist diese Aussage wahr: "Hashing mit quadratischer Sondierung benötigt niemals mehr Sondierungen als Hashing mit linearer Sondierung"? Falls ja, begründen Sie. Falls nein, geben Sie ein Gegenbeispiel.

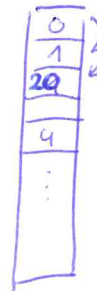
Gilt nicht. Gegenbeispiel:  $N=20$

Quadratisch



20 ~~ein~~fügen  
→ 3 Sondierungen

Linear



20 einfügen  
→ 2 Sondierungen

Matrikelnummer: \_\_\_\_\_

### Aufgabe 7 (4+5 = 9 Punkte)

Gegeben sei ein **Array a** aus  $n$  natürlichen Zahlen. Gesucht ist ein Algorithmus, der die **größte Primzahl** zurückgibt, die sich durch **Summierung von Zahlen des Arrays** bilden lässt (jede Zahl des Arrays darf hierbei maximal einmal vorkommen).

Beispiel: Für dieses Array...

3	9	6	2	4
---	---	---	---	---

... lautet die Lösung  $9 + 6 + 4 = 19$  (Primzahl!). Es lässt sich keine größere Primzahl bilden (z.B. ist  $9 + 6 + 4 + 2 = 21 = 3 \cdot 7$  nicht prim).

- a) Alice schlägt den folgenden Pseudo-Code zur Lösung des Problems vor (istPrim() prüft ob eine gegebene Zahl eine Primzahl ist). Welchem **Algorithmenmuster** entspricht Alice' Pseudo-Code? Geben Sie eine knappe Begründung.

```
1  summe = 0
2  for pos = 0, ..., n-1:
3      if istPrim(summe + a[pos]):
4          summe = summe + a[pos]
5  return summe
```

Greedy: Versuche mit möglichst wenig lokalen  
Ergänzungen zur Lösung zu kommen.  
↓  
(summe + a[pos])

- b) Ist Alice' Algorithmus korrekt? Begründen Sie.

Spezifikation: Größte Primzahl.

Algorithmus ist <sup>im Allgemeinen</sup> nicht korrekt.

Gegenbeispiel siehe oben: Ergebnis  $3 + 2 = 5$

Beste Lösung  $9 + 6 + 4 = 19$

⚡



Matrikelnummer: \_\_\_\_\_