Name: Kurs: Datum:

Klausur Grundkurs Informatik - Thema: Algorithmen

Aufgabe 1 – Grundbegriffe **(9 BE)**

a) Definieren Sie den Begriff „Algorithmus“! *(1BE)*
b) Nennen & erläutern Sie stichpunktartig 4 grundlegende Eigenschaften eines Algorithmus‘! *(4BE)*
c) Geben Sie an, wann ein Algorithmus „in-situ“ und wann er „stabil“ ist!
Nennen Sie je einen Algorithmus, der die entsprechende Eigenschaft besitzt! *(2+2BE)*

Aufgabe 2 – Sortieralgorithmen **(20 BE)**

a) Bubble-Sort *(5 BE)*

Sortieren Sie die Liste

L = [2 , 5 , 6 , 1 , 3]

mit dem im Unterricht kennengelernten BubbleSort-Algorithmus. Nutzen Sie dafür die folgende Tabelle und markieren Sie in jeder Zeile (außer der letzten), welches Zahlenpaar im nächsten Schritt des Algorithmus betrachtet wird!

Schritt	Liste L
0	**[2 , 5 , 6 , 1 , 3]**
1	[2 , 5 , 6 , 1 , 3]
2	[2 , 5 , 6 , 1 , 3]
3	[2 , 5 , 1 , 6 , 3]
4	[2 , 5 , 1 , 3 , 6]
5	[2 , 5 , 1 , 3 , 6]
6	[2 , 1 , 5 , 3 , 6]
7	[2 , 1 , 3 , 5 , 6]
8	[1 , 2 , 3 , 5 , 6]
9	[1 , 2 , 3 , 5 , 6]
10	[1 , 2 , 3 , 5 , 6]

b) Quick-Sort *(8 BE)*
Sortieren Sie die Liste

L = [2 , 5 , 6 , 1 , 3]

mit dem im Unterricht kennengelernten QuickSort-Algorithmus. Nutzen Sie dafür die folgende Tabelle. Markieren Sie jeweils (beispielsweise durch Klammerung) die momentan betrachtete Teilliste und tragen Sie nur die Zwischenschritte in die Tabelle ein, bei der die Indizes *i* und *j* an einer Stelle „anhalten“.

Als Pivot-Element soll jeweils das mittlere Element der Teilliste betrachtet werden. Falls nötig soll abgerundet werden.
Tragen Sie in der Spalte „nächster Schritt“ ein, ob der Algorithmus als nächstes tauscht [swap(Index_1, Index_2)] oder sich selbst rekursiv aufruft [QS(Anfangsindex_Liste, Endindex_Liste)]!
Bearbeiten Sie nur die Aufrufe, in denen Teillisten mit mindestens 2 Elementen betrachtet werden (also nicht z.B. QS(2,2))!
Nicht benötigte Zeilen lassen Sie einfach frei.

Liste L	Pivot p	i	j	Nächster Schritt
[2 , 5 , 6 , 1 , 3]	**6**	**3**	**5**	**Swap(3,5)**
[2 , 5 , 3 , 1 , 6]	3	5	4	QS(5,5) ; QS (1,4)
[2 , 5 , 3 , 1] 6	5	2	4	Swap(2,4)
[2 , 1 , 3 , 5] 6	1	3	3	Swap(3,3)
[2 , 1 , 3 , 5] 6	1	4	2	QS(1,2) ; QS(3,4)
[2 , 1] , 3 , 5 , 6	2	1	2	Swap(1,2)
[1 , 2] , 3 , 5 , 6	1	2	1	QS(1,1) ; QS (2,2)
1 , 2 , [3 , 5] , 6	3	3	3	Swap(3,3)
1 , 2 , [3 , 5] 6	3	4	2	QS(3,3) ; QS (4,4)

c) Heap-Sort *(7 BE)*

Sortieren Sie die Liste

L = [2 , 5 , 6 , 1 , 3]

mit dem im Unterricht kennengelernten HeapSort-Algorithmus.
Geben Sie dafür zuerst die Liste L als Baum an.
Stellen Sie danach die Heap-Eigenschaft in diesem Baum her und wenden Sie HeapSort an. Geben Sie dabei jeweils nur die Bäume an, welche nach Wiederherstellung der Heap-Eigenschaft entstehen. Geben Sie außerdem den finalen Baum an, in dem alle Zahlen isoliert sind. (Sie müssen also, neben dem Baum und dem ersten Heap, nur 5 weitere Heaps zeichnen.)
Geben Sie außerdem in jedem Schritt die „isolierten“ Teile des Baumes mit an, indem sie diesen einrahmen.

Aufgabe 3 – Rekursion & Iteration **(8 BE)**

Die Fibonacci-Folge ist eine unendliche Folge natürlicher Zahlen. Sie lässt sich durch die folgende Rekursionsvorschrift beschreiben:

Fib(1) = 1

Fib(2) = 1

Fib(x) = Fib(x-1) + Fib(x-2) für alle natürlichen Zahlen $x \geq 3$

Fib(x) beschreibt dabei die x-te Zahl in der Fibonacci-Folge.

a) Zeichnen Sie ein Struktogramm für einen rekursiven Algorithmus, welcher für eine Eingabe x die x-te Fibonacci-Zahl berechnet! *(2BE)*
b) Die Fibonacci-Zahlen lassen sich, durch eine Schleife, auch mit einem iterativen Ansatz berechnen. Zeichnen Sie ein Flussdiagramm für einen iterativen Algorithmus, welcher für eine Eingabe x die x-te Fibonacci-Zahl berechnet! *(4BE)*

   (Sie können bei beiden Aufgaben davon ausgehen, dass x stets eine natürliche Zahl ≥ 1 ist.)

c) Berechnen Sie sowohl für den rekursiven, als auch für den iterativen Ansatz die jeweilige Zeitkomplexität. *(2BE)*

Aufgabe 4 – Komplexität von Algorithmen **(5 BE)**

Geben Sie für die jeweiligen Funktionen die entsprechend kleinste obere Schranke (in der im Unterricht behandelten O-Notation) an.

a) f(n) = n^6 + 3 n^5 + 4 n^2 + 10
b) f(n) = n^2 + 2 n^2 + 6n
c) f(n) = 3^3 + 2n^2 + sin(n)
d) f(n) = n + n^3 – 3n^2
e) f(n) = n^3 + sin(n) * n^4

Aufgabe 5 – Swap-Sort **(5 BE)**

Gegeben sei ein 1-basiertes Array A der Länge n, welches mit Zahlen gefüllt ist. Das Sortierverfahren Swap-Sort geht zum Sortieren des Arrays folgendermaßen vor:

1. Setze den Index i = 1
2. Setze m als die Anzahl aller Zahlen in A, die kleiner als A(i) sind
3. Tausche A(i) mit A(m+1)
4. Wenn i = m+1: Erhöhe i um 1
5. Wenn i = n, ist der Algorithmus fertig. Ansonsten starte wieder bei Schritt 2.

Bearbeiten Sie die folgenden Aufgaben:

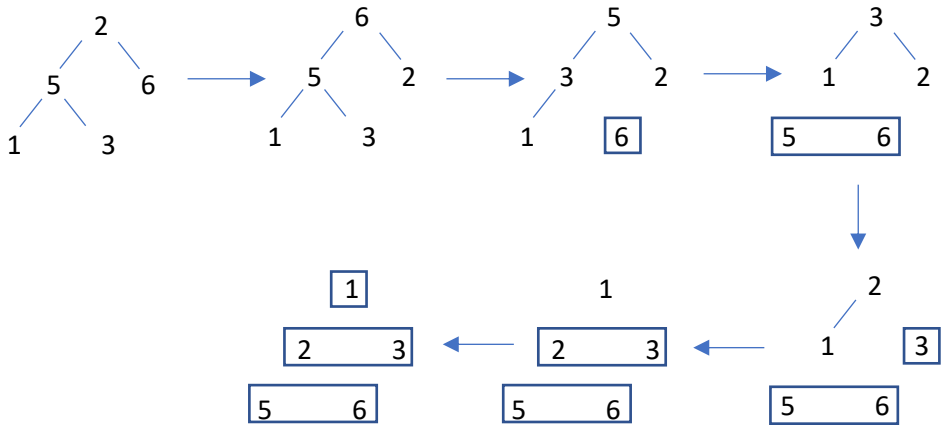
a) Zeichnen Sie ein Struktogramm oder ein Flussdiagramm zu SwapSort. Sie können davon ausgehen, dass Hilfsfunktionen zum Tauschen von Arrayeinträgen sowie zur Bestimmung von m existieren. *(2 BE)*
b) Berechnen Sie die Zeitkomplexität an, die SwapSort bei einem Array der Länge n besitzt. *(1BE)*
c) SwapSort hat ohne Anforderungen an das Basisarray keine Garantie, zu terminieren. Es ist daher nur mit einer Voraussetzung an dieses Array auch wirklich ein Algorithmus.
   Geben Sie ein Array der Länge 5 an, mit welchem das Verfahren nicht terminiert! Nennen Sie außerdem den Grund, weshalb das Verfahren nicht terminiert und geben Sie die entsprechende Voraussetzung an das Array A an, damit SwapSort ein Algorithmus wird! *(3BE)*

Musterlösung:

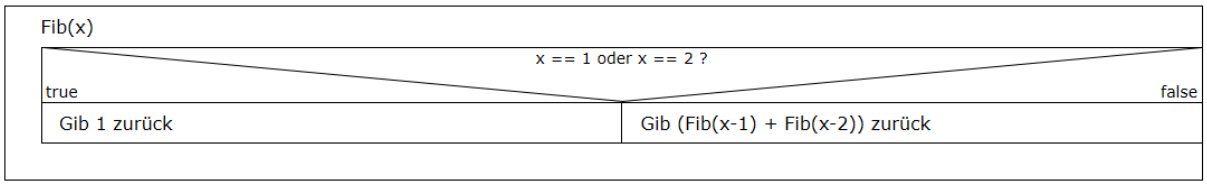
Aufgabe	Musterlösung	Bewertungsmaßstab	AB
1. a)	Ein Algorithmus ist eine eindeutige Handlungsvorschrift, die nach endlich vielen Schritten ein Problem löst.	1 BE für richtige Definition	I
b)	• Finitheit – Verfahren muss in endlichem Text beschreibbar sein • Dynamische Finitheit – zu jedem Zeitpunkt darf nur endlich viel Speicher benötigt werden • Ausführbarkeit – jeder Schritt muss tatsächlich ausführbar sein • Terminierung – das Verfahren darf nur endlich viele Schritte benötigen	Je ½ BE für Nennung und je ½ für kurze Erläuterung	I
c)	In-situ: Der Algorithmus benötigt nur konstanten oder keinen zusätzlichen Speicher. Stabil: Reihenfolge der Datensätze, deren Sortierschlüssel gleich ist, bleiben unverändert bzw. behalten ihre relativen Positionen. Beispiel für beide Eigenschaften: BubbleSort	Je 1 BE für korrekte Definition und je 1 BE für korrektes Beispiel	I
2. a) / b)	Siehe Klausur (für SuS ist nur das Fettgedruckte eingetragen)	Je ½ BE für richtige Zeile Je 1 BE für richtige Zeile	II
c)	Siehe Anhang 1	1 BE für Baum bzw. Heap	II
3. a)	Siehe Anhang 2	2 BE Korrektheit	II
b)	Siehe Anhang 3	2 BE für Idee, 2 BE Korrektheit	III/II
c)	Komplexität - iterativ n (da n Schleifendurchläufe * 1 (konstante Kosten)) Komplexität – rekursiv 2^n (da stetige Unterteilung in 2 Teilprobleme)	Je ½ BE für richtige Komplexität + je ½ BE für Begründung	II
4. a)	O(n^6)	Je 1 BE für richtige Komplexität	I
b)	O(n^2)		I
c)	O(n^2)		I
d)	O(n^3)		I
e)	O(n^4)		I
5. a)	Siehe Anhang 4	2 BE für Korrektheit	II
	n^2, da mindestens n mal eine Zahl mit den n-1 anderen verglichen werden muss(für m)	½ BE für richtige Komplexität + ½ für Begründung	II
	Voraussetzung: Die Zahlen müssen paarweise verschieden sein ➔ Gegenbeispiel: [5, 4, 3, 3, 1]	1 BE für Gegenbeispiel, 1 BE für Formulierung der Voraussetzung, 1 BE für Begründung, warum Algo. sonst nicht läuft	III
	Algorithmus/i hängt sonst an Dopplung		

Anhang:

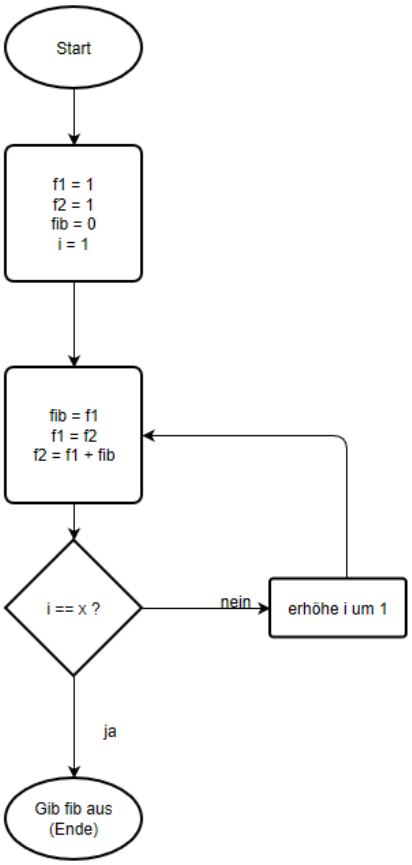
1)


2
5
6
1
3
6
5
2
1
3
5
3
2
1
6
3
1
2
5 6
2
1
3
5 6
1
2 3
5 6
1
2 3
5 6

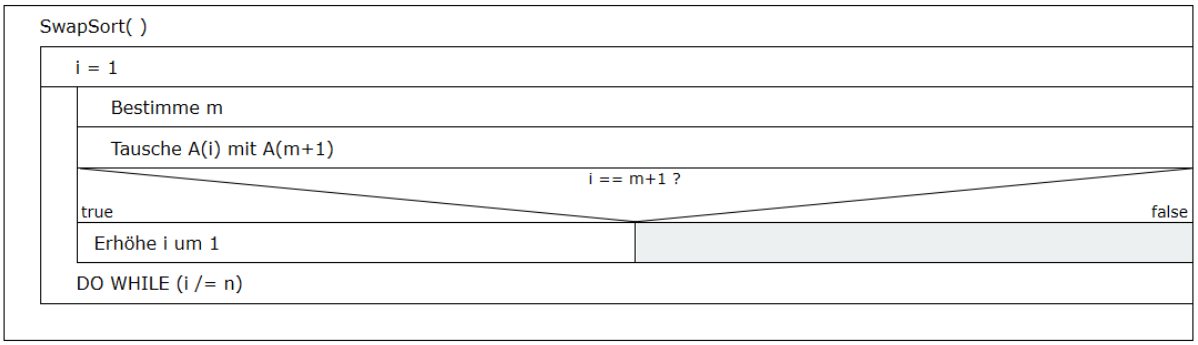
2)

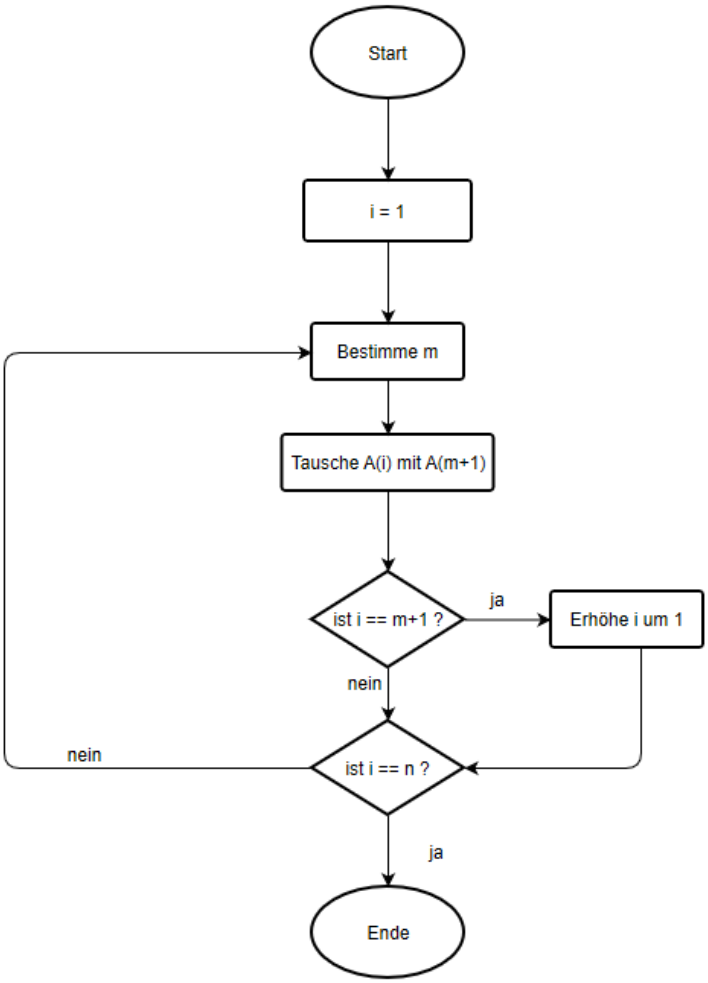

Fib(x)
x == 1 oder x == 2 ?
true
false
Gib 1 zurück
Gib (Fib(x-1) + Fib(x-2)) zurück

3)


Start
f1 = 1
f2 = 1
fib = 0
i = 1
fib = f1
f1 = f2
f2 = f1 + fib
i == x ?
nein
erhöhe i um 1
ja
Gib fib aus
(Ende)

4)


SwapSort( )
i = 1
Bestimme m
Tausche A(i) mit A(m+1)
i == m+1 ?
true
false
Erhöhe i um 1
DO WHILE (i /= n)


Start
i = 1
Bestimme m
Tausche A(i) mit A(m+1)
ist i == m+1 ?
ja
Erhöhe i um 1
nein
nein
ist i == n ?
ja
Ende