[redacted]

2. Klausur Algorithmen

Gruppe A

Name: [redacted]

Matrikelnummer: [redacted]

Aufgabe	Erreichte Punktzahl
1 (6 Punkte)	5,5
2 (7 Punkte)	3,0
3 (5 Punkte)	4,5
4 (6 Punkte)	5,5
5 (7 Punkte)	5,5
6 (12 Punkte)	11,5
7 (6 Punkte)	6,0
8 (5 Punkte)	5,0
9 (6 Punkte)	4,0
Summe 60 Punkte	50,5

Bearbeitungszeit: 100 Minuten

Note: 1,7

Gesamtnote: 1,7 } Rip. 03.04.17

Bitte kreuzen Sie an, wenn Sie Ihren 3. Versuch durchführen:

☐ **3. Versuch**

Die Klausur ist bestanden, wenn mindestens 30 Punkte erreicht werden.

1. Aufgabe (6 Punkte) (5,5)

Betrachten Sie nachfolgenden Algorithmus in Pseudocode:

```
1:    public static void mysterious (int [] A) {
2:          int n = A.length;
3:          boolean b = true;
4:
5:          for (int i = 0; i < n-1 && b; i++) {
6:               b = false;
7:               for (int j = 1; j < n-i; j++) {
8:                    if (A[j] < A[j-1] {
9:                         b = true;
10:                        int tmp = A[j];
11:                        A[j] = A[j-1];
12:                        A[j-1] = tmp;
13:                   }
14:              }
15:         }
16:   }
```

Analysieren Sie die Laufzeit des Algorithmus.
Schätzen Sie dazu mit Hilfe der O-Notation die **Worst-Case-Laufzeit** und die **Best-Case-Laufzeit** des Algorithmus in Abhängigkeit von der Problemgröße (n – Länge des Arrays A).
Begründen Sie Ihre Aussage, indem Sie zu jeder Anweisung den Aufwand für die einfache Ausführung angeben sowie die Anzahl der Ausführungen im Worst-Case bzw. Best-Case.
Skizzieren Sie die Situation, in der der Worst-Case bzw. Best-Case eintritt.

z. 2: $O(1)$ ✓
3: $O(1)$ ✓
5: $O(n)$ ✓ → bestcase $O(1)$ ✓
6: $O(1)$ ✓
7: $O(n)$ ✓
8-12: $O(1)$ ✓

etwas unklar, wann Sie den Aufwand der einzelnen Ausführung u. wann die Anzahl der Durchläufe meinen -0,5

Die Laufzeit beträgt im worst case $O(n^2)$, da für jede Ausführung der äußeren for Schleife im Inneren das Array einmal* durchgegangen wird, wenn das Array umsortiert ist.
Wenn das Array bereits am Anfang sortiert ist ist die Laufzeit im best case $\Omega(n)$, da die inner for-Schleife n-mal durchläuft, die äußere nur 1-mal.

* nicht ganz einmal, sondern nur ca $\frac{n}{2}$ mal, der Einfachheit weggelassen

2. Aufgabe (7 Punkte) (3)

a) Geben Sie zu jeder der folgenden Aussagen an, ob diese wahr oder falsch sind. Für falsche Antworten wird ½ Punkt abgezogen:

- $(\log_2 n)^2 \in O(n)$ falsch ✓ f.
- $4n^2 + 3n \in \Theta(n^2)$ wahr ✓
- $n! \in O(3^n)$ wahr f.
- $\log n \cdot n^2 + \sqrt{n} \in \Omega(100n^2 \cdot \sqrt{n})$ wahr f.
- $0{,}001 \cdot 2^n \in O(n^8)$ falsch ✓
- $10 \in O(1)$ wahr ✓

$3 \cdot 0.75 - 2 \cdot 0{,}5 = 0{,}5$

b) Geben Sie eine möglichst einfache, scharfe Funktion $g(n)$ an, so dass $f(n) \in O(g(n))$ mit $f(n) = 8n^3 + n^2 \cdot \log n + 7\sqrt{n} \cdot n^3$

$g(n) = n^3 \cdot \sqrt{n}$ ✓

c) Geben Sie eine möglichst einfache, scharfe Funktion $h(n)$ an, so dass $h(n) \in \Omega(f(n))$ mit $f(n) = 8n^3 + n^2 \cdot \log n + 7\sqrt{n} \cdot n^3$

$h(n) = n^3 \sqrt{n}$ ✓

2,5

3. Aufgabe (5 Punkte) 4,5

Stellen Sie den Verlauf des Aufteilungsschrittes von Quicksort (ohne die nachfolgenden Rekursionen) für folgende Zahlenfolgen dar. Dabei soll als Pivotelement das Element am rechten Rand gewählt werden. Stellen Sie die Zahlenfolgen nach jedem Austauschschritt dar.

a) 5, 2, 17, 13, 6, 9, 4, 11

5, 2, 4, 13, 6, 9, 17 11 ✓

5, 2, 4, 9, 6, 13, 17 11 ✓

5, 2, 4, 9, 6 11, 17, 13 ✓

b) 4, 7, 6, 8, 3

3, 7, 6, 8, 4 ✓

c) 5, 1, 2, 6

kein Austausch ? -0,5

4. Aufgabe (6 Punkte) 5,5

Betrachten Sie nachfolgenden Ausschnitt einer Klasse für eine einfach verketteten Liste mit Integer-Werten.

```
public class SimpleIntLinkedList {
      class Node {
            Integer value;
            Node next;
      }

      private Node head;

      /**
       * ermittelt den letzten Wert in der Liste
       * @return der letzte Wert der Liste, falls diese nicht leer ist,
       *         null sonst
       */
      public Integer getLast() {
            // …
      }
      // weitere Methoden, die jedoch nicht zu verwenden sind
}
```

Das Attribut **head** verweist immer auf das erste Listenelement vom Typ **Node**. Falls die Liste leer ist, verweist **head** auf **null**. Ergänzen Sie die Implementierung der Methode **getLast**, die den Wert des letzten Listenelements ermittelt.

```
public Integer getLast() {
    Node current = head;
    [illegible]
    [illegible]
    if(head == null)
        { return null;
    }
    boolean found = false;
    while (!found) {
        if(current.next != null) {
            current = current.next;
        } else {
            found = true;
        }
    }
    return current.value;
}
```

- 0,5

55

5. Aufgabe (7 Punkte)
Tragen Sie in nachfolgenden Hashtabellen die Schlüssel ein, wenn als Hashverfahren die Divisionsrestmethode angewendet wird und als Sondierungsverfahren

- Lineares Sondieren
- Quadratisches Sondieren

Dabei sollen folgende Schlüssel in der angegebenen Reihenfolge eingefügt werden: 46, 17, 24, 23, 10, 13, 21. Geben Sie die Hashfunktion an sowie für jeden Schlüssel den Wert der Hashfunktion

Lineares Sondieren

0	1	2	3	4	5	6	7	8	9	10
21	23	24	47	13		17				10

(V)

Quadratisches Sondieren

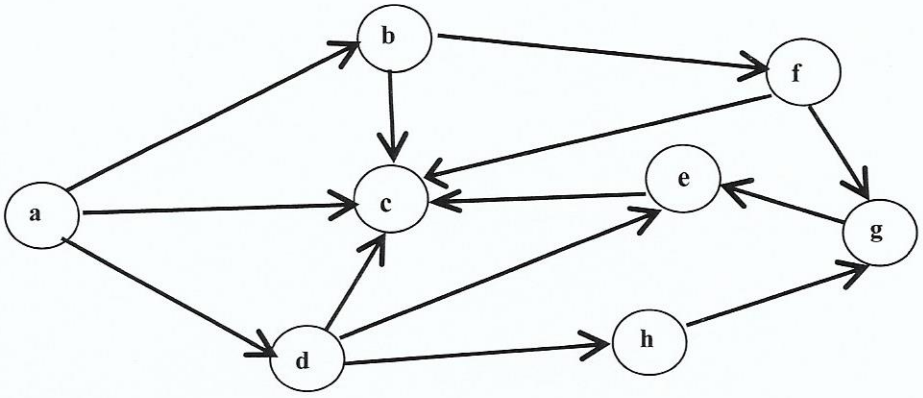
0	1	2	3	4	5	6	7	8	9	10
21	23	24	47			17			13	10

(V)

46
47 mod 11 = 3 f. −1 wg. Vereinfachung: −0,5
(weniger Kollisionen)
17 mod 11 = 6 ✓
24 mod 11 = 2 ✓
23 mod 11 = 1 ✓
10 mod 11 = 10 ✓
13 mod 11 = 2 ✓
21 mod 11 = 10 ✓

$h(k) = k \bmod 11$ −0,5

6. Aufgabe (12 Punkte) 11,5
Betrachten Sie nachfolgenden Graphen:



a) Führen Sie eine Breitensuche startend bei Knoten **a** durch. Bei Auswahlmöglichkeit zwischen mehreren Knoten ist immer der lexikographisch kleinste zu wählen.

Q: a, b, c, d, f, e, h, g

	a	b	c	d	e	f	g	h
pred	–	a	a	a	d	b	f	d
dist	0	1	1	1	2	2	3	2

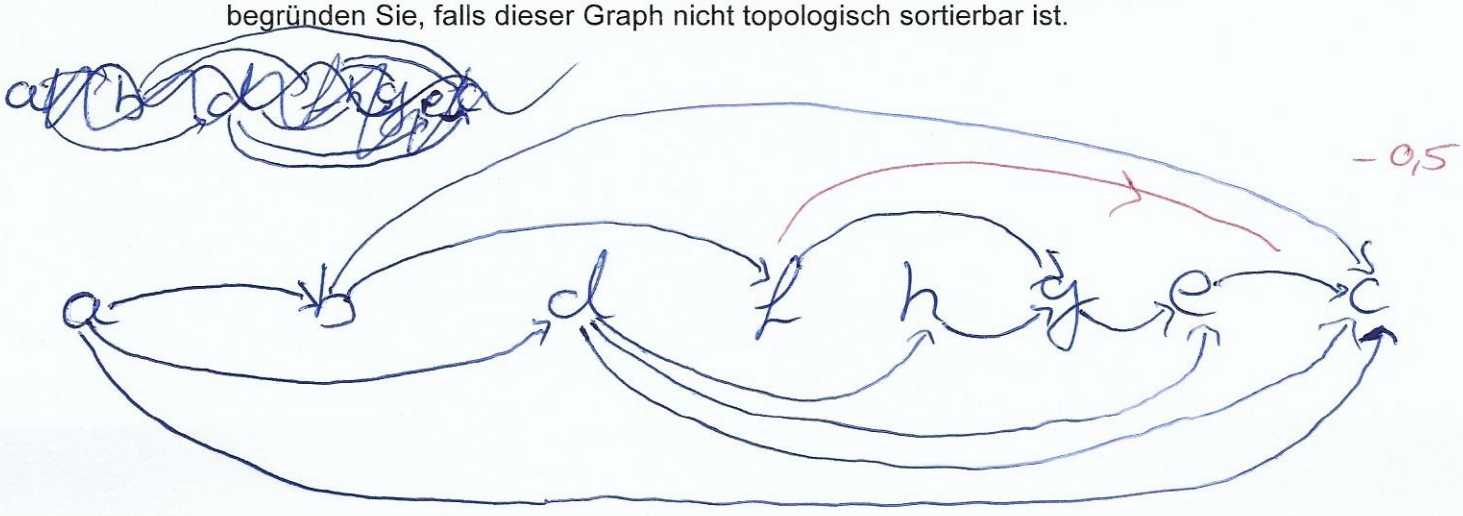
V

b) Führen Sie eine Tiefensuche startend mit Knoten **a** durch. Bei mehreren Auswahlmöglichkeiten eines Knotens wählen Sie immer den Knoten mit der lexikographisch kleinsten Beschriftung. Ermitteln Sie die **first**-, **last**- und **pred**-Werte und tragen Sie sie in nachfolgender Tabelle ein:

	a	b	c	d	e	f	g	h
first	1	2	3	12	7	5	6	13
last	16	11	4	15	8	10	9	14
pred	–	a	b	a	g	b	f	d

V

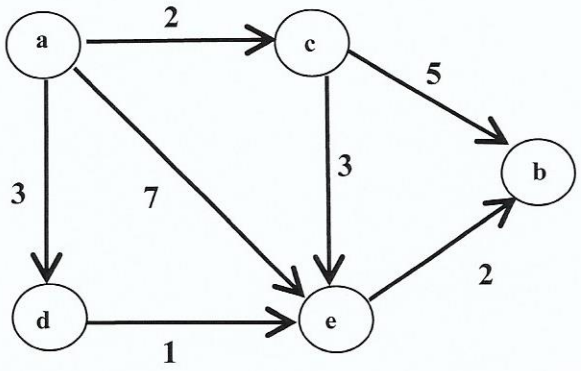
c) Geben Sie eine topologische Sortierung der Knoten des Graphen an bzw. begründen Sie, falls dieser Graph nicht topologisch sortierbar ist.



-0,5

7. Aufgabe (6 Punkte) (6)

Betrachten Sie nachfolgenden Graph:



Wenden Sie Dijkstra'a Algorithmus an zur Bestimmung der kürzesten Wege ausgehend von Knoten **a**. Stellen Sie dazu die Entwicklung der dist- und der pred-Werte dar nach jedem Schritt, in dem ein Knoten aus der Priority-Queue entfernt wurde und die Nachbarn dieses Knotens aktualisiert wurden:

Initialisierung

	a	b	c	d	e
pred	null	null	null	null	null
dist	0	∞	∞	∞	∞

1. Entfernung von Knoten ..a.. (hier bitte angeben, welcher Knoten aus der Queue entfernt wurde)

	a	b	c	d	e
pred	–	null	a	a	a
dist	0	∞	2	3	7

2. Entfernung von Knoten ..c.. (hier bitte angeben, welcher Knoten aus der Queue entfernt wurde)

	a	b	c	d	e
pred	–	c	a	a	c
dist	0	7	2	3	5

3. Entfernung von Knoten ..d.. (hier bitte angeben, welcher Knoten aus der Queue entfernt wurde)

	a	b	c	d	e
pred	–	c	a	a	d
dist	0	7	2	3	4

4. Entfernung von Knoten ..e.. (hier bitte angeben, welcher Knoten aus der Queue entfernt wurde)

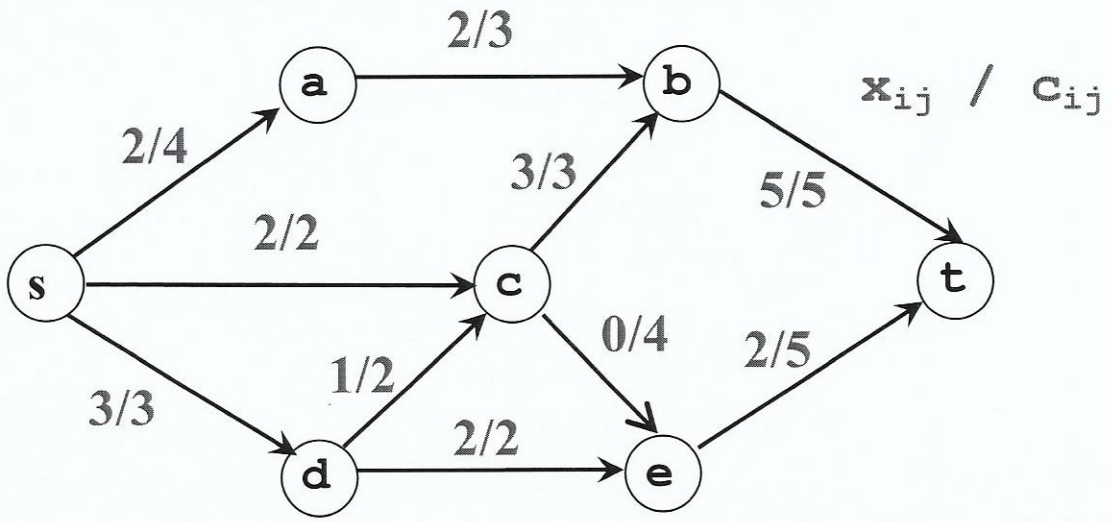
	a	b	c	d	e
pred	–	e	a	a	d
dist	0	6	2	3	4

5. Entfernung von Knoten ..b.. (hier bitte angeben, welcher Knoten aus der Queue entfernt wurde)

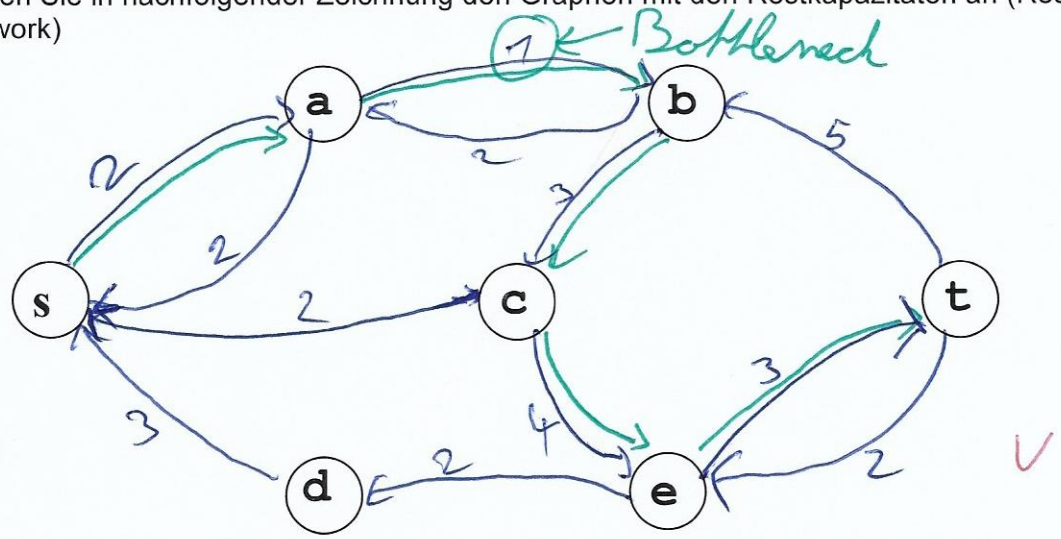
	a	b	c	d	e
pred	–	e	a	a	d
dist	0	6	2	3	4

8. Aufgabe (5 Punkte) 5

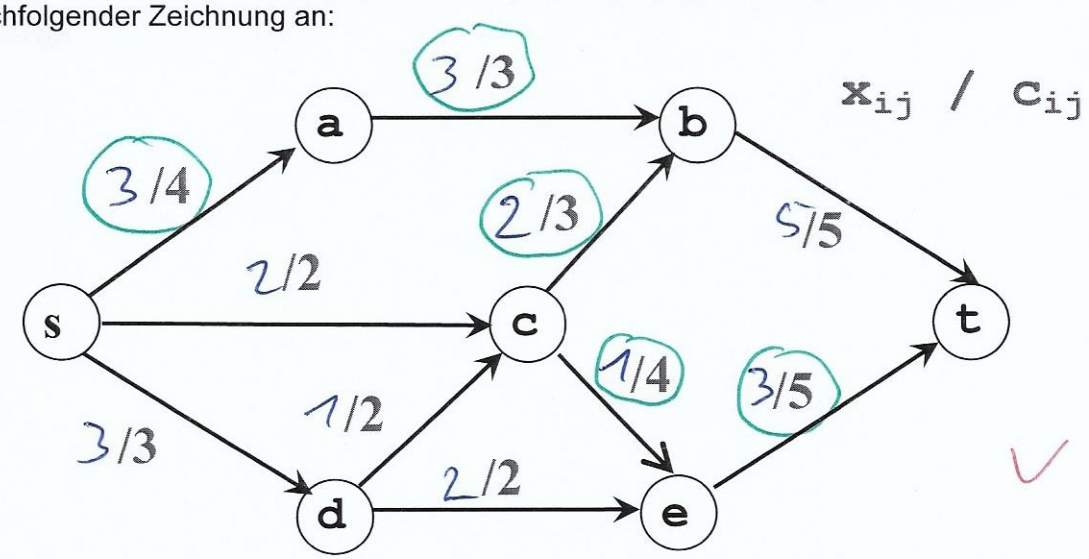
Betrachten Sie nachfolgendes s-t-Netzwerk mit einem zulässigen Fluss.



Geben Sie in nachfolgender Zeichnung den Graphen mit den Restkapazitäten an (Residual Network)



Ermitteln Sie in dem Graphen der Restkapazitäten einen erhöhenden Weg, zeichnen Sie ihn im Graph mit den Restkapazitäten ein und geben Sie den resultierenden Fluss in nachfolgender Zeichnung an:



Geben Sie den Wert des Flusses an!

8

9. Aufgabe (6 Punkte) (4)

Formulieren Sie die Bestimmung des Maximums in einem ganzzahligen Array als einen Divide-And-Conquer Algorithmus. Verwenden Sie dazu Pseudo-Code.

~~teile das Array in Teilbereich~~

~~maximum (A)~~
~~wenn Länge von A<=2~~
~~breich ab~~
~~teile das Array in zwei Teilarrays~~

Teile das Array in der Mitte, bis nur noch zweielementige Arrays bleiben
(*) {vergleiche die beiden Elemente jedes Arrays
füge die jeweils größeren zu neuen zweielementigen Arrays zusammen}
gib das übrige Element als maximum zurück
* wiederhole, bis nur noch ein Element übrig bleibt

Grundidee korrekt,
aber Rekursion kommt nicht
klar heraus!
nach 4