Stand: September 2023

Teil 2: Erläuterungen und Lösungsvorschläge

Die Illustrierenden Prüfungsaufgaben (Teil 1: Beispielaufgaben, Teil 2: Erläuterungen und Lösungsvorschläge) dienen der einmaligen exemplarischen Veranschaulichung von Struktur, Anspruch und Niveau der Abiturprüfung auf grundlegendem bzw. erhöhtem Anforderungsniveau im neunjährigen Gymnasium in Bayern.

Informatik

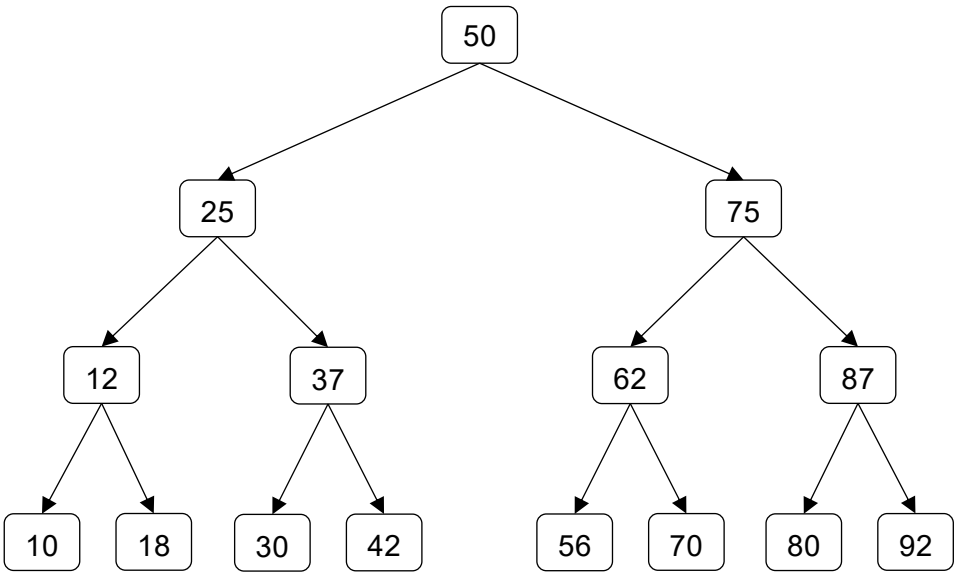
grundlegendes Anforderungsniveau

Die Bewertung der erbrachten Prüfungsleistungen hat sich für jede Aufgabe nach der jeweils am rechten Rand der Aufgabenstellung vermerkten, maximal erreichbaren Anzahl von Bewertungseinheiten (BE) zu richten.

Der Erwartungshorizont stellt für jede Teilaufgabe eine mögliche Lösung dar; als objektorientierte Programmiersprachen werden Java und Python verwendet. Nicht dargestellte korrekte Lösungen sind als gleichwertig zu akzeptieren.

Die von einem Prüfling insgesamt erreichten Bewertungseinheiten werden gemäß folgender Tabelle in Notenpunkte umgesetzt:

mind. zu erreichender Anteil an den insgesamt zu erreichenden Bewertungseinheiten (in %)	Notenpunkte	Notenstufe
95	15	+1
90	14	1
85	13	1−
80	12	+2
75	11	2
70	10	2−
65	9	+3
60	8	3
55	7	3−
50	6	+4
45	5	4
40	4	4−
33	3	+5
27	2	5
20	1	5−
0	0	6

Nr.		BE
1a	Möglicher balancierter Binärbaum:  Die Anzahl der maximal nötigen Vergleiche bei einer Suche entspricht der Anzahl der Ebenen des Suchbaums. Ein balancierter Baum hat bei fester Knotenzahl die kleinstmögliche Anzahl von Ebenen und ist deshalb für die Suche optimal. Mögliche Einfügereihenfolge für obigen Baum: 50, 25, 75, 12, 37, 62, 87, 10, 18, 30, 42, 56, 70, 80, 92	4

b Mögliche Lösung in Python: 9

In der Klasse BINBAUM:

```
def istVorhanden(self, skipassnr: int) -> bool:
    return self.wurzel.istVorhanden(skipassnr)

def sortiertAusgeben(self, skipassart: str):
    self.wurzel.sortiertAusgeben(skipassart)
```

In der Klasse BAUMELEMENT:

```
@abc.abstractmethod
def istVorhanden(self, skipassnr: int) -> bool:
    pass

@abc.abstractmethod
def sortiertAusgeben(self, skipassnr: str):
    pass
```

In der Klasse KNOTEN:

```
def istVorhanden(self, skipassnr: int) -> bool:
    if self.inhalt.nummerGeben() == skipassnr:
        return True
    else:
        if self.inhalt.nummerGeben() < skipassnr:
            return self.nachfolgerRe.istVorhanden(skipassnr)
        else:
            return self.nachfolgerLi.istVorhanden(skipassnr)

def sortiertAusgeben(self, skipassart: str):
    self.nachfolgerLi.sortiertAusgeben(skipassart)
    if self.inhalt.artGeben() == skipassart:
        self.inhalt.datenAusgeben()
    self.nachfolgerRe.sortiertAusgeben(skipassart)
```

In der Klasse ABSCHLUSS:

```
def istVorhanden(self, skipassnr: int) -> bool:
    return False

def sortiertAusgeben(self, skipassart: str):
    pass
```

Mögliche Lösung in Java:

In der Klasse BINBAUM:

```
boolean istVorhanden(int skipassnr) {
    return wurzel.istVorhanden(skipassnr);
}

void sortiertAusgeben(char skipassart) {
    wurzel.sortiertAusgeben(skipassart);
}
```

In der Klasse BAUMELEMENT:

```
abstract boolean istVorhanden(int skipassnr);

abstract void sortiertAusgeben(char skipassart);
```

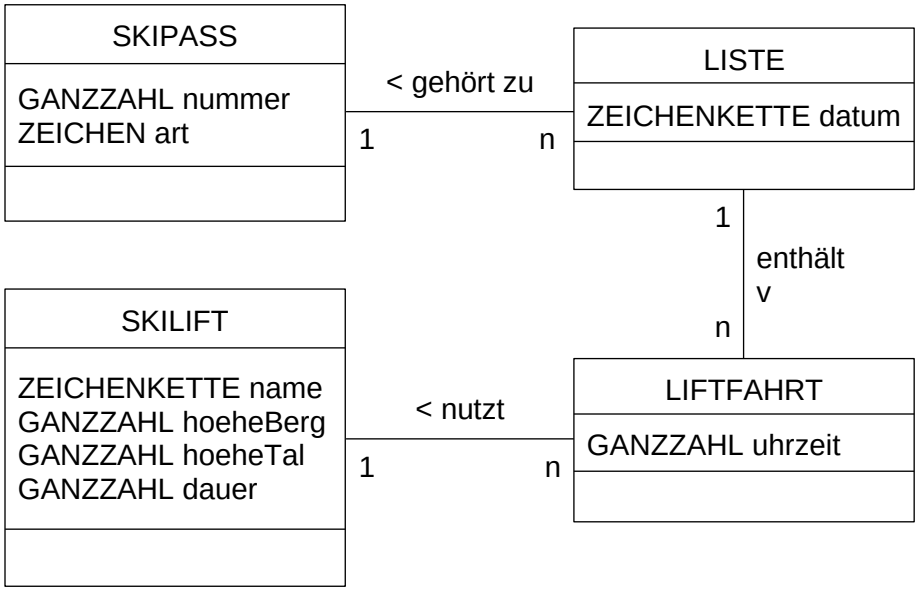
In der Klasse KNOTEN:

```
boolean istVorhanden(int skipassnr) {
    if (inhalt.nummerGeben() == skipassnr) {
        return true;
    } else {
        if (inhalt.nummerGeben() < skipassnr) {
            return nachfolgerRe.istVorhanden(skipassnr);
        } else {
            return nachfolgerLi.istVorhanden(skipassnr);
        }
    }
}

void sortiertAusgeben(char skipassart) {
    nachfolgerLi.sortiertAusgeben(skipassart);
    if (inhalt.artGeben() == skipassart) {
        inhalt.datenAusgeben();
    }
    nachfolgerRe.sortiertAusgeben(skipassart);
}
```

In der Klasse ABSCHLUSS:

```
boolean istVorhanden(int skipassnr) {
    return false;
}

void sortiertAusgeben(char skipassart) {
}
```

2a



5

b

Da die Anzahl der Liftfahrten von Fall zu Fall stark variieren kann, ist eine dynamische Datenstruktur, wie z. B. eine Liste sinnvoll.

Das Entwurfsmuster Kompositum anzuwenden hat beispielsweise den Vorteil, dass durch die Abschluss-Objekte Fallunterscheidungen bei Rekursionen vermieden werden können.

Die Daten sind insofern von der Struktur getrennt, als dass die Daten in der Klasse LIFTFAHRT gespeichert sind und die Knoten nur eine Referenz darauf enthalten.

3

c Mögliche Lösung in Python: 4

In der Klasse LISTE:

```
def liftfahrtEinfuegen(self, skilift: Skilift):
    neuerKnoten: Knoten = Knoten(self.anfang, Liftfahrt(skilift))
    self.anfang = neuerKnoten
```

In der Klasse KNOTEN:

```
def __init__(self, n: Listenelement, i: Liftfahrt):
    self.nachfolger = n
    self.inhalt = i
```

In der Klasse LIFTFAHRT:

```
def __init__(self, l: Skilift):
    self.uhrzeit = Uhr.minutenGeben()
    self.skilift = l
```

Mögliche Lösung in Java:

In der Klasse LISTE:

```
void liftfahrtEinfuegen(Skilift skilift) {
    Knoten neuerKnoten = new Knoten(anfang, new Liftfahrt(skilift));
    anfang = neuerKnoten;
}
```

In der Klasse KNOTEN:

```
Knoten(Listenelement n, Liftfahrt i) {
    nachfolger = n;
    inhalt = i;
}
```

In der Klasse LIFTFAHRT:

```
Liftfahrt(Skilift l) {
    uhrzeit = Uhr.minutenGeben();
    skilift = l;
}
```

d | 5

Mögliche Lösung in Python:

In der Klasse LISTE:

```
def liftfahrtenAusgeben(self):
    self.anfang.liftfahrtenAusgeben()
```

In der Klasse LISTENELEMENT:

```
@abc.abstractmethod
def liftfahrtenAusgeben(self):
    pass
```

In der Klasse KNOTEN:

```
def liftfahrtenAusgeben(self):
    self.nachfolger.liftfahrtenAusgeben()
    self.inhalt.datenAusgeben()
```

In der Klasse ABSCHLUSS:

```
def liftfahrtenAusgeben(self):
    pass
```

In der Klasse LIFTFAHRT:

```
def datenAusgeben(self):
    print(self.skilift.nameGeben())
```

Mögliche Lösung in Java:

In der Klasse LISTE:

```
void liftfahrtenAusgeben() {
    anfang.liftfahrtenAusgeben();
}
```

In der Klasse LISTENELEMENT:

```
abstract void liftfahrtenAusgeben();
```

In der Klasse KNOTEN:

```
void liftfahrtenAusgeben() {
    nachfolger.liftfahrtenAusgeben();
    inhalt.datenAusgeben();
}
```

In der Klasse ABSCHLUSS:

```
void liftfahrtenAusgeben() {
}
```

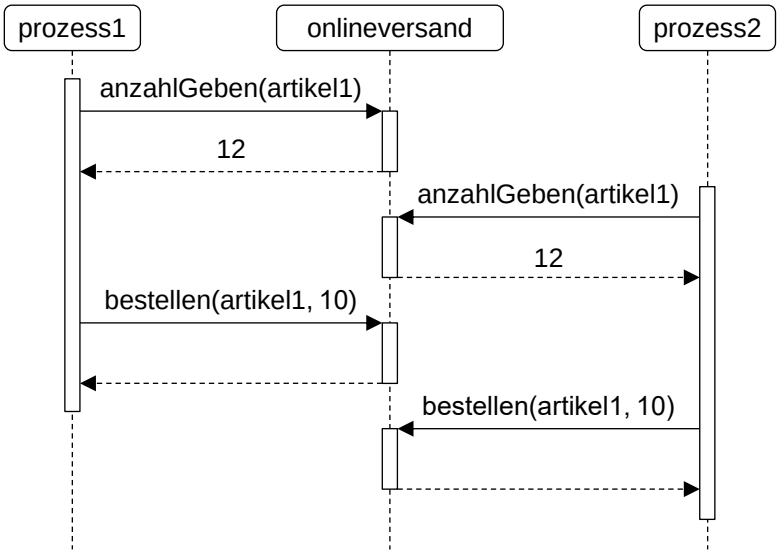
In der Klasse LIFTFAHRT:

```
void datenAusgeben() {
    System.out.println(skilift.nameGeben());
}
```

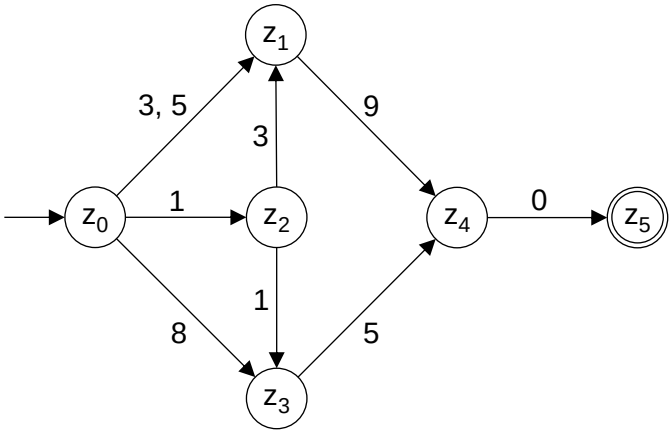
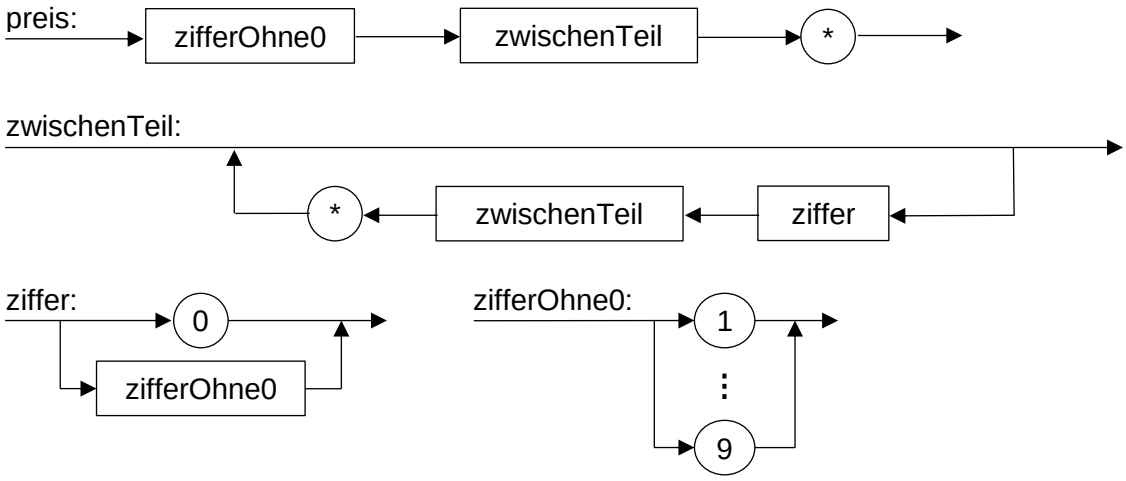
		30

Nr.		BE
1a	Z. B.: 1. FETCH-Phase – Teil 1: Opcode aus der Speicherzelle holen, deren Adresse im Befehlszähler steht; Opcode wird im Befehlsregister abgelegt; Befehlszähler wird um 1 erhöht. 2. DECODE-Phase: Entschlüsselung des Opcodes (hier in „add“). 3. FETCH-Phase – Teil 2: Operandenteil aus der Speicherzelle holen, deren Adresse im Befehlszähler steht (hier Inhalt der Speicherzelle x); Operand wird im Befehlsregister abgelegt; Befehlszähler wird um 1 erhöht. 4. EXECUTE-Phase: Summe aus Akkumulatorinhalt und Inhalt der Speicherzelle x wird von ALU berechnet und das Ergebnis in den Akkumulator geschrieben.	2
b	Die Ausführung von *rechnen1* kommt genau dann zu einem Ende, wenn $a \leq b$ ist. Die Ausführung von *rechnen2* terminiert immer. Die Methode *rechnen1* ist rekursiv, da sie in ihrer Definition einen Aufruf von sich selbst und eine Abbruchbedingung enthält.	2
c	rechnen1(6, 9) → 6 + rechnen1(7, 9) → 6 + 7 + rechnen1(8, 9) → 6 + 7 + 8 + rechnen1(9, 9) → 6 + 7 + 8 + 9 → 30 In beiden Methoden wird ausgehend vom Startwert a die Summe der ganzen Zahlen n mit $a \leq n \leq b$ ermittelt. In *rechnen1* geschieht dies über rekursive Aufrufe bei gleichzeitigem Inkrementieren des ersten Aufrufparameters, bei *rechnen2* mithilfe einer Variablen s, die den Startwert 0 hat und iterativ um a, a + 1, … erhöht wird.	2

d	Die Adressen der Speicherzellen, in denen Werte von Variablen abgelegt sind, werden mit Symbolen bezeichnet, die genauso heißen wie die entsprechenden Variablen. Die Adresse der Speicherzelle für den Rückgabewert wird mit dem Symbol *ergebnis* bezeichnet.
	`loadi 0`
	`store s`
	`wh: load b`
	`sub a`
	`jlt end`
	`load a`
	`add s`
	`store s`
	`loadi 1`
	`add a`
	`store a`
	`jmp wh`
	`end: load s`
	`store ergebnis`
	`hold`
		4

2a	 Der Bestand von *artikel1* soll durch *prozess2* beim letzten dargestellten Methodenaufruf um 10 reduziert werden, wodurch es zu einer für *prozess2* nicht erklärbaren Fehlermeldung oder zu einer Inkonsistenz kommen kann.	3
b	Monitorkonzept: Ein Monitor setzt sich aus einem oder mehreren Programmabschnitten bzw. Methoden eines Objekts zusammen, welche immer nur von einem Prozess genutzt werden können. Die von den Bestellprozessen ausgeführten Methoden müssen zu einer Methode zusammengefasst werden, die die Aufrufe von *anzahlGeben* und *bestellen* enthält und die durch den Monitor des Objekts *onlineversand* synchronisiert wird.	2
3a	Lieferant 1 hat den Gabelstapler reserviert und fordert zudem den Lagerverwalter für die Aktualisierung der Bestandsliste an. Lieferant 2 ist mit dem Lagerverwalter beim Aktualisieren der Bestandsliste und fordert den Gabelstapler an, um die Ware ins Lager zu bringen. Sofern eine Anlieferung erst abgeschlossen werden kann, wenn beide Vorgänge erledigt sind, liegt eine Verklemmung vor. Coffman-Bedingungen: • Wechselseitiger Ausschluss (mutual exclusion): Es gibt mindestens zwei Ressourcen, die nur von einem Prozess gleichzeitig genutzt werden können. • Belegen und warten (hold and wait): Ein Prozess muss eine Ressource behalten, während er auf eine weitere Ressource wartet. • Ununterbrechbarkeit (no preemption): Zugewiesene Ressourcen können einem Prozess nicht gewaltsam wieder entrissen werden. • Zyklisches Warten (circular wait): Es gibt eine zyklische Kette von Prozessen, die bereits eine oder mehrere Ressourcen zugewiesen bekommen haben und die gleichzeitig auf weitere Ressourcen warten, welche bereits dem jeweils nächsten Prozess in der Kette zugesprochen wurden. Eine Verklemmung erkennt man an einem Zyklus im Betriebsmittelgraphen.	4

b	Z. B.: • Es muss zuerst der Lagerverwalter aufgesucht und die Bestandsliste aktualisiert werden, bevor der Gabelstapler reserviert werden darf. „Circular wait“ wird umgangen. • Sofern ein Lieferant den Gabelstapler reserviert hat und der Lagerverwalter nicht verfügbar ist, wird der Gabelstapler wieder freigegeben. „Hold and wait“ wird umgangen.	1
4a	Nachdem ein Kunde einen Artikel gekauft hat, ermittelt die Methode ausgehend von der Warengruppe dieses Artikels einen Cluster von Warengruppen. Um den Kunden zu weiteren Einkäufen zu animieren, können ihm Angebote aus dem ermittelten Cluster unterbreitet werden.	1
b	Methode erreichbareAusgeben(start) setze alle Knoten auf unbesucht besuchen(start) endeMethode Methode besuchen(gruppe) setze gruppe auf besucht gib gruppe aus wiederhole für alle Nachbarknoten gruppe_n von gruppe wenn gruppe_n noch nicht besucht dann besuchen(gruppe_n) endeWenn endeWiederhole endeMethode	4
5a	Z. B.: manuelle Überprüfung, automatisiertes Testen, Debuggen, Komponententests und Integrationstests Hier ist sowohl das automatisierte Testen anhand vieler Testfälle, das Debuggen sowie ggf. das Durchführen von Komponententests mit anschließenden Integrationstests geeignet, da die Software bereits existiert und im Einsatz ist.	3
b	Die Refaktorierung des Quelltextes führt zur Erhöhung der Lesbarkeit und der Verständlichkeit sowie zur Vermeidung von Redundanz und erleichtert die Erweiterbarkeit.	1
c	Dokumentation, z. B.: • Die Entwickler-/Systemdokumentation trägt zum leichteren Verständnis des Quelltextes für andere Programmierer bzw. zu einem späteren Zeitpunkt bei. Dadurch wird leichtere Wartbarkeit, Korrektur und Erweiterbarkeit erreicht. • Die Benutzerdokumentation ist eine Systembeschreibung für Anwender, die Hinweise zur Installation und Bedienung enthält und Fehlbedienung vermindern soll.	1
		30

Nr.		BE
1a	 Der Attributwert 6 steht für den Fehlerzustand.	5
b	390, 590, 850, 1150, 1390	3
c	preis = '3' 5bis9 ziffer | 4bis9 ziffer ziffer | '1' 0bis2 ziffer ziffer | '1' '3' 0bis8 ziffer | '1' '3' '9' '0'; 0bis2 = '0' | '1' | '2'; 0bis4 = 0bis2 | '3' | '4'; 0bis8 = 0bis4 | '5' | '6' | '7' | '8'; 5bis9 = '5' | '6' | '7' | '8' | '9'; 4bis9 = '4' | 5bis9; ziffer = 0bis8 | '9';	4
d	 Da die Anzahl der Stellen des Preises nicht begrenzt ist, diese aber gezählt werden muss, um die entsprechende Anzahl an Sternchen zu setzen, müsste ein Automat, der L_3 erkennt, unendlich viele Zustände haben. Die Sprache L_3 ist also nicht regulär.	4
e	Beim Wasserfallmodell sind die einzelnen Projektphasen im Vorfeld fest definiert. Das gesamte Projekt und somit auch die Komponente zur Steuerung wird erst nach Abschluss der Softwareentwicklung ausgeliefert. Bei einer agilen Vorgehensweise werden dem Kunden lauffähige Komponenten der Software frühzeitig zur Verfügung gestellt und regelmäßig mit ihm abgestimmt.	2

2a	Es wird eine Liste der Zahlen von 2 bis 9 erzeugt. Anschließend wird für jede Zahl i aus {2; 3; ...; 9} jeweils das Doppelte, Dreifache usw. von i, sofern vorhanden, aus der Liste gestrichen. Es bleiben nur die Primzahlen 2, 3, 5 und 7 in der Liste übrig.	2
b	Verschiedene Ansätze sind denkbar, es genügt z. B. eine der folgenden Ideen: • Es genügt, i * k nur für i ≤ k zu berechnen. Daher reicht es aus, die innere Wiederholung nicht bei k = 2, sondern bei k = i beginnen zu lassen. • Es werden alle Produkte i * k mit i ≤ n und k ≤ n berechnet. Davon sind diejenigen nicht nötig, für die i * k > n gilt. Es genügt, das k der inneren Wiederholung nicht von k = 2 bis k = n laufen zu lassen, sondern nur solange i * k ≤ n gilt.	2
c	$10^8 : 10^5 = 10^3$, d. h. n wird vertausendfacht. **i** Bei linearem Verhalten wächst die Laufzeit in gleichem Maße, d. h. die Ausführung würde $20 \cdot 10^3$ s (≈ 5,6 h) dauern. **ii** Bei quadratischem Verhalten ergibt sich die $(10^3)^2$-fache Laufzeit, d. h. die Ausführung würde $20 \cdot 10^6$ s (≈ 231 d) dauern. Der gegebene Algorithmus hat wegen der verschachtelten, linear von n abhängigen Zählwiederholungen ein quadratisches Laufzeitverhalten.	4
3	*isthaltend(test, test)* kann *wahr* oder *falsch* zurückgeben. Gibt *isthaltend(test, test)* den Wahrheitswert *wahr* zurück, dann terminiert *test(test)*. In diesem Fall darf in der Methode *test* die Bedingung für die Wiederholung nie erfüllt sein. Dies ist nur möglich, falls *isthaltend(test, test)* den Wahrheitswert *falsch* zurück gibt – ein Widerspruch in diesem Fall. Gibt andererseits *isthaltend(test, test)* den Wahrheitswert *falsch* zurück, dann terminiert *test(test)* nicht. In diesem Fall muss in der Methode *test* die Bedingung für die Wiederholung immer erfüllt sein. Dies ist nur möglich, falls *isthaltend(test, test)* den Wahrheitswert *wahr* zurück gibt – ebenfalls ein Widerspruch. Man sieht, dass sich in den beiden möglichen Fällen ein Widerspruch ergibt, d. h., die Annahme ist falsch, es kann also keine universelle Methode *isthaltend* geben.	4
		30

IV

Nr.		BE
1a	Ein künstliches neuronales Netz mit Backpropagation ist ein überwachtes Lernverfahren. Ein solches Verfahren eignet sich zur Lösung des gegebenen Problems, denn es steht a priori eine Menge an Datensätzen zu Verfügung, die von Fachleuten gelabelt wurden. Dadurch können in der Lernphase die Resultate mit externen Sollwerten verglichen und somit „überwacht“ werden.	1
b	Beim Lernvorgang werden die Trainingsdaten in der Forward Propagation gemäß Gewichten und Schwellenwerten der Neuronen verarbeitet. Stimmt die resultierende Ausgabe nicht mit dem Sollwert überein, werden bei der Backpropagation die Parameterwerte angepasst. Im produktiven Einsatz wird die Forward Propagation mit neuen, ungelabelten Daten durchgeführt. Die Backpropagation kommt dann nicht mehr zum Einsatz.	2
c	Lebensmittel A: Aktivierung am ersten Neuron der verdeckten Schicht: $1145 \geq 0$, d. h. dieses Neuron feuert. Aktivierung am zweiten Neuron der verdeckten Schicht: $300 \geq 0$, d. h. dieses Neuron feuert. Aktivierung am dritten Neuron der verdeckten Schicht: $560 \geq 0$, d. h. dieses Neuron feuert. Aktivierung am Neuron der Ausgabeschicht: $5 \geq 0$, d. h. das Neuron feuert und das Lebensmittel A wird damit als ungesund klassifiziert. Lebensmittel B: Aktivierung am ersten Neuron der verdeckten Schicht: $-120 < 0$, d.h. dieses Neuron feuert nicht. Aktivierung am zweiten Neuron der verdeckten Schicht: $-1125 < 0$, d.h. dieses Neuron feuert nicht. Aktivierung am dritten Neuron der verdeckten Schicht: $175 \geq 0$, d. h. dieses Neuron feuert. Aktivierung am Neuron der Ausgabeschicht: $-10 < 0$, d. h. das Neuron feuert nicht und das Lebensmittel B wird damit als gesund klassifiziert.	3
d	Lernt das künstliche neuronale Netz und erreicht eine hohe Vorhersagequalität für die Trainingsdaten, dann kann es zur Überanpassung (Overfitting) gekommen sein. Das künstliche neuronale Netz reproduziert dann lediglich die Lösungen zu den Trainingsdaten und hat diese gewissermaßen auswendig gelernt. Durch das Overfitting passt das gefundene Modell nicht so gut zu den Testdaten, deswegen ergibt sich eine wesentlich schlechtere Qualität in der Vorhersage.	2
2a	Z. B. organisatorische Mängel: Auch Mitarbeitende, die die Datenbank nicht nutzen, bekommen einen Account mit unsicherem Initialpasswort, das dauerhaft gesetzt bleibt. Menschliche Fehlhandlungen: Das Vorgehen der Firma beim Anlegen der Mitarbeiteraccounts für die Datenbank könnte an Externe verraten werden. Diese könnten gezielt Mitarbeiterkennungen zusammen mit dem Initialpasswort für Anmeldeversuche nutzen und so Zugriff auf die Daten erlangen.	2
b	Vertraulichkeit: die persönlichen Umfragedaten eines Teilnehmenden sollten nicht ungefragt weitergegeben werden (z. B. an Krankenkassen oder Lebensversicherer). Authentizität: die Umfragedaten sollten nicht manipuliert werden (z. B. für Werbezwecke).	2

c	Der k-Means-Algorithmus fasst eine Menge von Daten zu einer vorher festgelegten Anzahl an Clustern zusammen. Diese Cluster könnten im Beispiel Kundengruppen repräsentieren, die der Lebensmittelproduzent mit passenden Produkten versorgen möchte. Es wäre nicht wirtschaftlich, für jeden Kunden ein genau passendes Produkt anzubieten. Stattdessen gruppiert man mehrere Kunden mit ähnlichen Präferenzen.	2
d	$\lvert\overline{C_1\ D}\rvert = \sqrt{(30 - 60)^2 + (40 - 50)^2} = \sqrt{900 + 100} \approx 31{,}6$ $\lvert\overline{C_2\ D}\rvert = \sqrt{(70 - 60)^2 + (70 - 50)^2} = \sqrt{100 + 400} \approx 22{,}4$ Die Distanz des Datenpunktes D ist zu C_2 geringer als zu C_1. Deswegen wird D dem Clusterzentrum C_2 zugeordnet.	3

e	Lösung in Python:
	`def klassifizieren():`
	`  minDist:float`
	`  i:int`
	`  for i in range(n):`
	`  minDist = daten[i].distanzGeben(zentren[0])`
	`  zuordnung[i] = 0`
	`  j:int`
	`  for j in range(k):`
	`  if daten[i].distanzGeben(zentren[j]) <minDist:`
	`  zuordnung[i] = j`
	
	`def zentrenAktualisieren():`
	`  i:int`
	`  for i in range(k):`
	`  sumNährwert:int=0`
	`  sumRegionalität:int=0`
	`  count:int=0`
	`  j:int`
	`  for j in range(n):`
	`  if zuordnung[j] == i:`
	`  sumNährwert= sumNährwert + daten[j].nährwertGeben()`
	`  sumRegionalität = sumRegionalität + daten[j].regionalitätGeben()`
	`  count+=1`
	`  zentren[i].nährwertSetzen(sumNährwert/count)`
	`  zentren[i].regionalitätSetzen(sumRegionalität/count)`
		7

	Lösung in Java:
	Es wird davon ausgegangen, dass die gegebenen indizierten Datenstrukturen Attribute derselben Klasse sind, in der die folgenden Methoden implementiert sind.
	`void klassifizieren() {`
	`  double minDist;`
	`  for (int i=0; i<n; i++) {`
	`  minDist = daten[i].distanzGeben(zentren[0]);`
	`  zuordnung[i] = 0;`
	`  for (int j=1; j<k; j++) {`
	`  if (daten[i].distanzGeben(zentren[j]) < minDist) {`
	`  minDist = daten[i].distanzGeben(zentren[j]);`
	`  zuordnung[i] = j;`
	`  }`
	`  }`
	`  }`
	`}`
	
	`void zentrenAktualisieren() {`
	`  int sumNährwert, sumRegionalität, count;`
	`  for (int i=0; i<k; i++) {`
	`  sumNährwert = 0;`
	`  sumRegionalität = 0;`
	`  count = 0;`
	`  for (int j=0; j<n; j++) {`
	`  if (zuordnung[j] == i) {`
	`  sumNährwert = sumNährwert + daten[j].nährwertGeben();`
	`  sumRegionalität = sumRegionalität + daten[j].regionalitätGeben();`
	`  count++;`
	`  }`
	`  }`
	`  zentren[i].nährwertSetzen(sumNährwert/count);`
	`  zentren[i].regionalitätSetzen(sumRegionalität/count);`
	`  }`
	`}`	

3a	Die Daten von Stammgästen von Fitnessstudios, die zum Training des künstlichen neuronalen Netzes verwendet wurden, sind nicht repräsentativ. Daher sind fehlerbehaftete Empfehlungen für Personen außerhalb dieser Gruppe nicht unwahrscheinlich.	1
b	Hyperparameter sind beispielsweise die Anzahl der Schichten und die Anzahl der Neuronen pro Schicht. Eine Veränderung dieser Werte löst das Problem schlecht gewählter Trainingsdaten allerdings nicht.	2
c	Im Folgenden wird eine mögliche Argumentation grob skizziert. Chancen: individualisierte Tipps zu einer gesunden Lebensführung unter der Voraussetzung eines gut trainierten Systems (individuelle Ebene), Reduktion der Kosten für das Gesundheitssystem (gesellschaftliche Ebene) Risiken: Abfluss von Daten, Fehleranfälligkeit und mangelhafte Zuverlässigkeit Fazit: Bei geeigneter Berücksichtigung der Risiken (z. B. Vermeidung des Abflusses von Daten durch regulatorische Rahmenbedingungen) ergeben sich auf individueller und gesellschaftlicher Ebene Potentiale der Technologie.	3
		30