

Abiturprüfung 2015

INFORMATIK

Hinweise zur Korrektur und Bewertung

(nicht für den Prüfling bestimmt)

Die Korrekturhinweise enthalten keine vollständigen Lösungen der Aufgaben, sondern einen Abriss des Erwartungshorizontes. Nicht genannte, aber *gleichwertige* Lösungswege und Begründungsansätze sind *gleichberechtigt*.

Die in den Lösungshinweisen gewählte objektorientierte Programmiersprache ist Java.

Die Bewertung der erbrachten Prüfungsleistungen hat sich an der bei jeder Teilaufgabe am linken Rand des Angabenblattes vermerkten, maximal erreichbaren Zahl von Bewertungseinheiten (BE) zu orientieren.

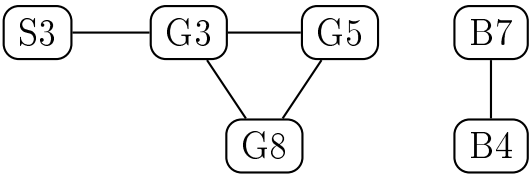
Umrechnung der erreichten Bewertungseinheiten in Notenpunkte:

Die insgesamt erreichten Bewertungseinheiten werden nach der folgenden Tabelle in Notenpunkte umgesetzt:

Notenpunkte	Notenstufen	Bewertungseinheiten	Intervalle in %
15	+1	120 ... 115	15
14	1	114 ... 109	
13	1–	108 ... 103	
12	+2	102 ... 97	15
11	2	96 ... 91	
10	2–	90 ... 85	
9	+3	84 ... 79	15
8	3	78 ... 73	
7	3–	72 ... 67	
6	+4	66 ... 61	15
5	4	60 ... 55	
4	4–	54 ... 49	
3	+5	48 ... 41	20
2	5	40 ... 33	
1	5–	32 ... 25	
0	6	24 ... 0	20

Aufgabe	BE	Hinweise
1. a)	10	vgl. Lösung zu Teilaufgabe 1 b
b)	4	<pre> classDiagram class SPIEL { } class KARTENGRUPPE { } class SPIELER { <<abstract>> spielzugAusfuehren() <<abstract>> } class PERSON { spielzugAusfuehren() } class PCSPIELER { spielzugAusfuehren() } class KARTE { farbe wert } SPIEL "1" -- "0..1" KARTENGRUPPE : hat als Ablage > SPIEL "0..1" -- "1" KARTENGRUPPE : hat als Talon > 1 SPIELER "2..4" -- "1" SPIEL : ist beteiligt an SPIELER "0..1" -- "1" KARTENGRUPPE : hat in der Hand KARTENGRUPPE "1" -- "0..40" KARTE : enthält> SPIELER < -- PERSON SPIELER < -- PCSPIELER </pre> The diagram illustrates the relationships between several classes in a game system. The SPIEL class is associated with KARTENGRUPPE via two directed associations: one labeled "hat als Ablage >" with multiplicity 1 at SPIEL and 0..1 at KARTENGRUPPE, and another labeled "hat als Talon > 1" with multiplicity 0..1 at SPIEL and 1 at KARTENGRUPPE. The SPIELER class (marked as abstract) has a directed association to SPIEL labeled "ist beteiligt an" with multiplicity 2..4 at SPIELER and 1 at SPIEL. SPIELER is also associated with KARTENGRUPPE via a directed association labeled "hat in der Hand" with multiplicity 0..1 at SPIELER and 1 at KARTENGRUPPE. KARTENGRUPPE is associated with KARTE via a directed association labeled "enthält>" with multiplicity 1 at KARTENGRUPPE and 0..40 at KARTE. The KARTE class has attributes "farbe" and "wert". SPIELER has an abstract method "spielzugAusfuehren()". PERSON and PCSPIELER are subclasses of SPIELER, both implementing the "spielzugAusfuehren()" method.
c)	4	Zwischen den Klassen SPIELER und SPIEL kann beispielsweise die Beziehung „ist beteiligt an“ bidirektional implementiert werden, da ein Objekt der Klasse SPIELER auf den Talon zugreifen muss, was über das Objekt der Klasse SPIEL möglich ist. Umgekehrt muss das Objekt der Klasse SPIEL jedes beteiligte Objekt der Klasse SPIELER zu gegebener Zeit auffordern, seinen nächsten Spielzug auszuführen.
d)	3	Abwägen im Hinblick auf zwei Aspekte, z. B. auf die Anzahl der zu verwaltenden Elemente, den Speicherplatzbedarf oder das Einfügen von Objekten.

e)	5	
f)	4	Es wird 10 000-mal eine zufällig ausgewählte Karte an den Listenanfang vorgezogen, wodurch eine zufällige Reihenfolge der Karten in der Liste erzeugt wird. Verwendung im Spiel: Mischen einer Kartengruppe.
g)	16	<pre> class Kartengruppe{ private Listenelement erster; void vorneEinfuegen(Karte neueKarte){ erster = new Knoten(neueKarte, erster); } Karte karteGeben(int position){ return erster.karteGeben(position); } } abstract class Listenelement{ abstract Karte karteGeben(int position); } class Abschluss extends Listenelement{ Karte karteGeben(int position){ return null; } } </pre>

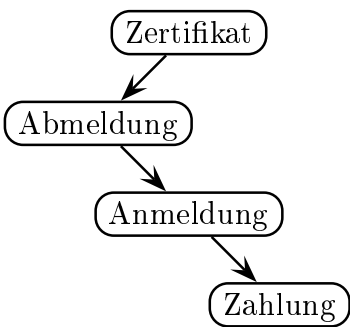
		<pre> class Knoten extends Listenelement{ private Karte inhalt; private Listenelement naechster; Knoten(Karte k, Listenelement n){ inhalt = k; naechster = n; } Karte karteGeben(int position){ if (position == 0){ return inhalt; } else { return naechster.karteGeben(position-1); } } } </pre>
2. a)	4	 Der Graph ist z. B. ungerichtet, unzusammenhängend, unbewertet/ungewichtet und zyklisch.
b)	3	Der Eintrag <i>wahr</i> als Wert von <i>matrix[i][j]</i> bedeutet, dass die Karte mit dem Index <i>i</i> zur Karte mit dem Index <i>j</i> passt.
c)	8	Z. B. Tiefensuche; das Ergebnis steht nach Anwendung des Algorithmus im Attribut <i>zaehler</i>. <pre> Methode zaehlenStarten(startKnotenNr) setze zaehler auf den Wert 0 wiederhole für alle Knoten setze Markierung des Knotens auf den Wert für unbesucht endeWiederhole zaehlenDurchfuehren(startKnotenNr) endeMethode </pre>

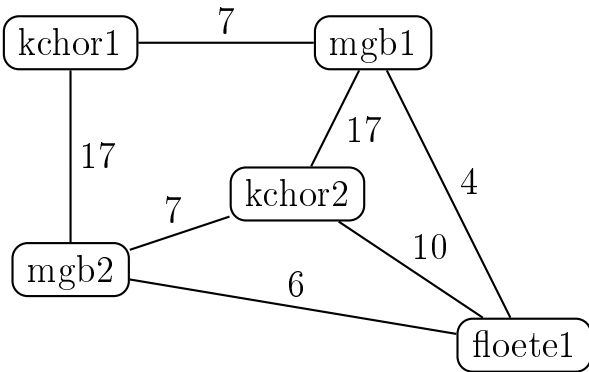
		Methode zaehlenDurchfuehren(knotenNr) markiere den Knoten mit der Nummer knotenNr als besucht zähle j von 0 bis anzahlKnoten - 1 wenn matrix[knotenNr][j] ist wahr und der Knoten mit der Nummer j wurde noch nicht besucht dann erhöhe den Wert von zaehler um 1 zaehlenDurchfuehren(j) endeWenn endeZähle endeMethode
d)	5	Der Algorithmus stellt für S8 und S9 dieselbe Erreichbarkeit fest. Bei Auswahl von S9 kann im Anschluss entweder S8 oder R9 gespielt werden - es bleibt in jedem Fall eine Karte übrig. Bei Auswahl von S8 wird in den nächsten Zügen zuerst S9 und dann R9 abgelegt - in diesem Fall bleibt keine Karte übrig.
3. a)	3	Bei 3 Ebenen sind es $1 \cdot 3 \cdot 3 = 9$ Spieler; bei n Ebenen 3^{n-1} Spieler.
b)	4	Bei 3 Ebenen sind es $1 + 3 = 4$ Spiele; bei n Ebenen $1 + 3 + 3^2 + \dots + 3^{n-2}$ Spiele.
c)	7	<pre> classDiagram class SPIELPLAN class BAUMELEMENT["BAUMELEMENT\n{abstract}"] class ABSCHLUSS class KNOTEN class SPIELER SPIELPLAN --> BAUMELEMENT : sieger > 1 BAUMELEMENT --> KNOTEN : 3 KNOTEN --> SPIELER : inhalt 1..n KNOTEN --> KNOTEN : spieler 0,1 ABSCHLUSS -- > BAUMELEMENT </pre>
	80	

Aufgabe	BE	Hinweise
1.	12	<pre> classDiagram class PERSON { vorname nachname geschlecht } class SCHUELER { schuelernummer lehrerlisteDrucken() } class LEHRKRAFT { lehrkraftkuerzel } class MUSIKGRUPPE { altersstufe niveau bezeichnung unterrichtszeit unterrichtsort schuelerlisteDrucken() hinzufuegen(schueler) entfernen(schueler) } PERSON < -- SCHUELER PERSON < -- LEHRKRAFT SCHUELER "n" -- "m" MUSIKGRUPPE : besucht LEHRKRAFT "1" -- "n" MUSIKGRUPPE : unterrichtet </pre>
2. a)	4	Erläutert werden können beispielsweise folgende Gesichtspunkte: Längenbegrenzung beim Feld, Aufwand für Einfügen und Entfernen.

b)	5	Eine Warteschlange ist eine dynamische Datenstruktur, bei der Datenelemente in der gleichen Reihenfolge entnommen werden, in der sie eingefügt wurden. Diese Datenstruktur ist für die Terminverwaltung ungeeignet. Beim Löschen abgelaufener Termine müsste gewährleistet sein, dass die Termine exakt in ihrer zeitlichen Reihenfolge eingefügt wurden. Dies ist aber unrealistisch, da beispielsweise Termine sehr kurzfristig bekannt werden können und dann i. d. R. schon spätere Termine abgespeichert sind.
3. a)	10	<pre> classDiagram class SCHUELER { } class ABSENZENLISTE { +anzahlAbsenztageGeben() +bereinigen() } class LISTENELEMENT { <<abstract>> +anzahlAbsenztageGeben() +bereinigen() } class ABSCHLUSS { +anzahlAbsenztageGeben() +bereinigen() } class KNOTEN { +anzahlAbsenztageGeben() +bereinigen() } class ABSENZ { +datum +absenzgrund +istAbsenzgrund(grund) } SCHUELER "1" -- "1" ABSENZENLISTE : hat ABSENZENLISTE -- "1" LISTENELEMENT : erster LISTENELEMENT < -- ABSCHLUSS LISTENELEMENT < -- KNOTEN KNOTEN -- "1" LISTENELEMENT : ^ nachfolger KNOTEN -- "1" ABSENZ : verwaltet </pre> Die Forderungen sind sinnvoll, weil beispielsweise • dadurch die Aufgaben (Organisation der Listenstruktur, Datenhaltung) auf spezifische Klassen aufgeteilt sind. Damit ist die Implementierung übersichtlicher; • dadurch Teile des Quelltextes ohne großen Aufwand wiederverwendet werden können; • mit dem Kompositum eine bewährte Strategie eingesetzt wird.

b)	17	<pre> class Absenzenliste{ private Listenelement erster; int anzahlAbsenztageGeben(){ return erster.anzahlAbsenztageGeben(); } void bereinigen(){ erster = erster.bereinigen(); } } abstract class Listenelement{ abstract int anzahlAbsenztageGeben(); abstract Listenelement bereinigen(); } class Knoten extends Listenelement{ private Absenz verwaltet; private Listenelement nachfolger; int anzahlAbsenztageGeben(){ return 1 + nachfolger.anzahlAbsenztageGeben(); } Listenelement bereinigen(){ nachfolger = nachfolger.bereinigen(); if(verwaltet.istAbsenzgrund("öffentliche Aufführung")){ return nachfolger; } else { return this; } } } class Abschluss extends Listenelement{ int anzahlAbsenztageGeben(){ return 0; } Listenelement bereinigen(){ return this; } } </pre>
----	----	---

4. a)	7	 <pre> graph TD Zertifikat --> Abmeldung Abmeldung --> Anmeldung Anmeldung --> Zahlung </pre> Der Vorteil des geordneten Binärbaums gegenüber der Liste besteht darin, dass bei einem balancierten Baum die Suche nach einem bestimmten Schlüssel i. d. R. schneller geht. Der vorliegende Baum ist zur Liste entartet. Neue Begriffe werden mit großer Wahrscheinlichkeit in den linken Teilbaum des Knotens „Zahlung“ eingefügt. Werden nun sehr viele Elemente eingefügt, die zu einem balancierten Teilbaum führen, so bleibt der Vorteil der schnelleren Suche – trotz der Entartung auf den ersten vier Ebenen – erhalten. Je mehr das weitere Einfügen zu einem nicht balancierten Teilbaum führt, desto weniger kommt der Vorteil des (balancierten) geordneten Binärbaums zum Tragen.
b)	4	<pre> void alleZeigen(){ linkerNachfolger.alleZeigen(); verwaltet.ausgeben(); rechterNachfolger.alleZeigen(); } </pre>

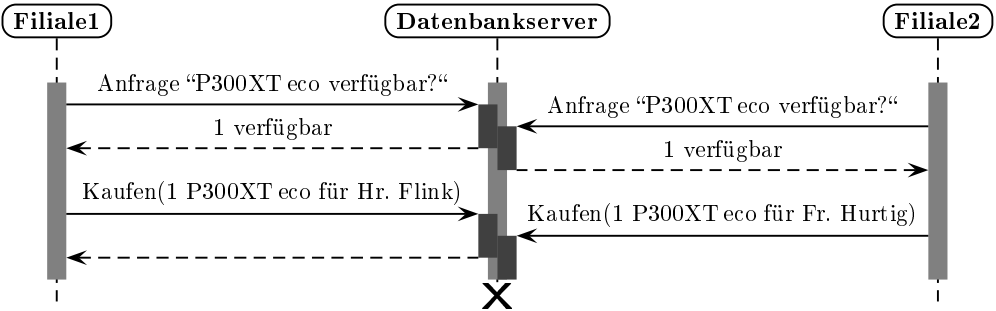
5. a)	8	Möglicher Graph:  Adjazenzmatrix: <table data-bbox="525 770 1321 1061"><tr><th></th><th>mgb1</th><th>mgb2</th><th>kchor1</th><th>kchor2</th><th>floete1</th></tr><tr><th>mgb1</th><td></td><td></td><td>7</td><td>17</td><td>4</td></tr><tr><th>mgb2</th><td></td><td></td><td>17</td><td>7</td><td>6</td></tr><tr><th>kchor1</th><td>7</td><td>17</td><td></td><td></td><td></td></tr><tr><th>kchor2</th><td>17</td><td>7</td><td></td><td></td><td>10</td></tr><tr><th>floete1</th><td>4</td><td>6</td><td></td><td>10</td><td></td></tr></table>		mgb1	mgb2	kchor1	kchor2	floete1	mgb1			7	17	4	mgb2			17	7	6	kchor1	7	17				kchor2	17	7			10	floete1	4	6		10	
	mgb1	mgb2	kchor1	kchor2	floete1																																	
mgb1			7	17	4																																	
mgb2			17	7	6																																	
kchor1	7	17																																				
kchor2	17	7			10																																	
floete1	4	6		10																																		
b)	2	Der Graph ist z. B. ungerichtet, bewertet, zyklisch, zusammenhängend.																																				
c)	3	Ein nicht zusammenhängender Graph besteht aus (mindestens) zwei (Teil-)Graphen, die durch keine Kante verbunden sind. Damit könnte man die Musikgruppen in (mindestens) zwei Teilmengen aufteilen, wobei es kein Kind gibt, das Musikgruppen aus beiden Teilmengen besucht.																																				

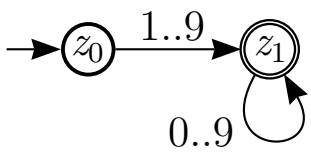
d)	8	$i = 0$ $b = \text{wahr}$ $j = 0:$ $i = \text{indizes}[0] \rightarrow b = \text{falsch}$ $i = 1$ $b = \text{wahr}$ $j = 0:$ $\text{matrix}[1][0] = 0$ und $i \neq \text{indizes}[0]$ $j = 1:$ $\text{matrix}[1][3] = 0$ und $i \neq \text{indizes}[1]$ Ausgabe: <code>piano2</code> $i = 2$ $b = \text{wahr}$ $j = 0:$ $\text{matrix}[2][0] = 18 > 0 \rightarrow b = \text{falsch}$ $i = 3$ $b = \text{wahr}$ $j = 1:$ $i = \text{indizes}[1] \rightarrow b = \text{falsch}$ $i = 4$ $b = \text{wahr}$ $j = 0:$ $\text{matrix}[4][0] = 3 > 0 \rightarrow b = \text{falsch}$ Im Kontext bedeutet das: Es werden die Bezeichnungen der Musikgruppen ausgegeben, die keinen gemeinsamen Schüler bzw. keine gemeinsame Schülerin mit den übergebenen Musikgruppen haben.
	80	

Aufgabe	BE	Hinweise												
1. a)	3	<table><tr><td colspan="2" rowspan="2"> wahr falsch </td><td colspan="2"> $U \leq 200000$ </td></tr><tr><td colspan="2"> wahr falsch </td></tr><tr><td colspan="2" rowspan="2"> $P = 0,002 \cdot U$ </td><td colspan="2"> $U \leq 400000$ </td></tr><tr><td> $P = 0,003 \cdot U$ </td><td> $P = 0,004 \cdot U + 1000$ </td></tr></table>	wahr falsch		$U \leq 200000$		wahr falsch		$P = 0,002 \cdot U$		$U \leq 400000$		$P = 0,003 \cdot U$	$P = 0,004 \cdot U + 1000$
wahr falsch		$U \leq 200000$												
		wahr falsch												
$P = 0,002 \cdot U$		$U \leq 400000$												
		$P = 0,003 \cdot U$	$P = 0,004 \cdot U + 1000$											
b)	7	<pre>1: load 101 2: subi 200000 3: jle 16 // U ≤ 200000 4: load 101 5: subi 400000 6: jle 12 // U ≤ 400000 7: load 101 8: muli 4 9: divi 1000 10: addi 1000 11: jmp 19 12: load 101 13: muli 3 14: divi 1000 15: jmp 19 16: load 101 17: muli 2 18: divi 1000 19: store 102 20: hold</pre>												

2. a)	5	<pre> graph LR start(()) --> z0((z0)) z0 -- b --> z1((z1)) z0 -- Z --> z2((z2)) z1 -- "S, b" --> z4((z4)) z1 -- Z --> z3((z3)) z2 -- b --> z3 z2 -- "S, Z" --> z5((z5)) z4 -- Z --> z6(((z6))) z3 -- "S, Z, b" --> z6 z5 -- b --> z6 </pre>
b)	3	Passwort = (b(S b)Z) (bZ(S Z b)) (Zb(S Z b)) (Z(S Z)b);
c)	8	<pre> class Passworttester{ private int zustand; boolean passwortTesten(String eingabe){ zustand = 0; for (int i = 0; i < eingabe.length(); i++){ zustandWechseln(eingabe.charAt(i)); } return (zustand == 6); } </pre>

		<pre> void zustandWechseln(char aktuellesZeichen){ switch (zustand){ case 0: switch (aktuellesZeichen){ case 'b': zustand = 1; break; case 'Z': zustand = 2; break; default: zustand = -1; break; } break; case 6: zustand = -1; break; ... } } </pre>
d)	2	Die Aufhebung der Einschränkung erhöht die Anzahl der möglichen Passwörter; unter Kenntnis der Vorgaben führt eine Brute-Force-Attacke daher nicht so schnell zum Ziel.
e)	5	Man fordert ein längeres Passwort, z. B. aus 11 Zeichen. Man verwendet ein größeres Eingabealphabet, beispielsweise durch Hinzunahme von Großbuchstaben. Mit dieser Vorgabe gibt es an jeder Stelle $26 + 26 + 10 + 32 = 94$ Möglichkeiten. Also ergibt sich bei einem Passwort aus 11 Zeichen im schlechtesten Fall eine ungefähre Suchzeit von $\frac{94^{11}}{2 \cdot 10^9} s \approx 80000a$. Dies ist deutlich länger als die vorgesehene Gültigkeitsdauer von einem Jahr.

3. a)	5	 Nebenläufige Prozesse sind Prozesse, die gleichzeitig ablaufen können, auf gemeinsame Ressourcen zugreifen und dabei zu inkonsistenten Daten führen können. Im Beispiel sind dies die Verkaufsvorgänge in den Filialen. Ein kritischer Abschnitt ist ein Programmbereich, in dem die Veränderung einer Ressource durch nebenläufige Prozesse zu Inkonsistenzen führen kann. Hier ist dieser Abschnitt der Ablauf von der Anfrage bis zum Abschluss des Kaufes.
b)	2	Ein Monitor stellt einen geschützten Bereich zur Verfügung, der den Zugriff immer nur jeweils einem Prozess erlaubt.
	40	

Aufgabe	BE	Hinweise
1. a)	2	$\langle \text{zahl} \rangle \rightarrow \langle \text{zahl} \rangle \langle \text{ziffer} \rangle$ $\rightarrow \langle \text{zahl} \rangle \langle \text{ziffer} \rangle \langle \text{ziffer1} \rangle$ $\rightarrow \langle \text{ziffer1} \rangle 0 \ 3$ $\rightarrow 2 \ 0 \ 3$
b)	3	 <pre> graph LR start(()) --> z0((z0)) z0 -- "1..9" --> z1(((z1))) z1 -- "0..9" --> z1 </pre>
c)	7	Alphabet: $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \text{load}, \text{store}, \text{add}, \text{sub}, \text{jmp}, \text{jeq}, \text{hold}, ;\}$ Menge der Nichtterminale: $V = \{\langle \text{zahl} \rangle, \langle \text{ziffer1} \rangle, \langle \text{ziffer} \rangle, \langle \text{programm} \rangle, \langle \text{zeile} \rangle, \langle \text{befehl} \rangle, \langle \text{befehl1} \rangle\}$ Startsymbol: $\langle \text{programm} \rangle$ Produktionsregeln: $\langle \text{programm} \rangle \rightarrow \langle \text{zeile} \rangle \langle \text{programm} \rangle \mid \langle \text{zeile} \rangle$ $\langle \text{zeile} \rangle \rightarrow \langle \text{zahl} \rangle \langle \text{befehl} \rangle ;$ $\langle \text{befehl} \rangle \rightarrow \langle \text{befehl1} \rangle \langle \text{zahl} \rangle \mid \text{hold}$ $\langle \text{befehl1} \rangle \rightarrow \text{load} \mid \text{store} \mid \text{add} \mid \text{sub} \mid \text{jmp} \mid \text{jeq}$ $\langle \text{zahl} \rangle \rightarrow \langle \text{ziffer1} \rangle \mid \langle \text{zahl} \rangle \langle \text{ziffer} \rangle$ $\langle \text{ziffer1} \rangle \rightarrow 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$ $\langle \text{ziffer} \rangle \rightarrow 0 \mid \langle \text{ziffer1} \rangle$

2. a)	4	<i>potenz1</i> wird zur Berechnung von 5^4 fünfmal aufgerufen, zur Berechnung von 5^8 neunmal. $\text{potenz2}(5, 4) = \text{quadrat}(\text{potenz2}(5, 2)) =$ $= \text{quadrat}(\text{quadrat}(\text{potenz2}(5, 1))) =$ $= \text{quadrat}(\text{quadrat}(5 \cdot \text{potenz2}(5, 0))) = \text{quadrat}(\text{quadrat}(5 \cdot 1))$ benötigt vier Aufrufe. Für $\text{potenz2}(5, 8) = \text{quadrat}(\text{potenz2}(5, 4))$ ist ein Aufruf mehr nötig, also fünf Aufrufe.																		
b)	3	In Algorithmus 1 werden $n + 1$ Aufrufe benötigt; bei Verdopplung von n verdoppelt sich also ungefähr die Anzahl der Aufrufe. In Algorithmus 2 ist bei Verdoppelung von n ein zusätzlicher Methodenaufruf nötig.																		
c)	4	Algorithmus 1: $t_1 \approx 1\mu s \cdot 1000001 \approx 1s$ Algorithmus 2: $1000000 \approx 2^{20}$, $t_2 \approx 20\mu s$ (exakte Anzahl: 21 Aufrufe)																		
3. a)	4	<table> <tr> <th>z_1</th><th>z_2</th><th>erg</th></tr> <tr> <td>13</td><td>5</td><td>0</td></tr> <tr> <td>6</td><td>10</td><td>5</td></tr> <tr> <td>3</td><td>20</td><td>5</td></tr> <tr> <td>1</td><td>40</td><td>$5+20=25$</td></tr> <tr> <td>0</td><td>80</td><td>$25+40=65$</td></tr> </table>	z_1	z_2	erg	13	5	0	6	10	5	3	20	5	1	40	$5+20=25$	0	80	$25+40=65$
z_1	z_2	erg																		
13	5	0																		
6	10	5																		
3	20	5																		
1	40	$5+20=25$																		
0	80	$25+40=65$																		

b)	3	Das Speichermodell der von-Neumann-Maschine sieht die gemeinsame Nutzung des Speicherwerks durch Programme und Daten vor. Vorteile sind z. B.: • Da nur ein Speicher erforderlich ist, werden Kosten gespart. • Ein gemeinsamer Speicherbereich lässt sich flexibel auf Programm und Daten aufteilen. Nachteile sind z. B.: • Daten und Befehle müssen über denselben Bus zum Rechen- bzw. Steuerwerk transportiert werden und sich dessen Bandbreite teilen (<i>von-Neumann-Flaschenhals</i>). • Ohne eine abgesicherte Speicherverwaltung können Programme Befehlssequenzen des eigenen Programms oder fremder Programme überschreiben (absichtlich oder auch unabsichtlich); dies stellt ein Sicherheitsrisiko dar.
c)	3	Beispiele: • Enthält Speicherzelle 101 den Wert 5, dann enthält der Akkumulator nach Ablauf des Programmabschnitts den Wert -1; • enthält Speicherzelle 101 den Wert 6, dann enthält der Akkumulator nach Ablauf des Programmabschnitts den Wert 0.

d)	7	<pre>1: load 101 2: jeq 17 3: shr 4: shl 5: sub 101 6: jeq 10 7: load 103 8: add 102 9: store 103 10: load 101 11: shr 12: store 101 13: load 102 14: shl 15: store 102 16: jump 1 17: end</pre> Das Ergebnis <i>erg</i> steht in der Speicherzelle 103.
	40	