


Matrikelnummer:			
 <b>DHBW</b> Duale Hochschule Baden-Württemberg Stuttgart <b>ÜBUNGSKLAUSUR</b>		Fakultät	<b>Technik</b>
		Studiengang:	<b>Informatik</b>
		Jahrgang / Kurs :	<b>22 A/B/C/D/IN</b>
		Studienhalbjahr:	<b>2. Semester</b>
Datum:	<b>Juli 2023</b>	Bearbeitungszeit:	<b>90 Minuten</b>
Modul:	<b>T3INF1003.1</b>	Dozenten:	<b>Schulz, Dietzsch,</b>
Unit:	<b>Algorithmen</b>		<b>Kochanowski, Kötter</b>
Hilfsmittel:		<b>Ein handschriftliches DIN A4 Blatt beidseitig</b>	

Aufgabe	Hinweis zum Inhalt	erreichbar	erreicht
1	Sortieralgorithmen	9	
2	Heaps	14	
3	Komplexität	6	
4	Funktionen analysieren	8	
5	Pizza aufteilen	10	
6	Graphen	15	
7	Hashing und AVL	15	
Summe	77		

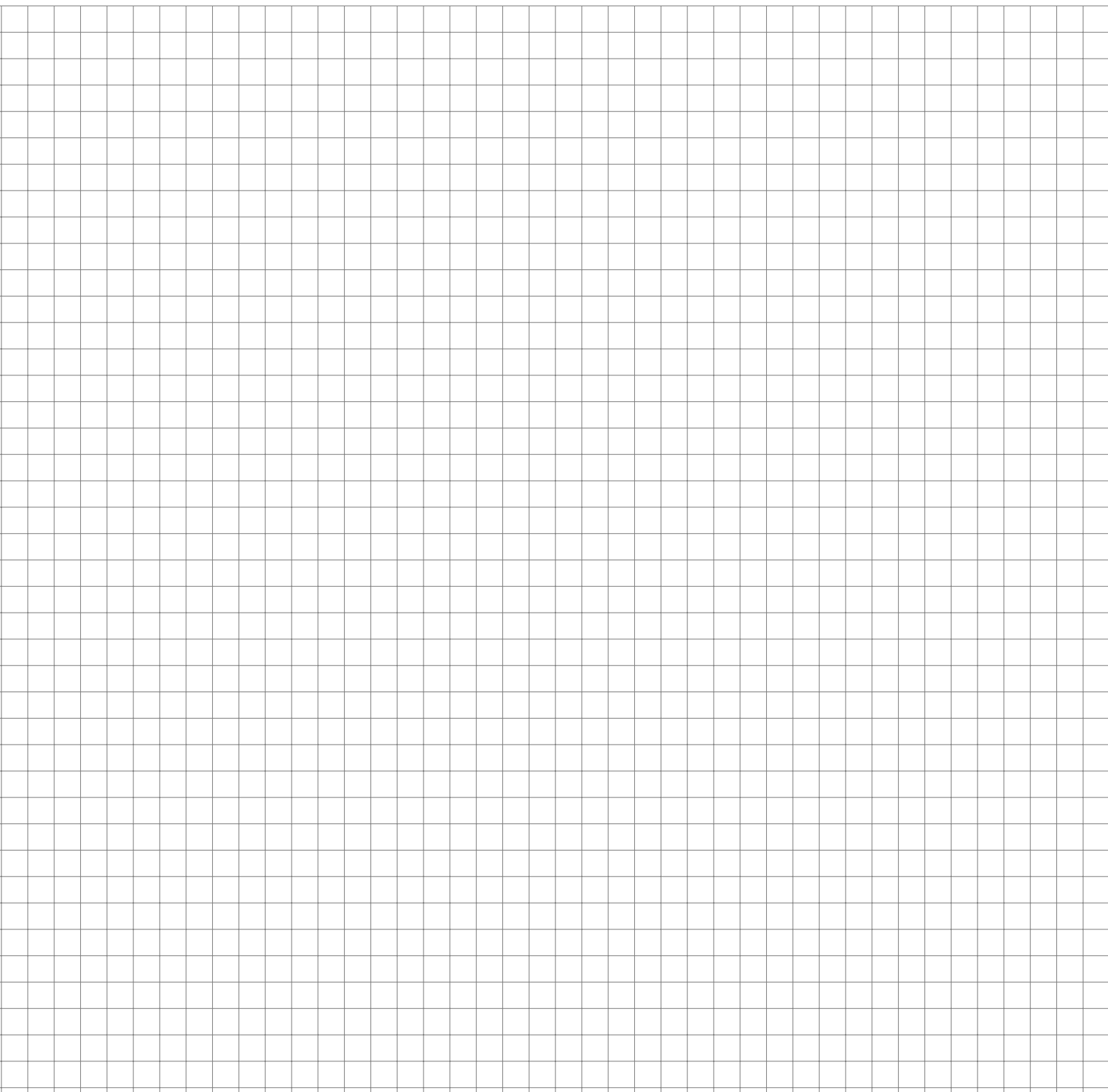
1. Sind Sie gesund und prüfungsfähig? Wer krank ist, kann jetzt die Prüfung noch verlassen. (Antrag auf Prüfungsrücktritt und ärztliches Attest!) Wer bleibt, hat damit erklärt, dass er sich gesund und prüfungsfähig fühlt.
2. Ein Täuschungsversuch und/oder die Benutzung nicht zugelassener Hilfsmittel führen zum Nichtbestehen der Klausur (5,0).
3. Wer den ordnungsgemäßen Ablauf der Prüfung stört oder sich nicht an die besonderen Regeln im Rahmen des Corona-Infektionsschutzes hält, kann von der weiteren Prüfung ausgeschlossen werden.
4. Auch außerhalb des Klausorraumes dürfen keine unerlaubten Hilfsmittel oder Unterlagen für die Klausur deponiert, bzw. verwendet werden.
5. Es dürfen während der Klausur keine Unterlagen und Informationen ausgetauscht oder weitergegeben werden.
6. Fragen an die Aufsicht sind nur hinsichtlich der Aufgabenstellung erlaubt.
7. Die Matrikel-Nr. wird auf das Deckblatt und alle weiteren Blätter der Klausur geschrieben. Der Name der/des Studierenden darf nicht auf der Klausur erscheinen.

**Aufgabe 1 (7+1+1 Punkte)**

Eine *lexikographische Ordnung*  $<_{lex}$  erweitert eine Ordnung auf Zeichen zu einer Ordnung auf Worten. Dabei werden Worte Zeichen für Zeichen verglichen. Wenn eines der Worte ein echtes Anfangswort eines anderen ist, dann ist das kürzere Wort kleiner. Wir verwenden die *umgekehrte* alphabetische Ordnung als Grundlage, d.h.  $a > b, b > c, \dots$ . Damit ist  $<_{lex}$  im wesentlichen die Umkehrung der normale Telefonbuchordnung. Wir zählen *Schlüsselvergleiche*, d.h. der Vergleich von *Jun* und *Jim* ist für uns ein einzelner Vergleich (und  $Jun < Jim$ ).

Betrachten Sie die Folge  $S = (Mo, Al, Tim, Li, Jo, Bo, Jil, Cal)$ .

- Sortieren Sie die Folge  $S$  aufsteigend gemäß der Ordnung  $<_{lex}$ . Verwenden Sie das in der Vorlesung gezeigte *Selection-Sort*-Verfahren (Sortieren durch Auswahl). Geben Sie den Zustand von  $S$  nach jedem Durchlauf der äußersten Schleife an.
- Wie viele *Vertauschungen* von Elementen der Folge  $S$  benötigt der Algorithmus? Vertauschungen eines Elementes mit sich selbst werden mitgezählt.
- Wie viele *Vergleiche* von Elementen der Folge  $S$  benötigt der Algorithmus?



**Aufgabe 2 (7+7 Punkte)**

a) Gegeben sei die Folge

$$S = (12, 6, 24, 3, 10, 16, 25, 5, 28, 2, 23, 29, 18)$$

Betrachten Sie die Folge als potentiellen Max-Heap in einem Array und führen Sie die Operation `heapify()` durch.

- Geben Sie eine graphische Darstellung des Baums nach jedem *bubble-down* eines Wertes an.
- Geben Sie den endgültigen Zustand als Array/Zahlenfolge an.

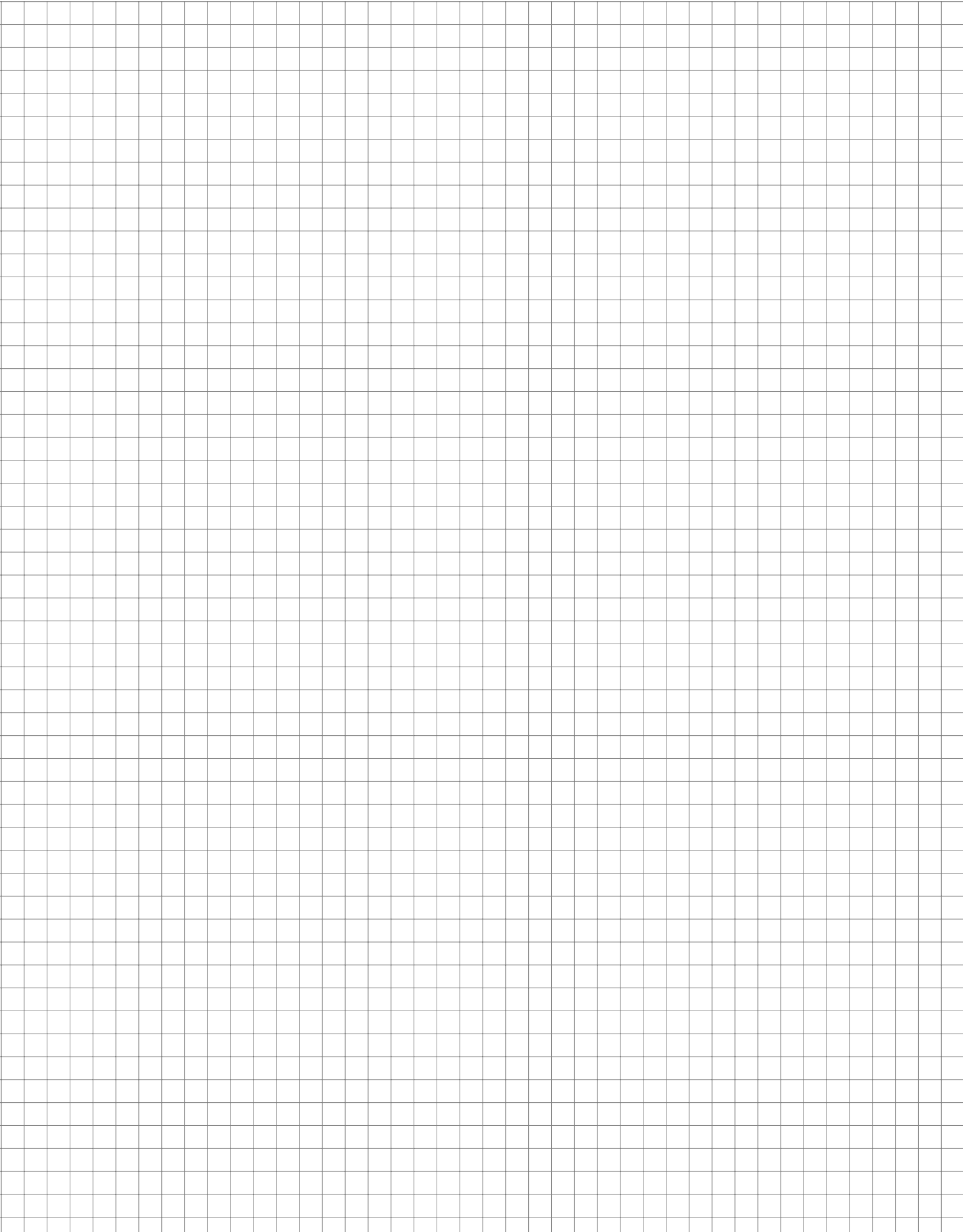
b) Betrachten Sie nun den Heap

$$T = (27, 26, 17, 22, 21, 9, 1, 14, 19, 13, 20, 7, 8)$$

in Array-Darstellung. Sortieren Sie  $T$  mit Heapsort. Brechen Sie das Verfahren ab, sobald die 3 größten Zahlen an ihrem Platz sind (und die Heap-Eigenschaft wieder hergestellt ist). Stellen Sie nach jedem Schritt den Teil des Arrays, der den Heap repräsentiert, als Baum dar. Stellen Sie am Ende (also nach 3 abgeschlossenen Schritten) den dann aktuellen Zustand des gesamten Arrays als Zahlenfolge da.



Fortsetzung

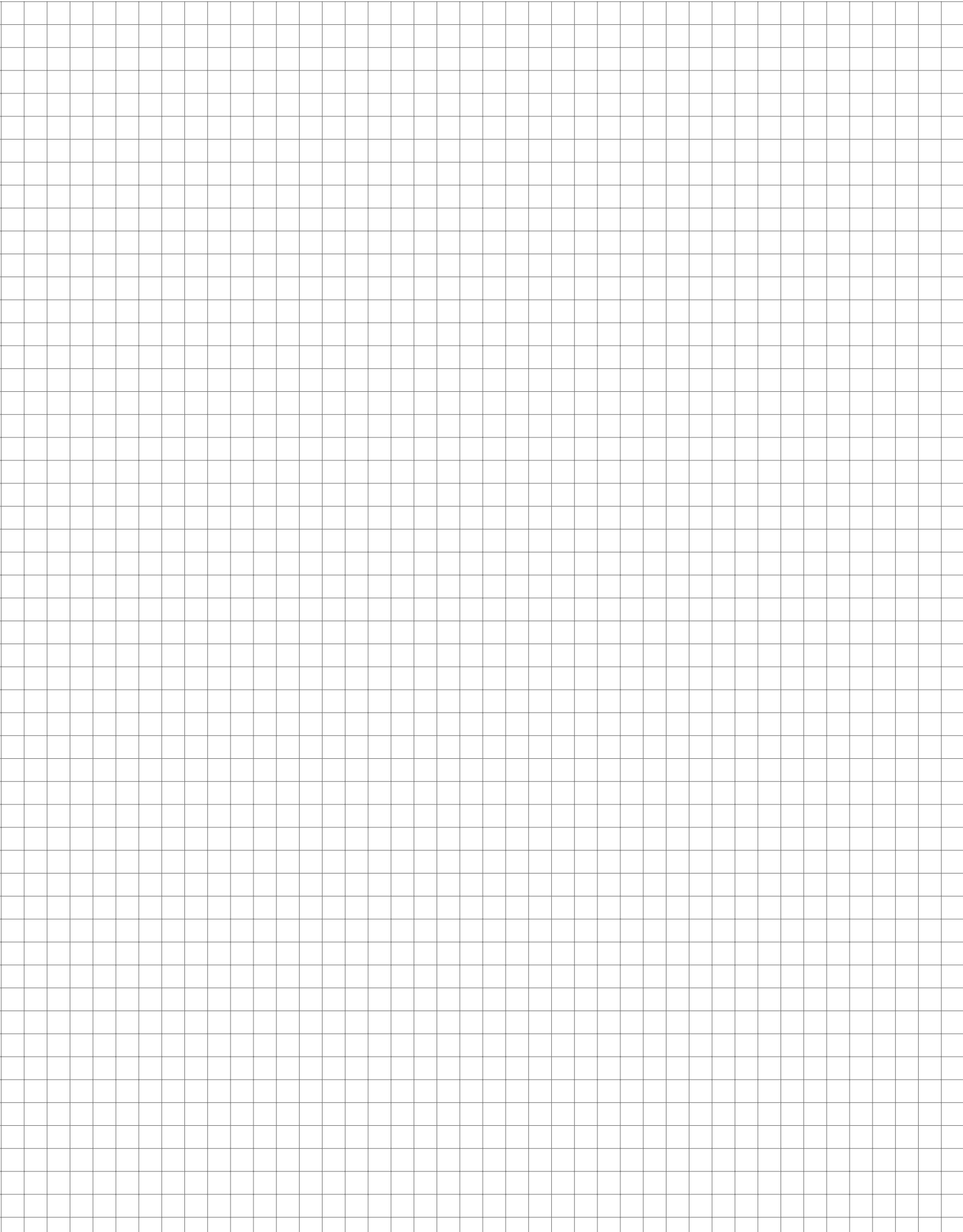


**Aufgabe 3 (2 + 2 + 2 Punkte)**

Für die folgenden Funktionen gilt  $x \in \mathbb{R}$

- a) Seien  $f : \mathbb{R} \rightarrow \mathbb{R}$  und  $g : \mathbb{R} \rightarrow \mathbb{R}$  definiert als die Funktionen  $f(x) = \sqrt[3]{x}$ ,  $g(x) = \log_2(x)$ .
- a1) Bestimmen Sie ein beliebiges  $c \in \mathbb{R}$  und  $k \in \mathbb{N}$  für die gilt  $c \cdot f(k) > g(k)$  und begründen Sie ihre Aussage.
- a2) Gilt allgemein  $g \in \mathcal{O}(f)$ ? Beweisen Sie ihre Aussage.
- b) Zeigen oder widerlegen Sie  $(x - x \cdot 2^{-x}) \in \mathcal{O}(x)$

Fortsetzung



**Aufgabe 4 (2+4+2 Punkte)**

- a) Betrachten Sie folgende Rekurrenzrelation:  $R(n) = 4R(\frac{n}{10}) + 3n(n^2 + 4)$ . Schätzen Sie (im Sinne  $\mathcal{O}$  bzw.  $\Theta$ ) die Funktion möglichst gut ab.
- b) Die unten stehende C-Funktion `hashrek()` berechnet aus einem Array-Ausschnitt von `n` Elementen ab dem Index `bot` einen einzelnen Integer-Wert.
- b1) Geben Sie eine Rekurrenz-Relation an, die die Laufzeit der Funktion in Abhängigkeit von `n` für den Normalfall (`n` relativ groß) beschreibt. Sie können konstante Aufwände/Faktoren mit Konstanten (z.B.  $c_1, c_2, c_3$ ) abschätzen.
- b2) Lösen Sie die Rekurrenzrelation und bestimmen Sie so die Laufzeitkomplexität der Funktion (im Sinne  $\mathcal{O}$  bzw.  $\Theta$ ).

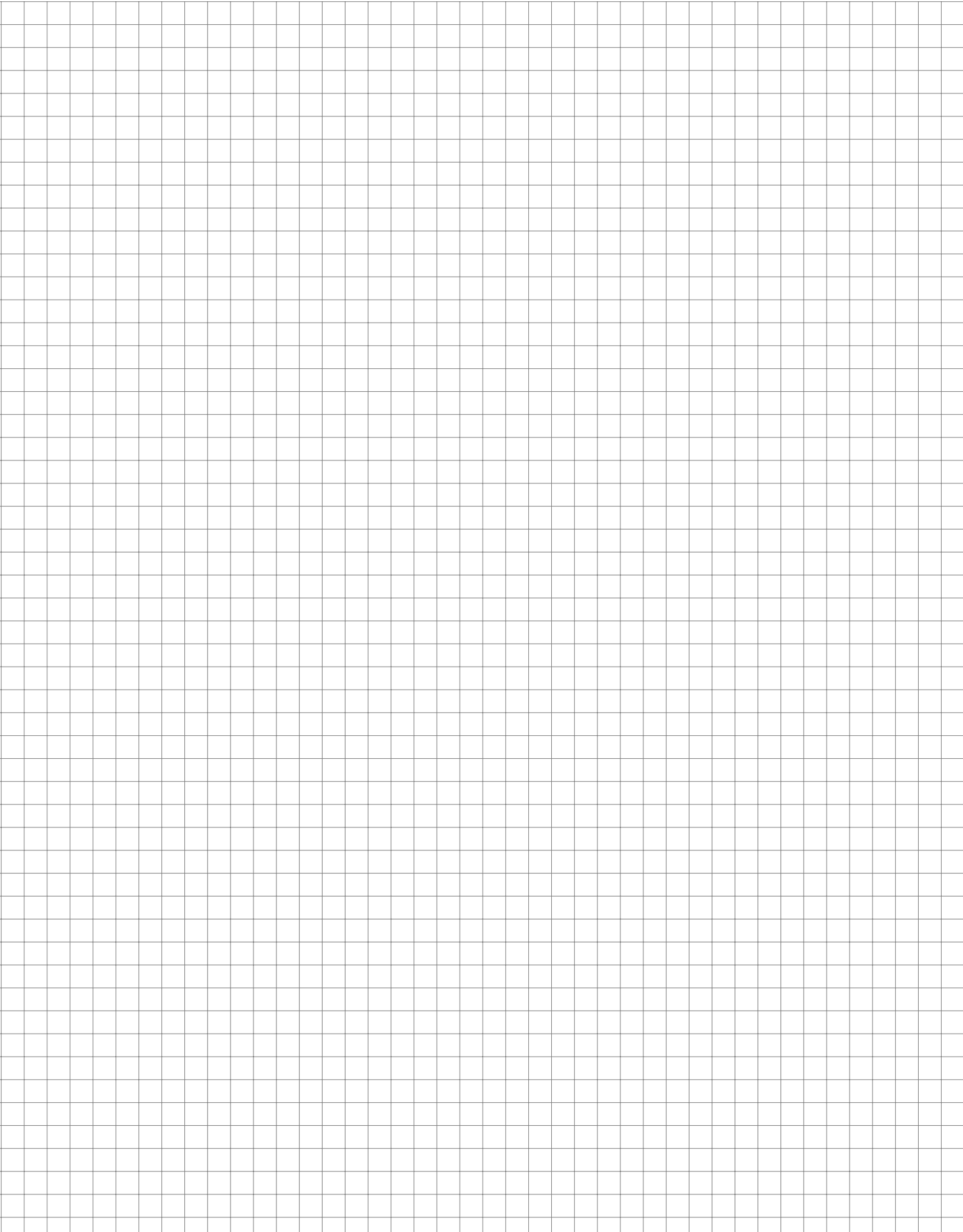
```

int hashrek(int array[], int bot, int n)
{
    int top, mid, hash1, hash2, res, i, j;

    if(n == 0)
    {
        return 0;
    }
    if(n == 1)
    {
        return array[bot];
    }
    top = bot+n;
    mid = (bot+top)/2;
    hash1 = hashrek(array, bot, mid-bot);
    hash2 = hashrek(array, mid, top-mid);
    res = 0;
    for(i=bot; i<bot+n; i++)
    {
        res++;
        res += hash1*(i+1)+hash2*i;
    }
    return res;
}

```

Fortsetzung

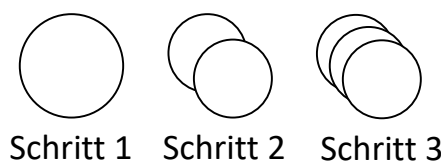




**Aufgabe 5 (3+3+3+1 Punkte)**

Algorithmen können auch im Alltag durchgeführt werden. Bei einer Party stellt sich Ihnen folgendes Problem: Sie möchten Pizza so aufteilen, dass jeder der  $g$  Gäste *genau* eines von  $s$  Stücken Pizza erhält. Dazu sollen Sie mehrere Algorithmen untersuchen.

Für DozentInnen und ProfessorInnen gibt es einen einfachen *Beispiel-Algorithmus*:



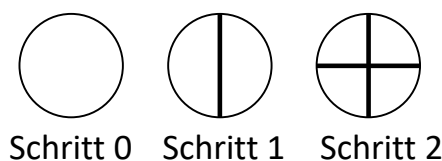
- Beginne mit 0 Pizzen.
- In jedem Schritt: Bestelle eine Pizza.
- Beende den Algorithmus, sobald es genau so viele Pizzastücke  $s$  wie Gäste  $g$  gibt. (Natürlich ist eine ganze Pizza ein Stück).

Als Studentin oder Student kommt dieser Algorithmus für Sie allerdings nicht in Frage. Sie können nur *eine* Pizza bestellen, die Sie für alle Gäste aufteilen müssen. Untersuchen Sie die folgenden drei Algorithmen und beantworten Sie jeweils die folgenden Fragen:

1. Berechnen Sie die Anzahl der Stücke  $s$  für die Schritte 1 bis 5. (Für den Dozenten-Algorithmus: 1, 2, 3, 4, 5)
  2. Geben Sie eine Formel für die Stücke  $s$  abhängig vom Schritt  $i$  an. (Im Dozenten-Beispiel:  $s(i) = i$ )
  3. Geben Sie eine Bedingung an, für welche Anzahl von Gästen  $g$  der Algorithmus aufgeht, so dass jeder Gast genau ein Stück bekommt und keine Stücke übrigbleiben. (Im Beispiel:  $g \in \mathbb{N}$ )
- a) Untersuchen Sie *Pizza-Algorithmus 1* und beantworten Sie die drei o.g. Fragen.

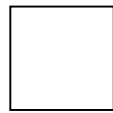


- Beginne mit einer runden Pizza.
  - In jedem Schritt: Schneide die komplette Pizza einmal durch den Mittelpunkt.
  - Beende den Algorithmus, sobald es gleich viele oder mehr Pizzastücke  $s$  als Gäste  $g$  gibt.
- b) Untersuchen Sie *Pizza-Algorithmus 2* und beantworten Sie die drei o.g. Fragen.

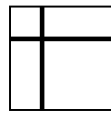


- Beginne mit einer runden Pizza.
- In jedem Schritt: Schneide alle bestehenden Stücke in der Mitte durch.
- Beende den Algorithmus, sobald es gleich viele oder mehr Pizzastücke  $s$  als Gäste  $g$  gibt.

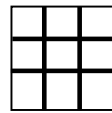
c) Untersuchen Sie *Pizza-Algorithmus 3* und beantworten Sie die drei o.g. Fragen.



Schritt 0



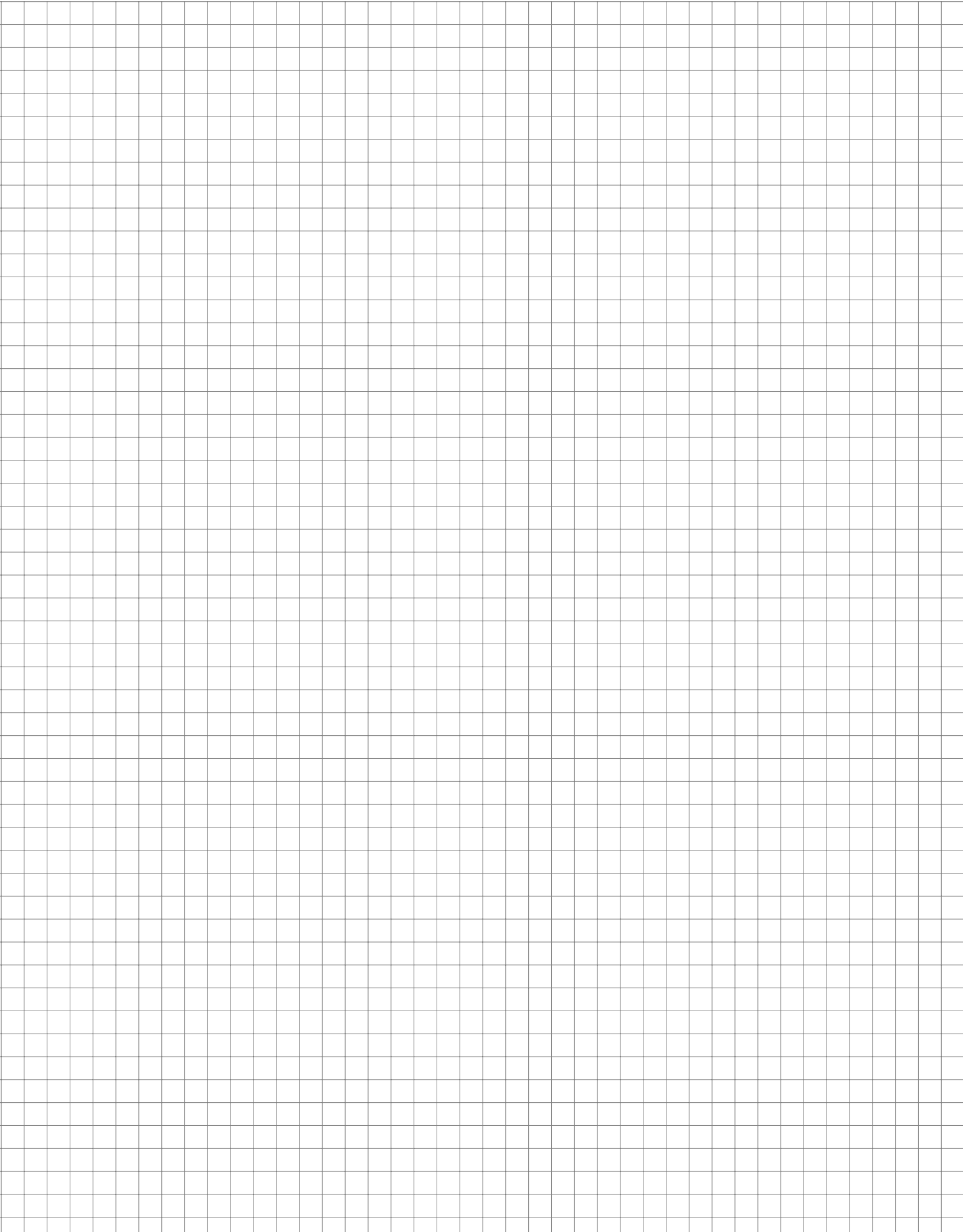
Schritt 1

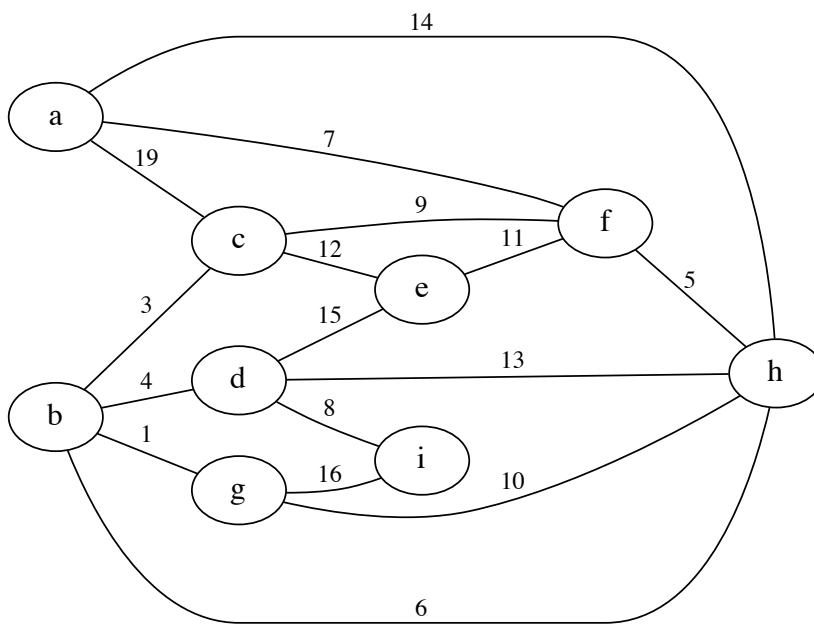


Schritt 2

- Beginne mit einer rechteckigen Party-Pizza.
  - In jedem Schritt: Mache je einen horizontalen und einen vertikalen Schnitt durch die komplette Pizza.
  - Beende den Algorithmus, sobald es gleich viele oder mehr Pizzastücke  $s$  als Gäste  $g$  gibt.
- d) Können Sie für jede mögliche Anzahl von Gästen  $g \in \mathbb{N}$  mittels mindestens einem der Algorithmen 1, 2 oder 3 (NICHT dem Beispiel-Algorithmus) eine Pizza so zerschneiden, dass jeder Gast genau ein Stück bekommt, ohne dass Stücke übrig bleiben? Begründen Sie ihre Antwort.

Fortsetzung



**Aufgabe 6 (5+2+1+7 Punkte)**a) Betrachten Sie den folgenden Graph  $G_a$ :Abbildung 1: Aufgabe 6a: Graph  $G_a$  (Prim, Ausgangsknoten **d**)

Bestimmen Sie für  $G_a$  ausgehend vom Knoten **d** einen minimalen Spannbaum mit Hilfe des Prim-Algorithmus. Sie können hierzu eine Liste der verwendeten Kanten angeben. Geben Sie die Reihenfolge an, in der Sie die Knoten dem Spannbaum hinzufügen. Wie hoch ist das Gesamtgewicht der Kanten des minimalen Spannbaums?

b) Sei  $G_b = (V, E)$  ein ungerichteter vollständig verbundener Graph (d.h. jeder Knoten ist mit jedem anderen Knoten direkt verbunden) mit  $|V| = n$  und einer Kantengewichtsfunktion  $e : E \rightarrow \mathbb{R}$ , die jeder Kante das Gewicht 3 zuordnet.

b1) Wie hoch ist das Gewicht eines minimalen Spannbaums (MST) von  $G_b$ ? Begründen Sie Ihr Ergebnis.

b2) Es seien  $a, b \in V$  und  $a \neq b$ . Wie lang ist die kürzeste Route von  $a$  nach  $b$ , die Dijkstra's Algorithmus findet?

c) Siehe nächste Seite!

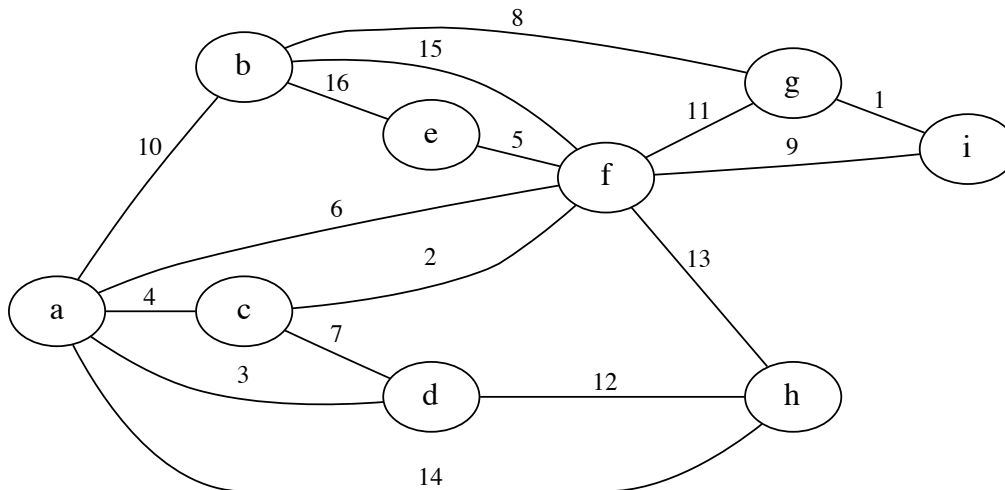


Abbildung 2: Aufgabe 6c: Graph  $G_c$  (Dijkstra, Ausgangsknoten **a**)

Verwenden Sie den Algorithmus von Dijkstra, um die minimale Entfernung aller Knoten in  $G_c$  vom Knoten **a** zu bestimmen. Geben Sie die Knoten in der Reihenfolge an, in der sie im Algorithmus durchlaufen werden (d.h. in der ihre endgültige Entfernung bestimmt wird). Es kann durchaus vorkommen, dass mehr als eine Reihenfolge möglich ist - in diesem Fall wählen Sie bitte den Knoten mit dem alphabetisch kleineren Namen zuerst (wenn z.B.  $a$  und  $g$  möglich wären, expandieren Sie  $a$  zuerst).

[illegible][illegible]

**Aufgabe 7 (3+4+4+2+2 Punkte)**

a) Gegeben sei die Schlüsselfolge

$$S = (21, 8, 2, 17, 13, 9)$$

und die Hash-Funktion  $h(x) = (x + 3) \bmod 13$ .

- a1) Geben Sie zunächst das Ergebnis der Hash-Funktion für jeden Schlüsselwert in  $S$  an. Verwenden Sie dafür eine Tabelle wie unten gezeigt.
- a2) Fügen Sie nun die Elemente von  $S$  in der angegebenen Reihenfolge in die bereits z.T. gefüllte Hash-Tabelle der Größe 13 (Indizes 0-12) ein, die unten abgebildet ist. Verwenden Sie *linear Probing* mit Abstand 1, um Konflikte zu lösen. Geben Sie den vollständigen Inhalt der Hash-Tabelle nach jeder Einfüge-Operation an. Wie viele Kollisionen treten auf? Bedenken Sie, dass es für jeden Wert mehrere Kollisionen geben kann.

b) s. nächste Seite

c) s. nächste Seite

**Schlüsseltabelle (a1)**

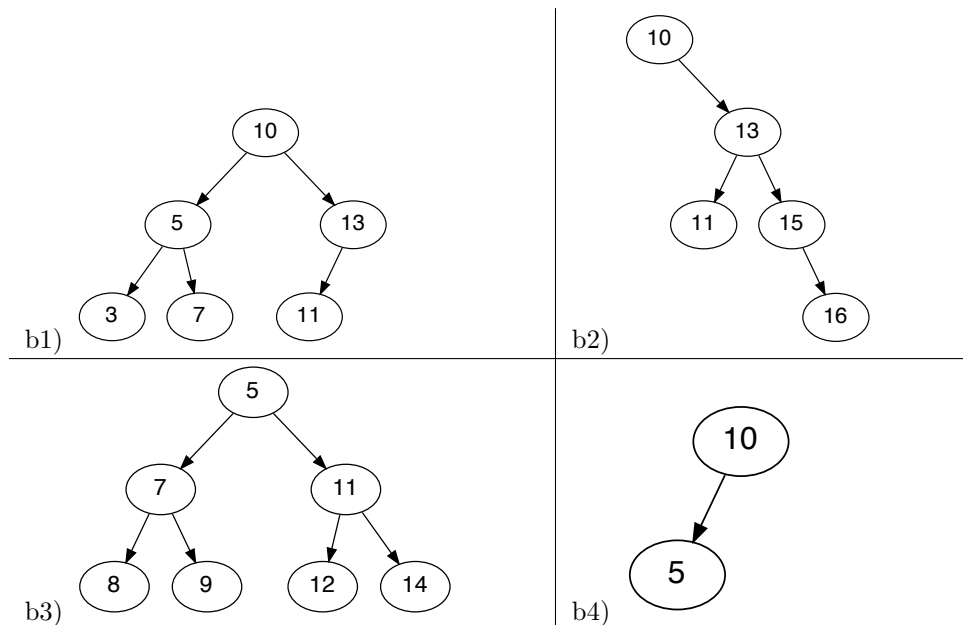
Key $x$	$h(x)$
21	
8	
2	
17	
13	
9	

**Linear Probing (a2)**

	-	21	8	2	17	13	9
0							
1							
2	12						
3	26						
4	27						
5							
6	29						
7							
8							
9							
10							
11							
12							

Anzahl der Kollisionen:

- b) Betrachten Sie die folgenden Binärbäume. Entscheiden Sie für jeden Baum, ob er ein Max-Heap, Min-Heap, Binärer Suchbaum oder AVL-Baum ist. Als richtig gewertet werden Bäume, bei denen alle 4 Eigenschaften korrekt (mit je einem Kreuz für *ja* und einem Kreis für *nein*) markiert sind.



Baum	b1	b2	b3	b4
Max-Heap				
Min-Heap				
Binärer Suchbaum				
AVL-Baum				

- c) Betrachten Sie den AVL-Baum  $B$ .

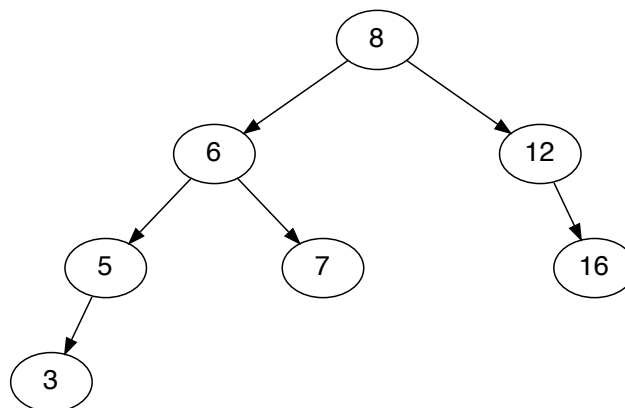
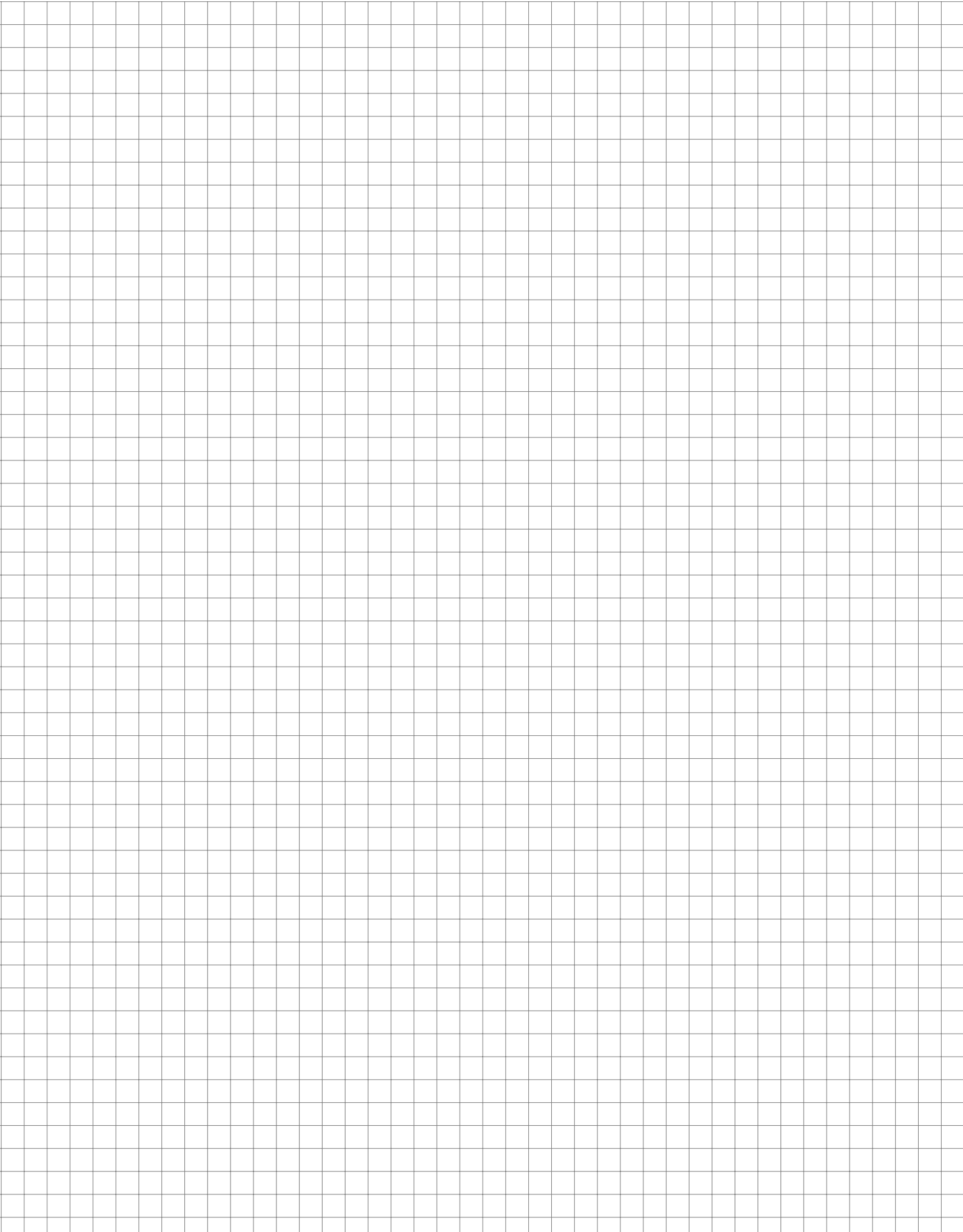


Abbildung 3: AVL-Baum  $B$

- c1) Fügen Sie den Schlüssel **4** in den Baum  $B$  ein und stellen Sie die AVL-Eigenschaft gemäß dem AVL-Algorithmus wieder her. Zeichnen Sie das Ergebnis.
- c2) Löschen Sie den Schlüssel **12** aus dem Baum  $B$  (nicht aus dem Ergebnis von Aufgabenteil c1) und stellen Sie die AVL-Eigenschaft gemäß dem AVL-Algorithmus wieder her. Zeichnen Sie das Ergebnis.

Fortsetzung





Fortsetzung

– Ende der Klausur –