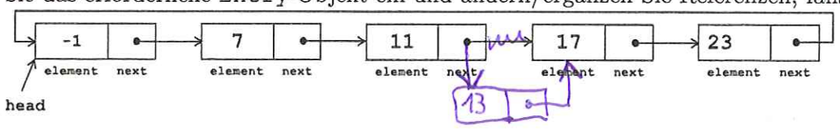
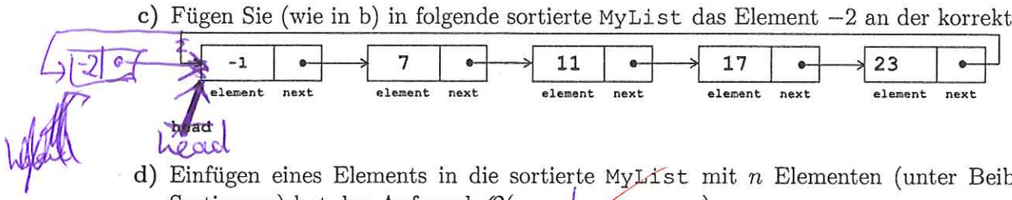
b) Fügen Sie in folgende sortierte `MyList` das Element 13 an der korrekten Stelle ein. Zeichnen Sie das erforderliche `Entry`-Objekt ein und ändern/ergänzen Sie Referenzen, falls notwendig.



2

c) Fügen Sie (wie in b) in folgende sortierte `MyList` das Element -2 an der korrekten Stelle ein.



½

d) Einfügen eines Elements in die sortierte `MyList` mit n Elementen (unter Beibehaltung der Sortierung) hat den Aufwand: $\mathcal{O}($ nlogn $)$

0

e) Implementieren Sie das Einfügen eines neuen Elements in die sortierte Liste:

```
public void addSorted(Entry toAdd) { // keeps list sorted
    Entry cur = head.next;
    Entry drag = head;
```

if (cur

0

1

Aufgabe 4 (ADT)

(9 Punkte)

2½

Der ADT `AA` (AssociativeArray) entspricht einer Reihung vom Typ `T` mit **beliebiger** Länge und hat folgende Eigenschaften: Der Konstruktor `create` instanziiert die Reihung. Mit den Methoden `get` und `set` kann man die Elemente nach ganzzahligen (nicht notwendigerweise positiven) Indizes auslesen bzw. mit einem Wert != `null` aktualisieren. `delete` löscht (falls vorhanden) die Belegung eines Indizes. Lesende Zugriffe auf unbelegte/gelöschte Indizes liefern `null`. Die Anzahl der belegten Indizes kann mit `size` abgefragt werden. Ergänzen Sie die Operationen und Axiome.

adt *AA*
sorts *AA, T, int*
ops

f create : ~~int~~ → AA
f get : T → int
f set : T → int
f delete : T → null
f size : AA → int

axs

get (r,s) = r f
set (r,s) = s f

0

Prof. Dr. Michael Philippsen
Department Informatik, FAU

13. Januar 2014

Mini-Klausur: Algorithmen und Datenstrukturen

Name, Vorname: Pistler, Lukas Matrikelnummer: 21801328

Bewertung (Punkteverteilung unter Vorbehalt - wird vom Prüfer ausgefüllt!)

Aufgabe	1	X 2 X	3	4	Σ
Maximal	4 / 6	2 / 19	2½ / 11	0 / 9	8.5 /45

Aufgabe 1 (Wissensfragen)

(6 Punkte)

Bei den folgenden Teilaufgaben werden richtige Antworten positiv, falsche entsprechend negativ gewertet. Unbearbeitete Teilaufgaben geben keinen Abzug. Die Aufgabe wird insgesamt mit mindestens 0 Punkten bewertet.

a) Die folgenden zwei Code-Zeilen verhalten sich in einem Java-Programm immer äquivalent und sind daher austauschbar:

```
assert (x > 4711) : "Expected 0.815";
if (x > 4711) throw new java.lang.AssertionError("Expected 0.815");
```

☐ wahr ☒ falsch

1

b) Welche der folgenden Aussagen zur Rekursion sind richtig:

☐ Die Fibonacci-Funktion aus der Vorlesung ist verschachtelt rekursiv.
☒ Zwei Funktionen, die sich gegenseitig aufrufen, nennt man verschränkt rekursiv. 1
☒ Ein fehlender Basisfall kann zur Endlosrekursion führen. 1
☐ Die lineare Rekursion ist ein Spezialfall der Endrekursion.

c) Wird der ADT `Set<T>` (zur Darstellung von Mengen) mit einer doppelt verketteten Liste ohne Sortierung implementiert, dann haben das Einfügen eines Wertes in eine bestehende Menge mit n Werten und das anschließende Löschen eines anderen Wertes aus dieser Menge zusammen eine Laufzeitkomplexität von:

☐ $\mathcal{O}(1)$ ☐ $\mathcal{O}(log_2 n)$ ☒ $\mathcal{O}(n)$ ☐ $\mathcal{O}(n^2)$

1

d) Eine Streutabelle, die Kollisionen mit Hilfe einer verketteten Liste auflöst, kann (ausreichend Arbeitsspeicher vorausgesetzt) beliebig viele Elemente speichern.

☐ richtig ☒ falsch

-1

e) In Java gibt es generische Klassen, die bei der Instanziierung durch einen generischen Klassenparameter konkretisiert werden. Als Klassenparameter können dabei sowohl Klassen als auch Interfaces verwendet werden.

☒ richtig ☐ falsch

1

4

Aufgabe 2 (Magische Quadrate) (19 Punkte)

Ein magisches Quadrat der Kantenlänge n ist eine quadratische Anordnung der Zahlen $1, 2, \ldots, n^2$, sodass die Summe der Zahlen jeder Zeile, jeder Spalte und jeder Diagonalen gleich ist (nämlich $\Sigma = \frac{n^3+n}{2}$) und jede der Zahlen $1, 2, \ldots, n^2$ genau einmal darin vorkommt. Das Beispiel rechts zeigt ein solches Quadrat mit $n = 3$ und $\Sigma = 15$.

2	7	6	→ 15
9	5	1	→ 15
4	3	8	→ 15
↙ 15	↓ 15	↓ 15	↘ 15

a) Ergänzen Sie die Methode `isSolved`. Sie soll überprüfen, ob `sq` eine gültige Lösung enthält, also dass die Summe jeder Spalte, Zeile und Diagonale gleich $\frac{n^3+n}{2}$ sind.

```
boolean isSolved(int[][] sq) {
    int n = sq.length;
    int checkSum = ((n * n * n) + n) / 2;
    int rowSum = 0, colSum = 0, diag1Sum = 0, diag2Sum = 0;
```

```
for (int i=1; i<n; i++) {
?
if (rowSum==checkSum && colSum==checkSum && Gebiet
if (rowSum==colSum && colSum==diag1Sum && diag1Sum==diag2Sum
    && diag2Sum==checkSum)
if (checkSum==15)
return true;
```

b) Ergänzen Sie die Methode `solve`. Sie soll für ein anfangs leeres Quadrat mittels Backtracking und unter Verwendung von `isSolved` eine Belegung finden, sodass es schließlich ein magisches Quadrat darstellt. Die Methode `solve` wird dabei im folgenden Kontext aufgerufen:

```
MagicSquare(int n) {
    int[][] sq = new int[n][n];
    if (solve(sq, 0, new boolean[n * n]))
        print(sq);
}
```

Verwenden Sie das Array `used`, um zu speichern, welche der Zahlen $1, 2, \ldots, n^2$ Sie beim Backtracking schon verwendet haben. Genau dann, wenn es keine Lösung gibt, soll `solve` `false` zurückgeben; falls mehrere Lösungen existieren, genügt eine beliebige.


```
boolean solve(int[][] sq, int pos, boolean[] used) {
    int n = sq.length;
    int col = pos % n;
    int row = (pos - col) / n;
    // Basisfall
```

```
if(isSolved(sq)) {
return true;
}
```

```
// Rekursion
```

```
return false;
}
```

Aufgabe 3 (Listen) (11 Punkte)

Gegeben sei der Rumpf einer Implementierung einer aufsteigend sortierten, einfach verketteten Liste von `int`-Werten mit Wächterelement.

a) Implementieren Sie die geeignete Initialisierung der Liste im Konstruktor – beachten Sie hierbei das Wächterelement.

```
class Entry {
    int element;
    Entry next;
    Entry(int element, Entry next) {
        this.element = element;
        this.next = next;
    }
}
```

```
public class MyList {
    // Waechterelement
    private Entry head = new Entry(-1, null);
    MyList() {
```

```
}
```