— Vom Prüfungsteilnehmer vollständig auszufüllen —

Winter / Sommer – Semester ____/____

Studiengang: _____

Prüfungsfach: _____

Matrikelnummer: | | | | | | | | |

Platznummer	Raumnummer Turnhalle	
_____	_____	

		Vom Prüfer/in bzw. 1. Korrektor/in auszufüllen	
Prüfungstag:		(Note)	
Arbeitszeit:			Unterschrift des Erstprüfer/s/in
Beginn der Bearbeitung:		Vom 2. Korrektor/in auszufüllen	
Ende der Berarbeitung:		(Note)	
			Unterschrift des Zweitprüfer/s/in

Erlaubte Hilfsmittel: _____

_____

_____

Bemerkungen(z.B.Unterbrechung der Bearbeitung, wird vom Aufsichtsführenden ausgefüllt)

von _____ **Uhr bis** _____ **Uhr**

von _____ **Uhr bis** _____ **Uhr**

von _____ **Uhr bis** _____ **Uhr**

_____

_____

_____

Hinweis für die Korrektur von schriftlichen Arbeiten

Erst– und Zweitkorrektor sind auf der Prüfungsarbeit zu vermerken (§13 Abs. 4 Satz 4 RaPO)

Algorithmen und Datenstrukturen

in den Studiengängen Medieninformatik und Wirtschaftsinformatik

---

Hinweise:

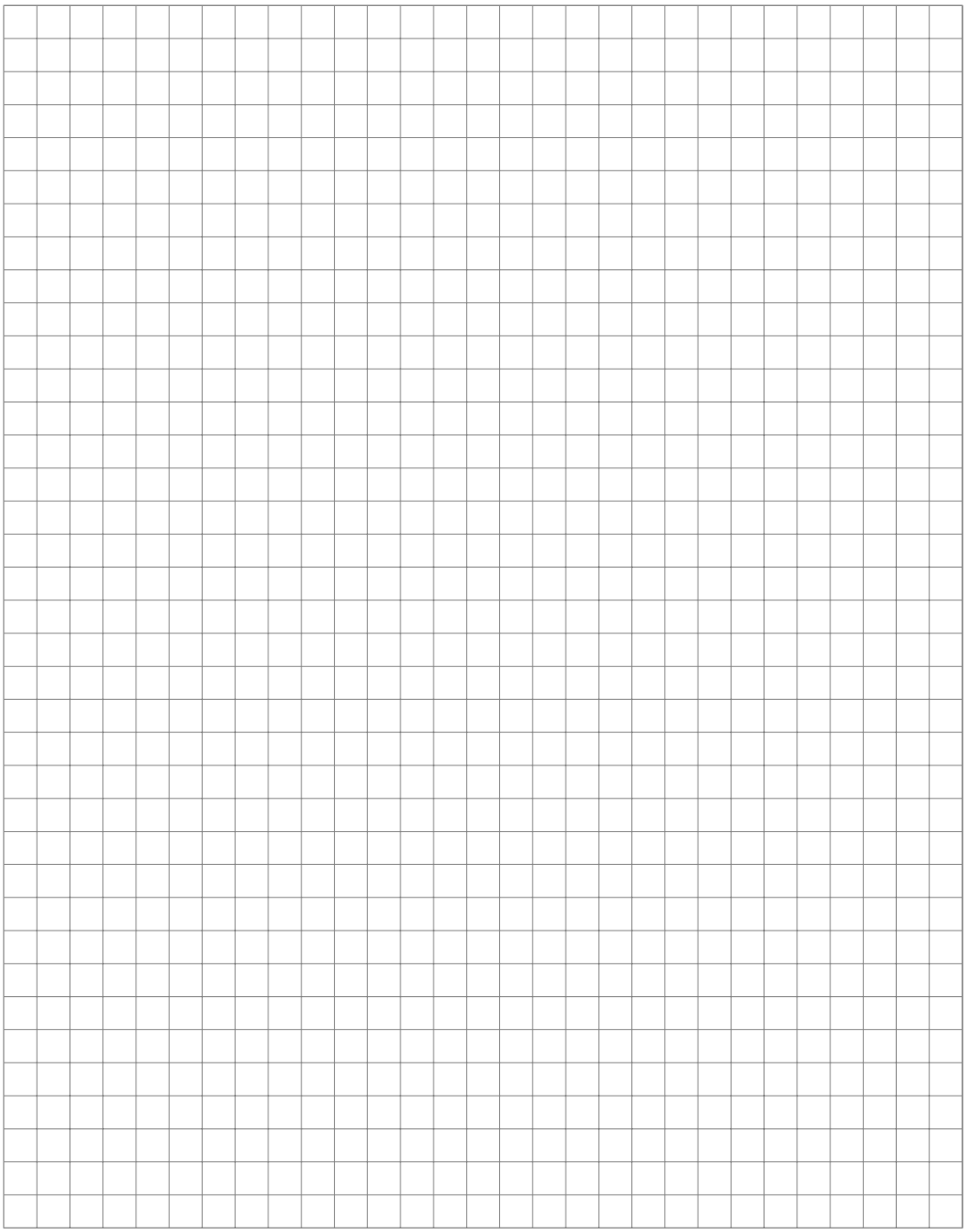
- Bitte geben Sie nur eine Lösung pro Antwort ab.
- Die Angabe umfasst insgesamt 16 Seiten und 5 Aufgaben.
- Die Bearbeitungszeit beträgt 90 Minuten.
- Für das Bestehen der Klausur sind mindestens 21 Punkte notwendig.
- **Bitte öffnen Sie die Heftklammern nicht!**
- **Vergessen Sie nicht die Matrikelnummer auf der Klausur anzugeben!**

Matrikelnummer: _____

***Aufgabe 1*:** **12 Punkte**

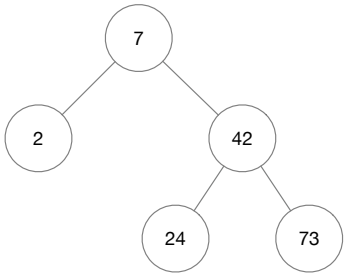
Schreiben Sie eine Funktion `merge` die zwei sortierte Arrays mit `int`-Werten ≥ 0 als Parameter bekommt und ein sortiertes Array mit den Werten aus den Arrays an den Aufrufer zurückgibt. Dabei sollen mehrfach vorkommende Werte nur *einmal* in das Ergebnisarray übernommen werden.

Für die Arrays `{1,2,3,4,5}` und `{2,4,6,8,10,12}` enthält das Ergebnisarray die Werte `{1,2,3,4,5,6,8,10,12}`. Für die beiden Arrays `{1,2,2,3,3,3,4,4,4,4}` und `{0,5,10}` ist das Ergebnis `{0,1,2,3,4,5,10}`



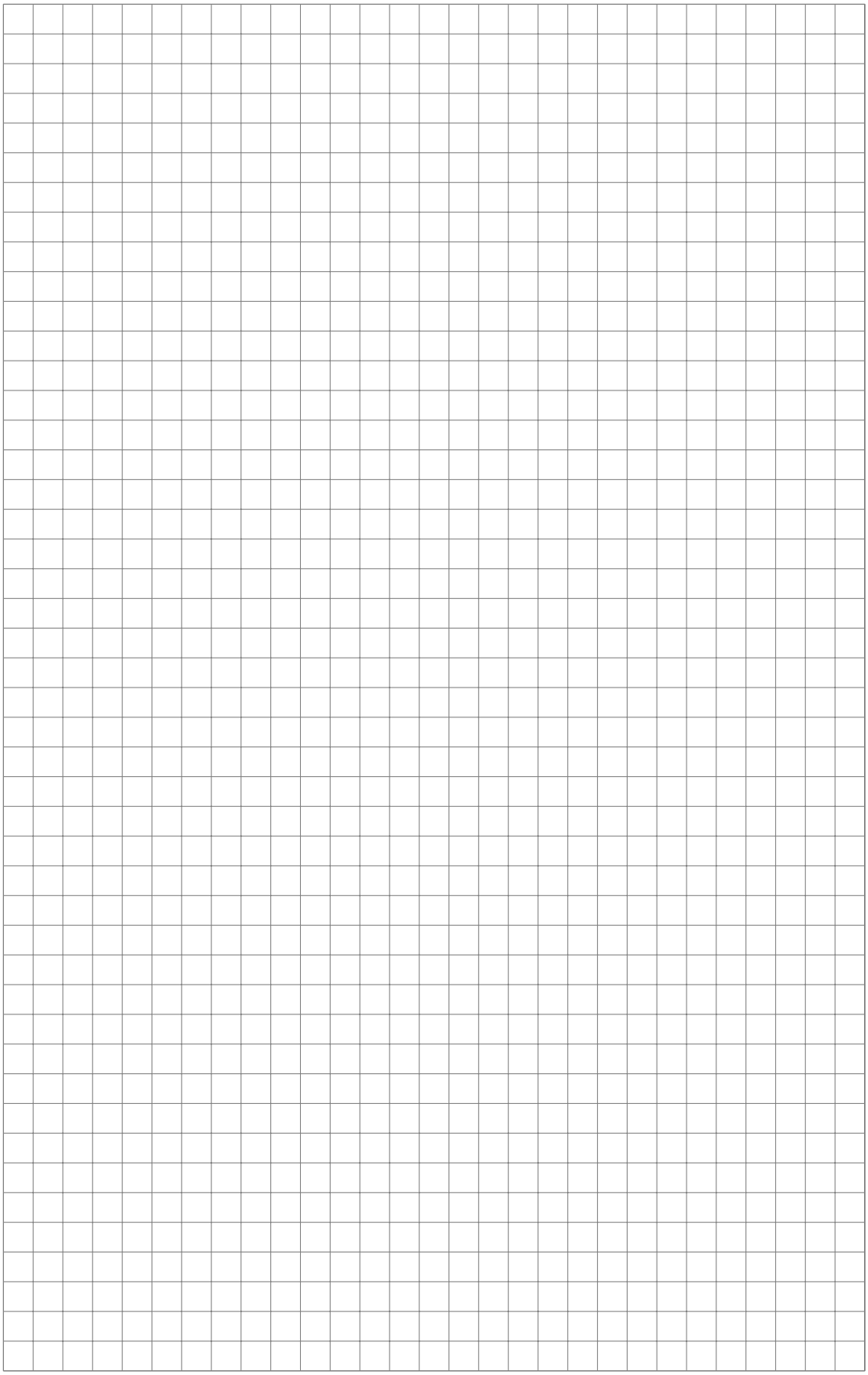
***Aufgabe 2*:** **12 Punkte**

Gegeben ist folgender AVL-Baum:

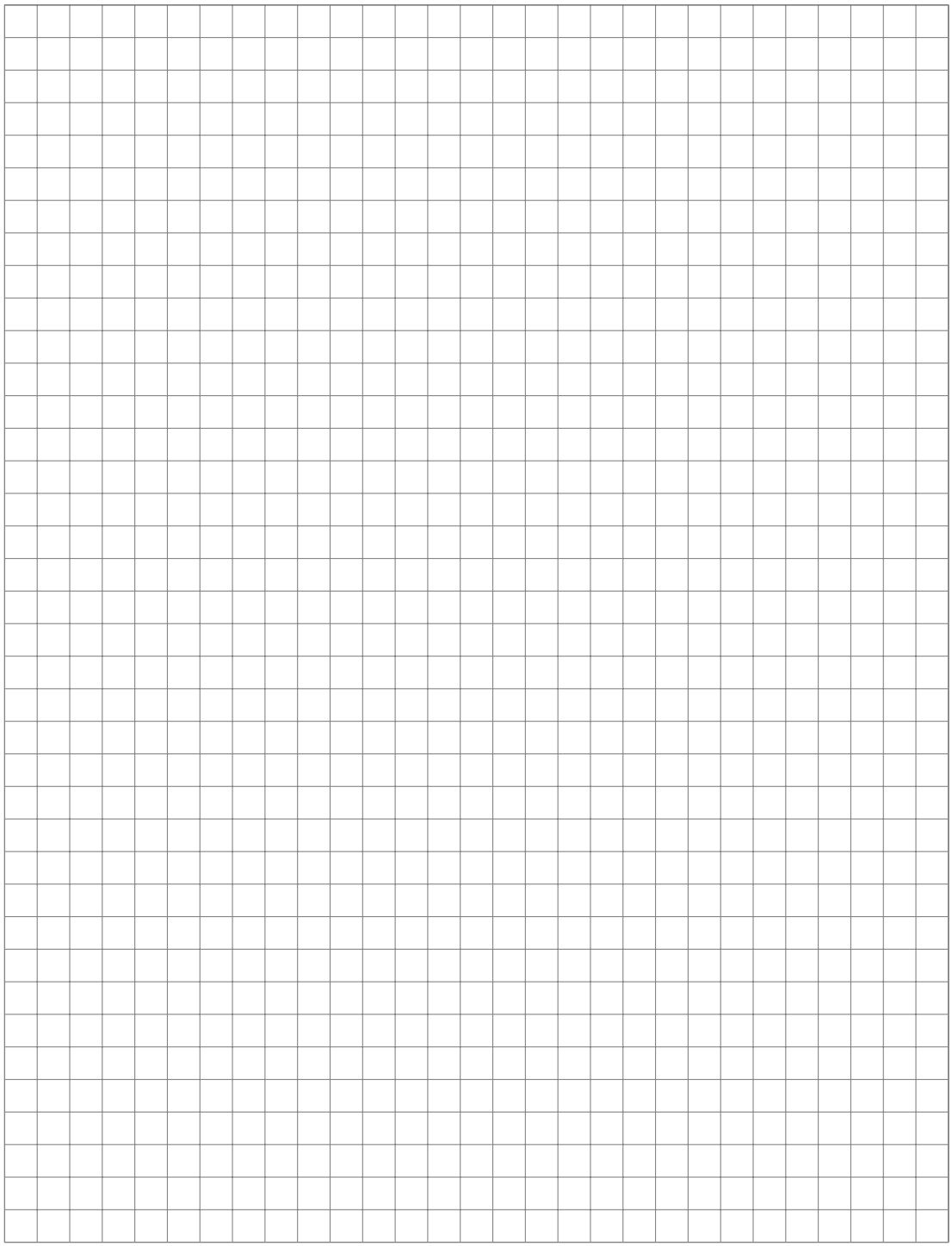


Teilaufgabe a:

In diesen AVL-Baum soll der Wert 15 so eingetragen werden, dass wieder ein AVL-Baum entsteht. Falls dazu Rotationen notwendig sind, müssen die notwenigen Teilschritte durch entsprechende Grafiken klar zu erkennen sein.



Teilaufgabe b:
Aus dem entstandenen AVL-Baum soll zuerst der Wert 73 und dann der Wert 42 gelöscht werden. Falls dazu Rotationen notwendig sind, müssen die notwenigen Teilschritte durch entsprechende Grafiken klar zu erkennen sein.
Falls Sie Teilaufgabe a) nicht lösen konnten (und auch nur dann), können Sie als Ausgangbaum ausnahmsweise auch den AVL-Baum aus der Aufgabenstellung und aus diesem zuerst den Werte 24 und dann den Wert 2 löschen.



***Aufgabe 3*:** **6 Punkte**

Schreiben Sie eine Funktion

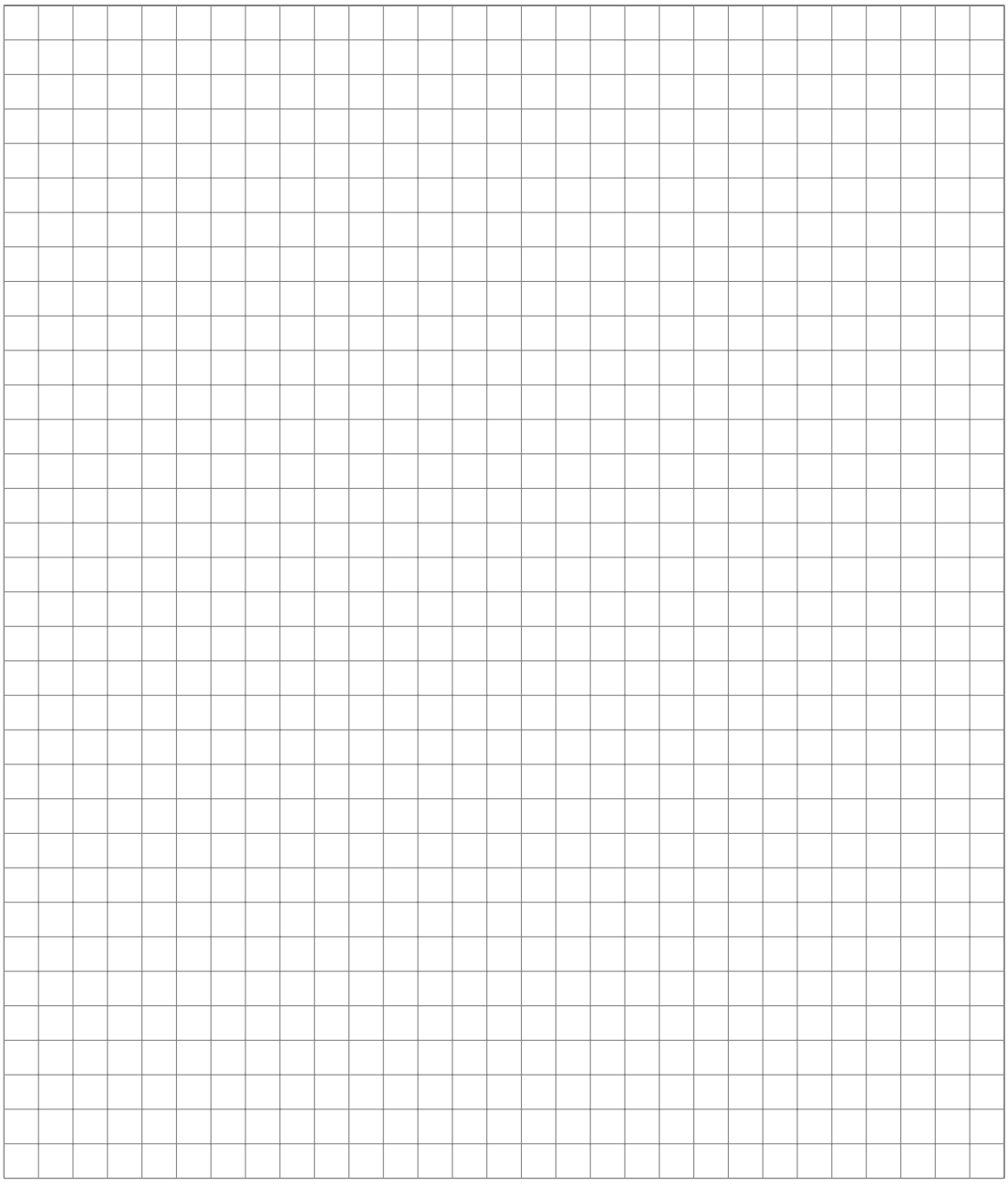
```
boolean hasOne(int aValue)
```

die feststellt ob der übergebene `int` Wert `aValue` mindestens einmal die Ziffer 1 einhält. In diesem Fall soll die Funktion `1` zurückgeben, andernfalls `0`.

Der Wert `aVaule` darf dabei nicht in einen String umgewandelt werden.

Beispiele:

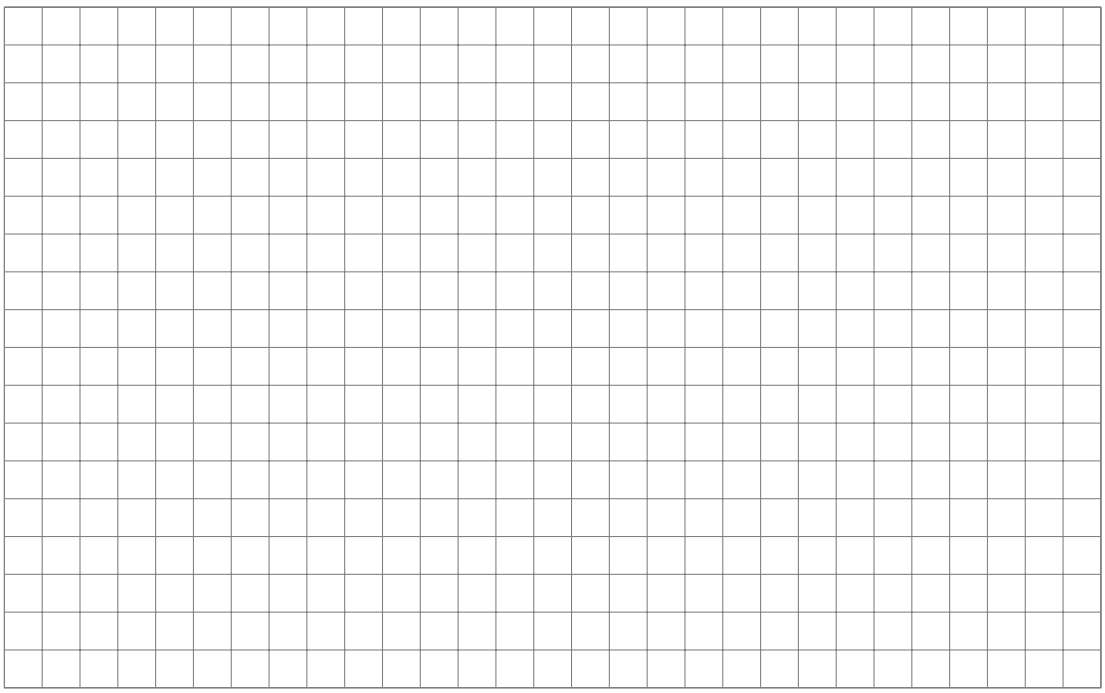
- `hasOne(10)` ⇒ `1`
- `hasOne(-11)` ⇒ `1`
- `hasOne(22)` ⇒ `0`

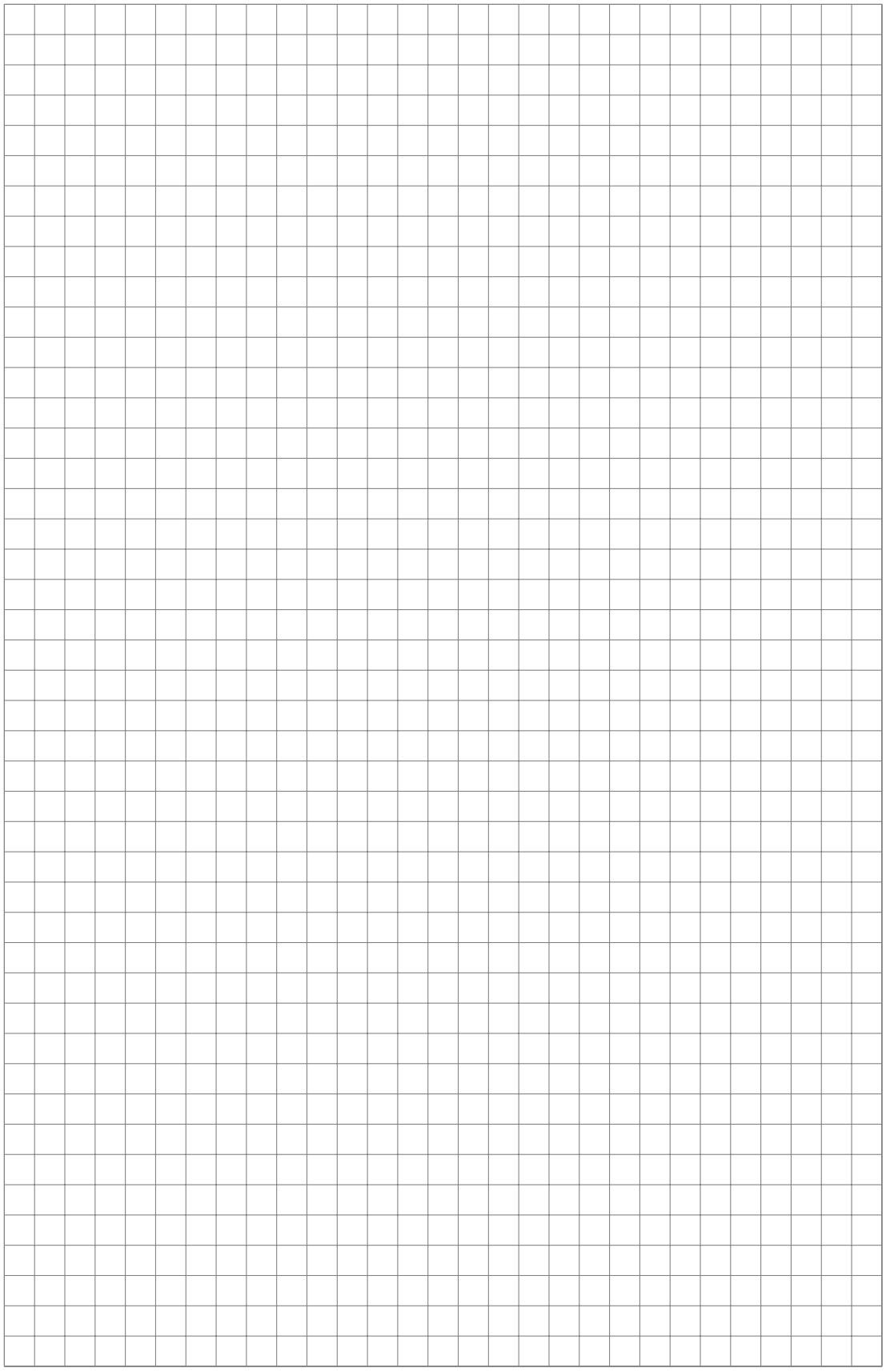


***Aufgabe 4*:** **8 Punkte**

Passen Sie folgende Implementierung von Bucket-Sort so an, dass Arrays mit beliebigen `int`-Werten sortiert werden können. Geben Sie dazu an, an welchen Stellen neuer Code eingefügt werden muss und welche Zeilen zu ändern sind.

```
1  public void bucketSort(int array[])
2  {
3      int max = array[0];
4      for (int i=1; i<array.length; i++)
5      {
6          if (array[i] > max) max = array[i];
7      }
8
9      int buckets[] = new int[max+1];
10     for (int i=0; i<array.length; i += 1)
11     {
12         buckets[array[i]] = buckets[array[i]] + 1;
13     }
14
15     int idx=0;
16     for (int b=0; b<buckets.length; b += 1)
17     {
18         for (int i=0; i<buckets[b]; i += 1)
19         {
20             array[idx] = b;
21             idx += 1;
22         }
23     }
24 }
```





***Aufgabe 5*:** **4 Punkte**

Folgende Methode liefert für ein nichtleeres `int`-Array ein Array als Ergebnis in dem der kleinste und der größte Wert des Array gespeichert sind.

```
1  public int [] getMinMax(int array[])
2  {
3      int max = array[0];
4      int min = array[0];
5      for (int i=1; i<array.length; i++)
6      {
7          if (array[i] > max) max = array[i];
8          if (array[i] < min) min = array[i];
9      }
10
11     int res[] = {min, max};
12     return res;
13 }
```

Den überwiegende Teil der Laufzeit verbraucht dabei die `for`-Schleife für die 2∗array.length Vergleiche der Array-Elemente.

Geben Sie eine Implementierung an, die sicherstellt, dass für beliebige Arrays nur rund $1.5 * \text{array.length}$ Vergleiche der Array-Elemente notwendig sind.