

Hochschule RheinMain Sommersemester 2016 Fachbereich DCSM Prof. Dr. Adrian Ulges

Probeklausur zur Veranstaltung Algorithmen und Datenstrukturen 12.06.2016

Name:	Vorname:				
Matrikelnummer:	Unterschrift:				
Mit meiner Unterschrift bestätige ich, dass ich d	ie Anmerkungen unten zur Kenntnis genommen,				
die Aufgaben eigenständig gelöst, sowie nur die zugelassenen Hilfsmittel verwendet habe.					

- Die Klausurdauer beträgt 90 Minuten.
- Bitte legen Sie Studierendenausweis und Lichtbildausweis auf Ihren Tisch.
- Bitte schreiben Sie deutlich. Unleserliche Lösungen werden nicht gewertet. Die Bindung der Blätter dieser Klausur darf nicht entfernt werden. Sie dürfen auch die Rückseiten der Blätter verwenden (weiteres Schmierpapier befindet sich am Ende).
- Lesen Sie die Aufgabenstellungen **vollständig**. Sollten während der Klausur Unklarheiten bestehen, ist es möglich kurze **Fragen** zu stellen.
- Für die Vergabe von Punkten ist generell die Angabe eines Lösungswegs erforderlich.
- Es sind **keine Hilfsmittel** zugelassen. Entfernen Sie Mobiltelefone, Vorlesungsmitschriften, sonstige lose Blätter und Bücher von Ihrem Tisch.
- Täuschungsversuche aller Art werden mit der Note 5 geahndet.
- Beachten Sie insbesondere, dass elektronische Geräte (z.B. Mobiltelefone, Smartwatches oder Kameras) unerlaubte Hilfsmittel sind! Bereits das Berühren eines nicht erlaubten Hilfsmittels während der Prüfung stellt einen Betrugsversuch dar.
- **Toilettengänge** während der Prüfung kosten Ihre Zeit und schaffen für alle Unruhe. Erledigen Sie sie möglichst vor der Prüfung. Wenn es trotzdem sein muss: Es darf immer nur einer gleichzeitig. Melden Sie sich bei der Aufsicht an und warten Sie auf das OK.

Viel Erfolg!

Aufgabe	1	2	3	4	5	6	7	\sum
Erreichte Punkte	15	15	13	15	13	13	X	100
Erreichbare Punkte								
Note								



Aufgabe 1 (3+3+4+3+2 = 15 Punkte)

Gegeben sei der folgende Algorithmus in Pseudo-Code:

```
# Eingabe
# p: Eine ganze Zahl

j = 0
result = 0
while j<p:
    result = result + (2j+1)
    j = j + 1

return result</pre>
```

a) Geben Sie ein Struktogramm des Algorithmus an.

b) Terminiert der Algorithmus für alle Eingabewerte $p \in \mathbb{Z}$? Geben Sie eine Begründung.



c)	Berechnen Sie die Ausgabe des Algorithmus für die Eingabewerte von 0 bis 4. Gegeben Werte $p \geq 0$, welche Funktion f berechnet der Algorithmus? Geben Sie eine Vermutung ab, eine Begründung ist nicht erforderlich.
d)	Schreiben Sie den Algorithmus als rekursive Funktion foo(p) in Pseudo-Code.
	<pre>function foo(p):</pre>
e)	Ergänzen Sie Ihre rekursive Funktion so, dass auch für negative Werte das korrekte Ergebnis $f(p)$ (siehe Aufgabe (c)) zurückgeliefert wird.
	<pre>function foo(p):</pre>



a)	ndung oder (falls nein) ein Gegenbeispiel an.) Jeder Algorithmus mit deterministischem								
	Ablauf ist auch terminierend.								
	Begründung/Gegenbeispiel:								
b)	Alle vergleichsbasierten Sortierverfahren besit-	☐ gilt	☐ gilt nich						
	zen einen Worst-Case-Aufwand von $\Theta(n^2)$.								
	Begründung/Gegenbeispiel:								
c)	Ein Algorithmus der Aufwandsklasse $\Theta(n)$	☐ gilt	☐ gilt nich						
<i>c)</i>	benötigt immer weniger Rechenschritte als ein Algorithmus der Aufwandsklasse $\Theta(n^2)$.	g	_ girt illeii						
	Begründung/Gegenbeispiel:								
d)	Backtracking-Verfahren finden immer die glo-	☐ gilt	☐ gilt nich						
	bal optimale Lösung eines Problems.								
	Begründung/Gegenbeispiel:								
e)	Bei einfach verketteten Listen gehören sämtli-	☐ gilt	☐ gilt nich						
	che Standard-Operationen zur Aufwandsklasse O(1).								



Matrikelnummer: _____

Aufgabe 3 (5+4+4 = 13 Punkte)

a) Geben Sie rechts jeweils eine Funktion an, die <u>beide</u> Bedingungen auf der linken Seite erfüllt. Sollte keine solche Funktion existieren, markieren Sie dies durch einen <u>Strich</u>. *Hinweis: Es ist keine Herleitung erforderlich*.

$$a(n) = \Theta(n^2) \text{ und } a(n) \in O(\frac{4n^6}{3n^3})$$
 $\rightarrow a(n) = \underline{\hspace{1cm}}$

$$b(n) = O(n \cdot log_2(n))$$
 und $b(n) \in \Omega(n^2) \rightarrow b(n) =$

$$log_2(n) \in O(c(n))$$
 und $c(n) \in O(2^n)$ $\rightarrow c(n) =$

$$d(n) = \Theta(1) \text{ und } d(n) \in \Theta(\log(n)) \qquad \rightarrow d(n) = \underline{\hspace{1cm}}$$

$$O(e(n)) \subseteq O(n^{10})$$
 und $e(n) \in \Omega(log(n)) \rightarrow e(n) =$ ______



```
# Eingabe
# feld: Ein array
# n: Die Länge von feld
# start: Eine Position in feld

function sub(feld, n, start):

pos = start
max = feld[pos]

while pos<start+10 and pos<n:
   if feld[pos]>max:
   max = feld[pos]
   pos += 1

return max
```

```
# Eingabe
# feld: Ein array
# n: Die Länge von feld

function super(feld, n):

   s = 0
   pos = 0

   while pos<n:
      s += sub(feld, n, pos)
      pos += 10

return s</pre>
```

b) Gegeben seien zwei Methoden sub() und super() (welche sub() aufruft). Bestimmen Sie zunächst die Worst-Case-Laufzeit von sub(), d.h. zählen Sie die Anzahl der elementaren Operationen. Als elementare Operationen sollen gelten: Arithmetische Operationen, Vergleiche und Feldzugriffe, aber keine Zuweisungen.

c) Bestimmen Sie die **Worst-Case-Aufwandsklasse** von super() in Abhängigkeit von *n*. Hier genügt eine Abschätzung gemäß den Rechenregeln der O-Notation. Es ist kein explizites Zählen von Einzelschritten erforderlich.



Aufgabe 4 (6+3+6 = 15 Punkte)

a) Die erste Zeile der unteren Tabelle stellt ein Array mit 9 Elementen dar. Führen Sie eine <u>absteigende</u> Sortierung mittels InsertionSort durch. Tragen Sie in jeder neuen Zeile das Ergebnis nach einer weiteren Insertion ein.

4	8	6	5	2	4	1	9	2



b) Diese Java-Implementierung realisiert einen absteigenden InsertionSort. Ist sie <u>stabil?</u> Geben Sie eine Begründung. Falls nein, wie müssten Sie die Implementierung ändern um Stabilität zu erreichen?

```
static void sort(int[] a) {
   for(int i=0; i<a.length; ++i) {
      int val = a[i];
      int j = i;
      while(j>0 && a[j-1]<=val) {
        a[j] = a[j-1];
        j--;
      }
      a[j] = val;
   }
}</pre>
```



c) Die erste Zeile der unteren Tabelle stellt ein Array mit 8 Elementen dar. Führen Sie auf dem Array einen Mergesort durch. Sortieren Sie das Array erneut <u>absteigend</u>. Führen Sie in jeder Zeile <u>eine Misch-Operation</u> durch. Markieren Sie hierzu jeweils die beiden Bereiche die gemischt werden.

4	8	6	5	2	4	1	9
	ı	I	I	I	I	I	
		ı	I				
							, ,
	I	I	I		I	I	



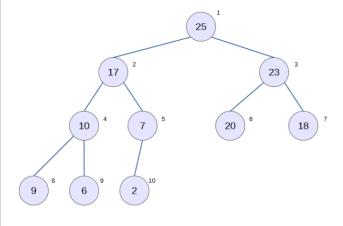
Aufgabe 5 (5+4+4 = 13 Punkte)

```
# Eingabe
# a: Array, das den Heap darstellt
# n: Anzahl der Elemente des Heaps

pos = 1
min = a[1]

while pos <= n:
    l = left(pos)
    r = right(pos)
    if r<=n and a[r]<a[l]:
        pos = r
        min = a[r]
    else if l<=n:
        pos = l
        min = a[l]
    else:
        pos = n+1

return min</pre>
```



Gegeben sei ein Max-Heap wie in der Vorlesung definiert. Wir möchten das **Minimum** des Heaps finden. Hierzu schlägt Alice den obigen Algorithmus (in Pseudo-Code) vor.

<u>Hinweise</u>: Wie in der Vorlesung spezifiziert beginnt der Heap bei Element 1, und die Methoden left() und right() liefern die Positionen der Kindknoten.

- a) Welche Lösung gibt der Algorithmus für den Beispiel-Heap auf der rechten Seite zurück? Schildern Sie kurz den Verlauf des Algorithmus (welche Werte nimmt pos an?).
- b) Welchem der in der Vorlesung vorgestellten Algorithmenmuster entspricht der Algorithmus? Begründen Sie kurz.
- c) Findet der Algorithmus immer das Minimum des Heaps? Falls ja, begründen Sie. Falls nein, geben Sie ein Gegenbeispiel.



Aufgabe 6 (6+7=13 Punkte)

```
class LinkedList<T> {
  private Node head;
  private Node last;

private class Node {
  T obj;
  Node next;
}

...
} head
  last
  obj next
  obj next

obj next

obj next

obj next

"Hallo" "Listen" "Welt"
```

Gegeben sei eine einfach verkettete Liste. Gemäß unserer Spezifikation aus der Vorlesung (oben links) besteht die Liste aus Knoten (Nodes) und besitzt für Anfang und Ende separate Knoten head und last.

a) Implementieren Sie eine private Methode get(int i), die den i-ten Knoten der Liste zurückgibt. Für i=1 soll das 1. Element zurückgeliefert werden, für i=0 der Knoten head, für i=n+1 (für n-elementige Listen) der Knoten last. Für i<0 und i>n+1 soll eine ListException geworfen werden (diese brauchen Sie nicht extra zu definieren).

private Node get(i) throws ListException {

b) Implementieren Sie eine Methode swap(int j, int k), die die Listenelemente an den Positionen j und k vertauscht. Die Zähler j und k beginnen bei 1. Sie können außerdem davon ausgehen, dass j < k.

Hinweis: Verwenden Sie die Methode get () aus der vorherigen Aufgabe.

```
void swap(int j, int k) throws ListExcepti-
on {
```