Algorithmen und Datenstrukturen 1

Abschlussklausur SoSe 2012
10. Juli 2012

Prof. Dr. Carsten Lecon
Fachhochschule Aalen
Fakultät Elektronik und Informatik
Studiengang Informatik

Name:	
Vorname:	
Semester:	
Matrikelnummer:	

- Bitte geben Sie diesen Zettel sowie die übrigen Aufgabenzettel zusammen mit den Lösungsblättern ab.
- Das erste Blatt der abgegebenen Klausur muss dieses **Deckblatt** sein.
- Die Blätter sind nach Aufgaben **aufsteigend sortiert.**
- Die Blätter sind **zusammengeheftet.** Nicht angeheftete Blätter werden nicht gewertet!

Viel Erfolg!

Einträge in der folgenden Tabelle sind nicht vom Prüfling auszufüllen
(dennoch getätigte Angaben werden ggf. korrigiert …).

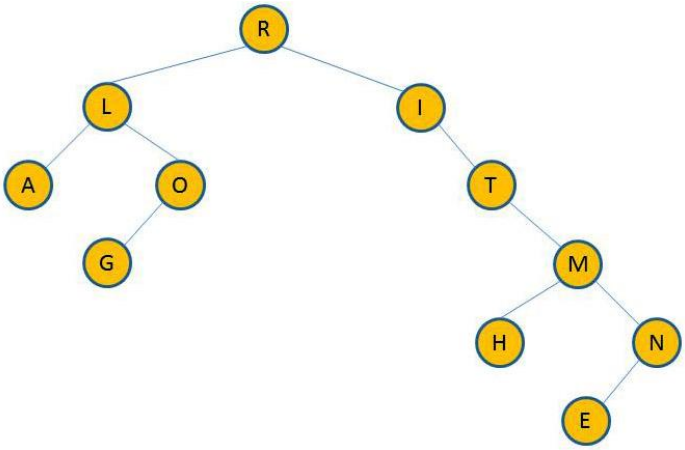
Aufgabe	Thema	Max. Punkte	Erreichte Punkte
1	BST	10	
2	Rekursion	7	
3	Bubblesort	10	
4	Heapsort	20	
5	Shell-/Ins.sort	14	
6	RST	19	
7	Allg. Fragen	12	
8	Quicksort	14	
9	Algorithmik	20	
Punkte aus Übungen			
SUMME		126	

Binäre Suchbäume [10]

a) Fügen Sie die Zahlen 25, 7, 3, 10, 33, 26, 1, 30, 6 in dieser Reihenfolge in einen anfangs leeren binären Suchbaum ein. Zeichnen Sie den Suchbaum, nachdem Sie alle Zahlen eingefügt haben. [4]

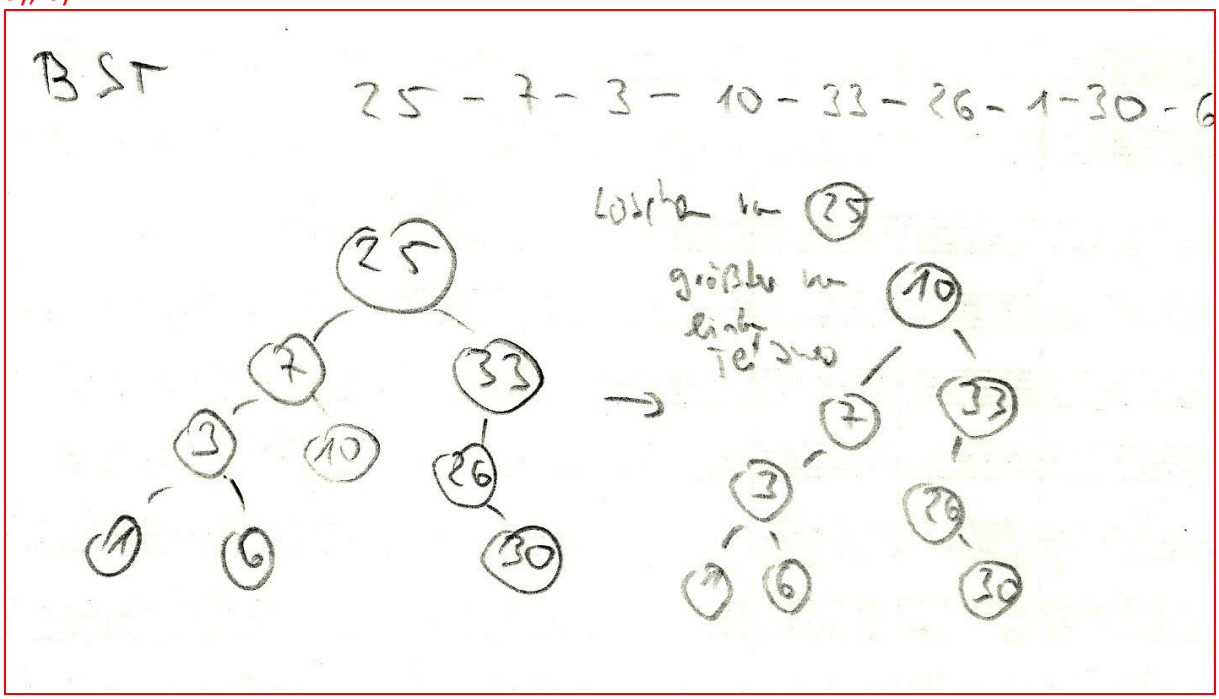
b) Löschen Sie aus dem Suchbaum, den Sie in der obigen Aufgabe erstellt haben, den Wurzelknoten. Zeichnen Sie den resultierenden Suchbaum und beschreiben Sie Ihr Vorgehen. [2]

c) Gegeben sei folgender Binärbaum. Geben Sie die Knotenwerte an, wenn der Baum in *Inorder*-Reihenfolge traversiert wird. [4]

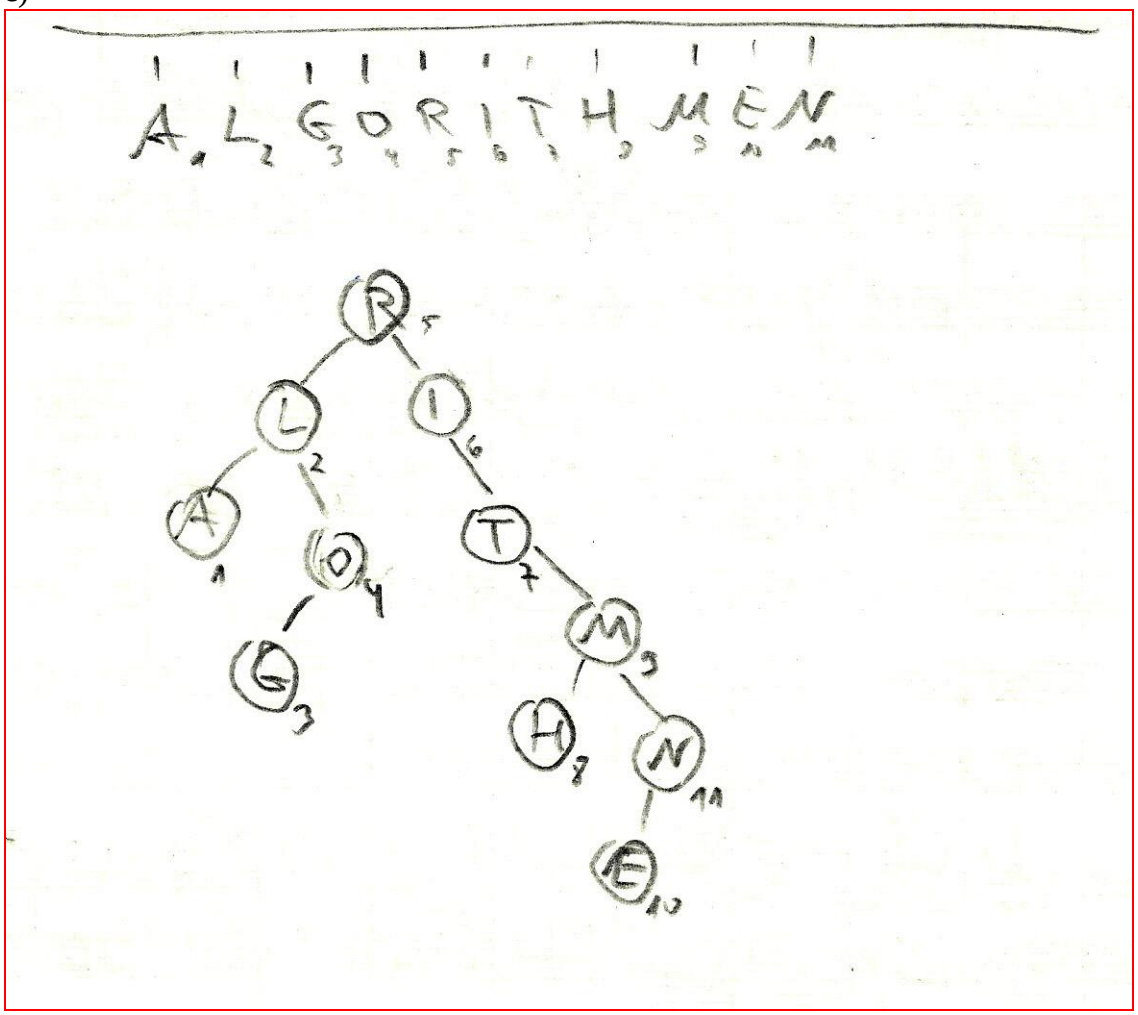


LÖSUNG:

a), b)


BST
25 - 7 - 3 - 10 - 33 - 26 - 1 - 30 - 6
Lösche 25
größter im linken Teilbaum

c)


A L G O R I T H M E N

Rekursion [7]
Gegeben ist die folgende Funktion *f* (Wertebereich: alle natürlichen positiven Zahlen einschließlich 0):

f(n)= 0 für n=0
f(n)= n + f(n-1) sonst.

a) Berechnen Sie die folgenden Funktionswerte: [4]

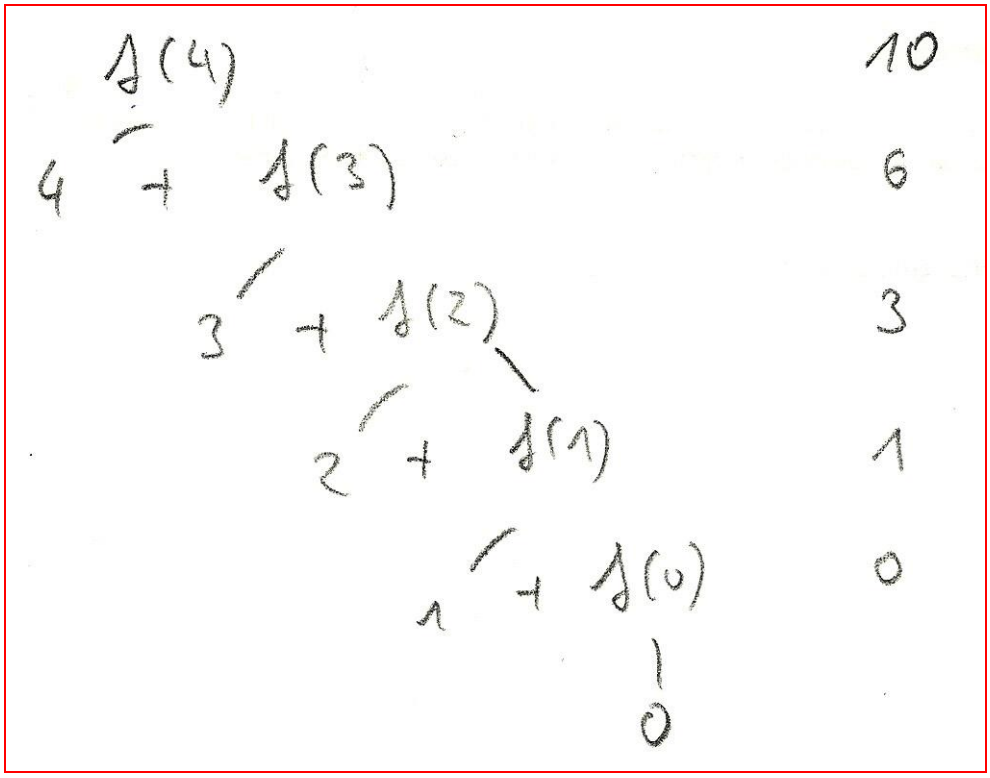
(n)	1	2	3	4	5	6
f(n)						

b) Zeichen Sie den Rekursionsbaum für den Aufruf *f(4)*. [3]

LÖSUNG:

a)

(n)	1	2	3	4	5	6
f(n)	1	3	6	10	15	21



Bubblesort [10]

Gegeben sei die Zahlenfolge {13, 9, 10, 6, 8, 5, 1, 4}.

a) Sortieren Sie diese Zahlenfolge mit dem *Bubblesort*-Algorithmus (2. Variante). Schreiben Sie die Zahlenfolge nach jedem Durchlauf der äußeren Schleife in nachfolgende Tabelle (die Anzahl der Zeilen der Tabelle spiegelt nicht unbedingt die tatsächliche Anzahl von Schleifendurchläufen wider…). [8]

b) Was ist der Vorteil der zweiten Variante von *Bubblesort* gegenüber der ersten? [2]

Durchlauf								
0	13	9	10	6	8	5	1	4
1								
2								

LÖSUNG:

a)

Durchlauf								
0	13	9	10	6	8	5	1	4
1	9	10	6	8	5	1	4	13
2	9	6	8	5	1	4	10	13
3	6	8	5	1	4	9	10	13
4	6	5	1	4	8	9	10	13
5	5	1	4	6	8	9	10	13
6	1	4	5	6	8	9	10	13
7	1	4	5	6	8	9	10	13

b) Es werden nicht alle Zahlen miteinander verglichen, sondern bei einer Vorsortierung kann der Tauschvorgang abgebrochen werden, sobald eine sortierte Zahlenfolge vorliegt.

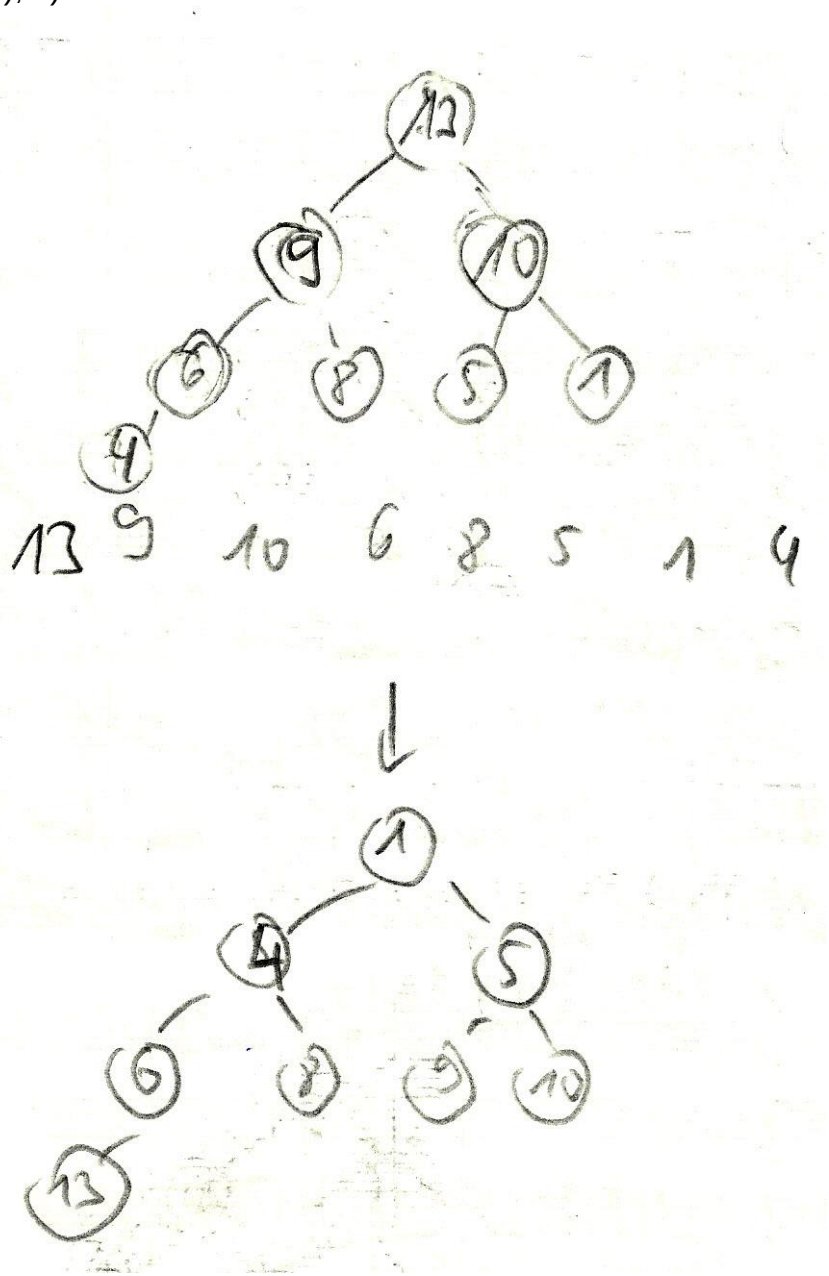
Heapsort [20]

Gegeben sei die Zahlenfolge {13, 9, 10, 6, 8, 5, 1, 4}.

a) Zeichnen Sie den entsprechenden Heap-Baum. [1]
b) Handelt es sich dabei um einen Max- oder einen Min-Heap oder weder noch? Begründen Sie Ihre Antwort. [2]
c) Sortieren Sie die obige Zahlenfolge mittels des Heapsort-Algorithmus. Tragen Sie die Zwischenstände in folgende Tabelle ein. [15]
d) Zeichnen Sie den Heap-Baum nach der Sortierung. [1]
e) Handelt es sich bei dem Heap-Baum nach der Sortierung um einen Max- oder einen Min-Heap oder weder noch? Begründen Sie Ihre Antwort. [1]

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Array-Index**
13	9	10	6	8	5	1	4	Ursprungs-Array
							4	Array nach Herstellung des Max-Heaps
								Array nach Sortierung
								Array nach Wiederherstellung des Max-
								Array nach Sortierung
								Array nach Wiederherstellung des Max-
								Array nach Sortierung
								Array nach Wiederherstellung des Max-
								Array nach Sortierung
								Array nach Wiederherstellung des Max-
								Array nach Sortierung
								Array nach Wiederherstellung des Max-Heaps
								Array nach Sortierung
								Array nach Wiederherstellung des Max-Heaps
								Array nach Sortierung
								Array nach Wiederherstellung des Max-Heaps
								Array nach endgültiger Sortierung

LÖSUNG:
a), d):



c)

1	2	3	4	5	6	7	8	**Array-Index**
13	9	10	6	8	5	1	4	Ursprungs-Array
13	9	19	6	8	5	1	4	Array nach Herstellung des Max-Heaps
4	9	10	6	8	5	1	13	Array nach Sortierung
10	9	5	6	8	4	1		Array nach Wiederherstellung des Max-
1	9	5	6	8	4	10		Array nach Sortierung
9	6	5	1	8	4			Array nach Wiederherstellung des Max-

4	6	5	1	8	9			Array nach Sortierung
8	6	5	1	4				Array nach Wiederherstellung des Max-
4	6	5	1	8				Array nach Sortierung
6	4	5	1					Array nach Wiederherstellung des Max-
1	4	5	6					Array nach Sortierung
5	4	1						Array nach Wiederherstellung des Max-Heaps
1	4	5						Array nach Sortierung
4	1							Array nach Wiederherstellung des Max-
1	4							Array nach Sortierung
1								Array nach Wiederherstellung des Max-
1	4	5	6	8	9	10	13	Array nach endgültiger Sortierung

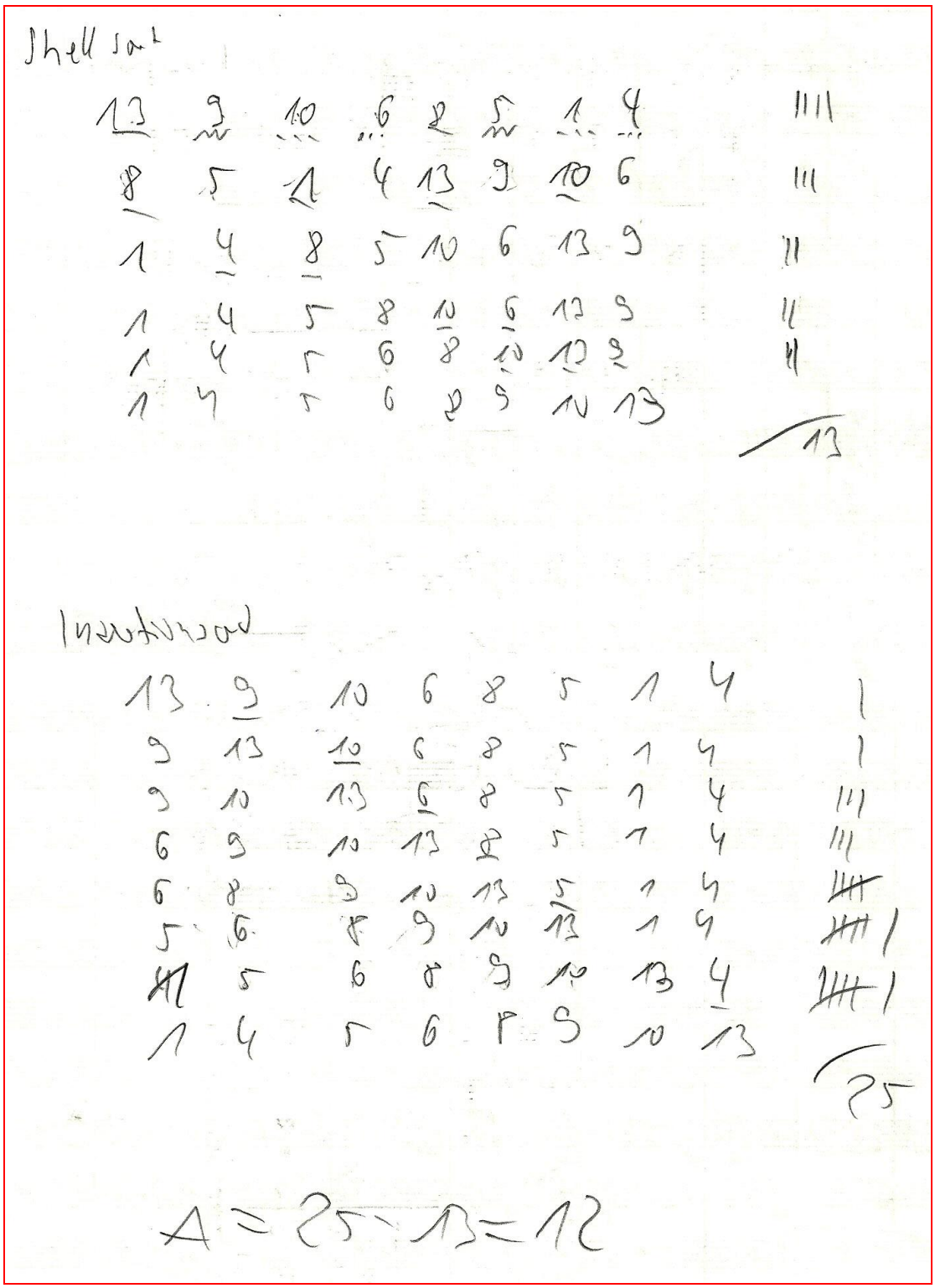
b): Max-Heap, da alle Kindknoten jeweils kleinere Werte als die Eltern aufweisen.
e): Min-Heap, da alle Kindknoten jeweils größere Werte als die Eltern aufweisen.

Shellsort vs. Insertionsort [14]
Gegeben sei die Zahlenfolge {13, 9, 10, 6, 8, 5, 1, 4}.

a) Sortieren Sie diese Zahlenfolge mittels *Shellsort*. Geben Sie die Zwischenschritte (nach Durchlauf der äußeren Schleife) an. [7]
b) Sortieren Sie diese Zahlenfolge mittels *Insertionsort*. Geben Sie die Zwischenschritte (nach Durchlauf der äußeren Schleife) an. [4]
c) Wie viele Vertauschungen sind beim *Insertionsort* im Vergleich zum *Shellsort* mehr erforderlich? [1]
d) Ist der *Shellsort* auch bei vollständig sortieren Arrays effizienter? Begründen Sie Ihre Antwort. [2]

LÖSUNG:

a), b)



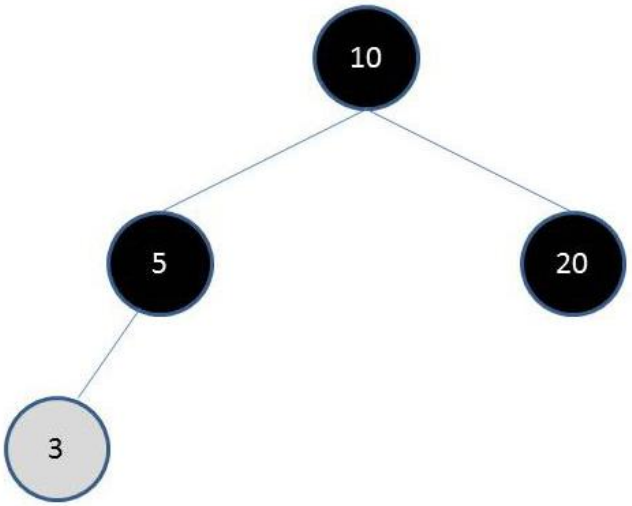
c): 12 Vertauschungen mehr beim Insertionsort gegenüber dem Shellsort.

d) Nein. Neben der Anwendung des „normalen“ Insertionsorts, bei dem keine Vertauschungen erforderlich sind, hat der Shellsort zusätzlichen Aufwand für die Vorsortierung (die hier gerade nicht erforderlich ist).

Rot-Schwarz-Baum [19]
Fügen Sie in folgenden Rot-Schwarz-Baum (hellgraue Knoten sind rote Knoten) nacheinander die sechs Knoten mit den Schlüsseln 6, 7, 2, 4, 1, 0, -1 ein.

Knoten	1b	LR	RR
6			
7			
2			
4			
1			
0			
-1			

Zeichnen Sie jeweils den resultierenden Baum. Geben Sie jeweils den Zustand direkt nach dem Einfügen und ggf. nach einer Wiederherstellung der Rot-Schwarz-Eigenschaft an. Notieren Sie in obiger Tabelle, welche Aktionen beim Einfügen des entsprechenden Knotens ggf. erfolgen. Tragen Sie bei „1b“ die jeweilige Anzahl der Anwendung der Umfärbungen ein, bei LR („Linksrotation“) und RR („Rechtsrotation“) jeweils den Knoten, um den rotiert wird.



LÖSUNG:

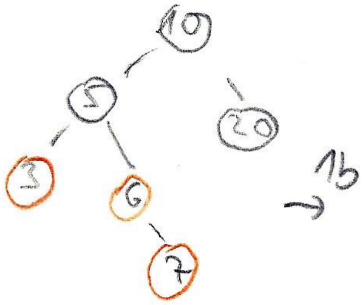
Knoten	1b	LR	RR	Punkte
6				2
7	1			3
2				2
4				2
1	1		10	4
0			2	2
-1	2			4

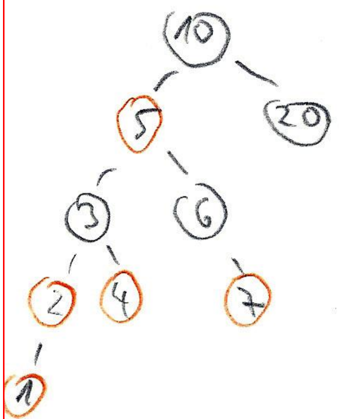
AlDa1 - Klausur SS 2012

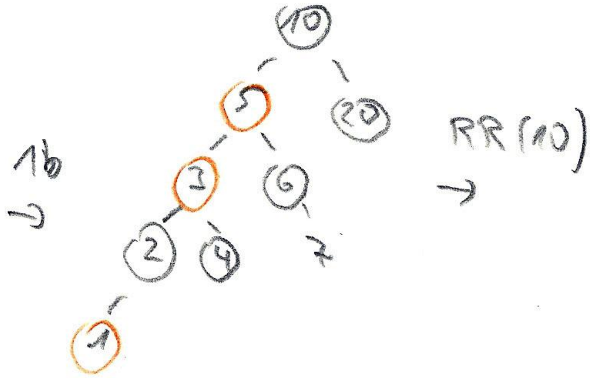
3.7.2012

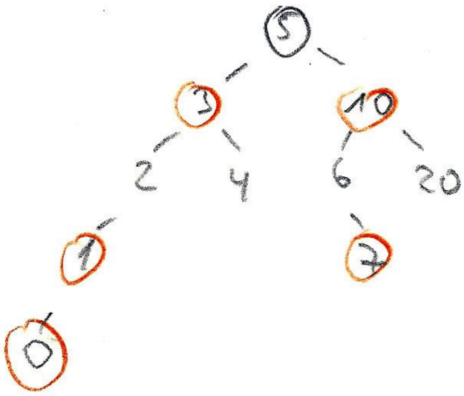
Einfügen 6, 1, 7, 2, 4, 1, 0, -1

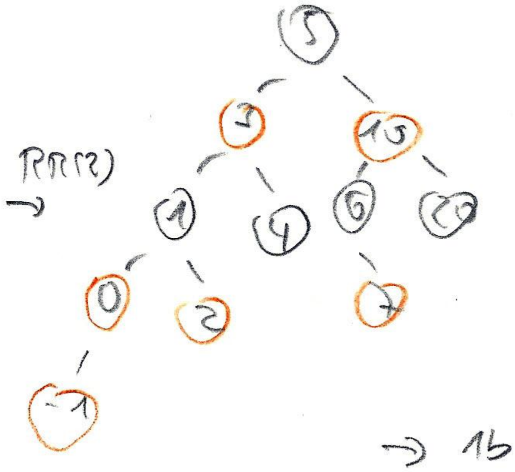
1b ↓ (7) — 1b + RR ↓ (1) — RR ↑ (-1)

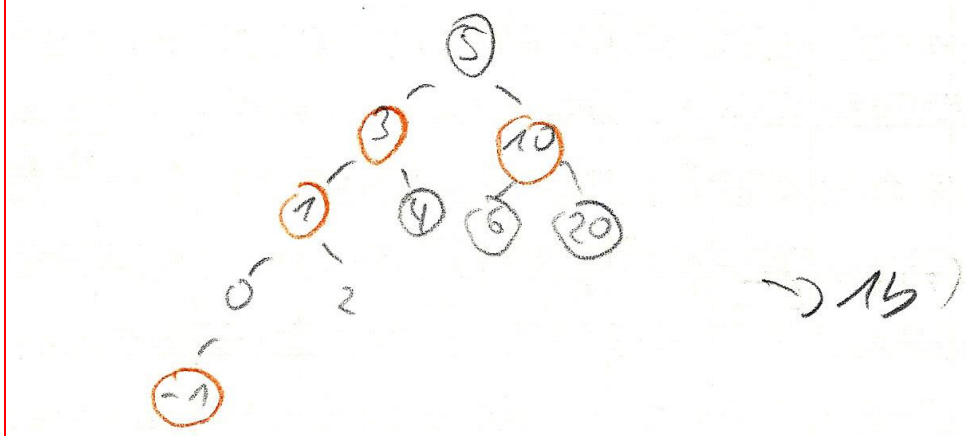




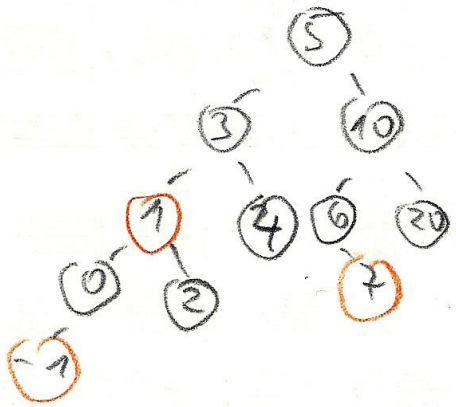








→ 15



Allgemeine Fragen [12]

a. Welche der folgenden Laufzeiten bezeichnet den schnellsten, den zweitschnellsten und den drittschnellsten Algorithmus? [2]
    a. quadratische Laufzeit
    b. kubische Laufzeit
    c. Laufzeit von n log(n)

    Schnellste Laufzeit: ___
    Zweitschnellste Laufzeit: ___
    Drittschnellste Laufzeit: ___

b. Wann ist Mergesort besser geeignet als Quicksort? [2]

c. Erläutern Sie das Prinzip der binären Suche in einem Array. Welche Voraussetzungen müssen erfüllt sein? [2]

d. Ein Zimmermann muss eine 2,05-Meter hohe Decke mit Holzbalken abstützen. Da es keine 2,05-Meter hohen Balken gibt, muss er mehrere Balken aufeinanderstapeln. Zudem möchte er sein Lager aufräumen, also möglichst viele Balken verwenden. Holzbalken mit folgenden Höhen stehen ihm dabei zur Verfügung; wobei sich von jeder Sorte 20 Stück im Lager befinden: [6]

    15 cm
    20 cm
    40 cm
    60 cm
    150 cm

    Der Zimmermann wendet den *Greedy*-Algorithmus an. Welche Holzbalken (und wie viele jeweils davon) wählt er?
    Ist dies die optimale Lösung (Verbauung möglichst vieler Holzbalken)? Falls nicht, wie sieht die optimale Lösung aus?

LÖSUNG:
Welche der folgenden Laufzeiten bezeichnet den schnellsten, den zweitschnellsten und den drittschnellsten Algorithmus? [1]

a. quadratische Laufzeit
b. kubische Laufzeit
c. Laufzeit von n log(n)

Schnellste Laufzeit: c
Zweitschnellste Laufzeit: a
Drittschnellste Laufzeit: b

b.

Wann ist Mergesort besser geeignet als Quicksort? []
Wenn ein stabiler Sortieralgorithmus gefragt ist, also eine bereits bestehende Sortierung beibehalten werden soll.

c. Erläutern Sie das Prinzip der binären Suche in einem Array. Welche Voraussetzungen müssen erfüllt sein? []
In der Mitte beginnen und dann je nach Relation links oder rechts rekursiv weitersuchen. Voraussetzung ist, dass die Werte im Array sortiert sind.

d. Ein Zimmermann muss eine 2,05-Meter hohe Decke mit Holzbalken abstützen. Da es keine 2,05-Meter hohen Balken gibt, muss er mehrere Balken aufeinanderstapeln. Zudem möchte er sein Lager aufräumen, also möglichst viele Balken verwenden. Holzbalken mit folgenden Höhen stehen ihm dabei zur Verfügung; wobei sich von jeder Sorte 20 Stück im Lager befinden:

15 cm
20 cm
40 cm
60 cm
150 cm

Der Zimmermann wendet den *Greedy*-Algorithmus an. Welche Holzbalken (und wie viele jeweils davon) wählt er?
Ist dies die optimale Lösung (Verbauung möglichst vieler Holzbalken)? Falls nicht, wie sieht die optimale Lösung aus?
→ Greedy: 1*150+1*40+1*15 = 205 (→ 3 Teile)
→ Nicht optimal, optimal:
Vorüberlegung: Es muss auf jeden Fall 1*15 genommen werden, da man sonst nicht auf 205 kommen kann, Rest: 190. Hier „inverses“ Greedy: 10*15 + 2*20 = 190. Zusammen: 11*15 + 2*20 (→ 13 Teile)

Quicksort [14]

Gegeben sei folgender Java-Code für einen *Quicksort*-Algorithmus („Quicksort3“):

```
public static void quicksort3(int left, int right) {
     if (left >= right) return;
     int pivot = left;
     int i = left+1;
     int j = right;
     while (i <= j) {
          while (A[i] < A[pivot]) i++;
          while (A[j] > A[pivot]) j--;
          if (i<=j) {
               swap(i, j);
               i++;
               j--;
          } else swap(j, pivot);
     }
     quicksort3(___, ___);
     quicksort3(___,___);
}
```

Das Array *int A[]* sowie die *swap*-Methode seien bereits implementiert.

a) Vervollständigen Sie den Code, indem Sie die vier Argumente der beiden rekursiven Aufrufe (vorletzte und vorvorletzte Zeile) ergänzen. [4]
b) Sortieren Sie die Zahlenfolge {5, 3, 7, 2, 4, 1} mit diesem „Quicksort3“-Algorithmus. Notieren Sie alle Aufrufe (zum Beispiel „QS3(0,5)“, …). [10]

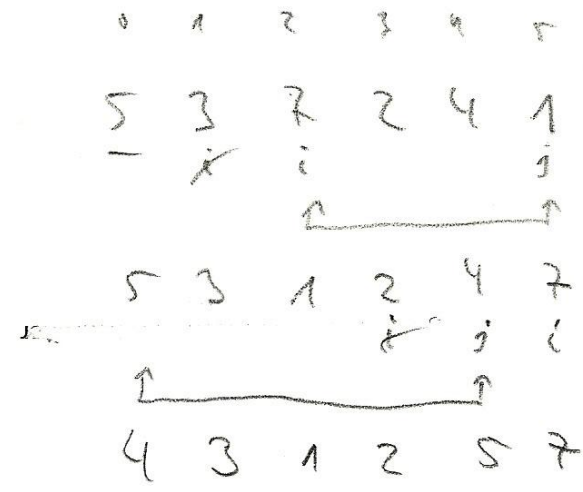
Lösung Quicksort3

```
     quicksort3(left, j);
     quicksort3(i, right);
```

```
qs3(0,5):  5 3 7 2 4 1
qs3(0,4):  4 3 1 2 5
qs3(0,3):  2 3 1 4
qs3(0,1):  2 1
qs3(0,1):  1 2
qs3(0,0):  1
qs3(1,1):  2
qs3(2,1):
qs3(2,3):  3 4
qs3(2,2):  3
qs3(3,3):  4
qs3(4,4):  5
qs3(5,5):  7

Nach Sortierung:
1 2 3 4 5 7
```

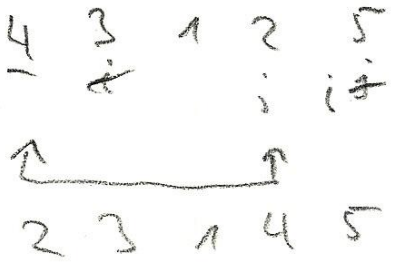
27.2.2012



QS4(0,5)

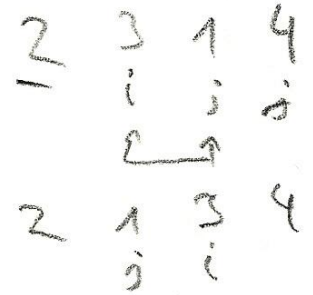
QS4(0,4) - QS4(5,5)

QS(0,4)



QS4(0,3) - QS4(5,5)

QS(0,3)



QS(0,1) - QS(2,3)

QS4(0,1)

→ QS4(0,1) — QS4(2,1)

QS(0,1)

QS4(0,0) - QS4(1,1)

QS4(2,1)

→ QS4(0,0) - QS4(1,1)

In Anlehnung an www.educ.ethz.ch/unt/um/inf/ad/quick/Brunner_Quicksort.pdf

Algorithmik [20]
Entwickeln Sie einen Algorithmus für die Bestimmung der maximalen Teilsumme einer Folge von Zahlen:

- **Eingabe**: Folge X von ganzen Zahlen.
- **Ausgabe**: Maximale Summe einer zusammenhängenden Teilfolge von X (optional Ausgabe der entsprechenden Array-Indexe).

Beispiel:

Array-Index	0	1	2	3	4	5	6	7	8	9
X	29	-39	59	26	-53	58	97	-93	-25	86

Lösung: 187, Indexe 2 bis 6.
Dies ist die Folge X[2] bis X[6], das heißt die Summe 59+26-53+58+97=187

Entwerfen Sie für dieses Problem einen entsprechenden Algorithmus und implementieren Sie ihn in Java.

a) Schreiben Sie eine entsprechende Java-Methode; Rückgabewert dieser Methode ist die maximale Summe, optional können auch die entsprechenden Indexe (im obigen Beispiel 2 und 6) ausgegeben werden.
b) Wie ist die Laufzeit Ihres Algorithmus?

LÖSUNG:

```
public static int resultU, resultO;

public static int maxSum(int A[]) {
     int u; // untere Grenze
     int o; // obere Grenze
     int result=0; // maximale Summe
     int sum; // Summe
     for (u=0; u<A.length; u++) {
          for (o=u; o<A.length; o++) {
               sum = 0;
               for (int i=u; i<=o; i++) {
                    sum += A[i];
               } // for (i)
               if (sum > result) {
                    result = sum;
                    resultU = u;
                    resultO = o;
               } // if
          } // for (o)
     } // for (u)
     return result;
} // maxSum
```

Laufzeit $O(n^3)$

Alternative (effizienter):

```
public static int maxSumBesser(int A[]) {
     int result = 0;
     int u; // untere Grenze
     int o; // obere Grenze
     int sum; // Summe
     for (u=0; u<A.length; u++) {
          sum = 0;
          for (o=u; o<A.length; o++) {
               sum += A[o];
               if (sum > result) {
                    result = sum;
                    resultU = u;
                    resultO = o;
               } // if
          } // for (o)
     } // for (u)
     return result;
} // maxSumBesser
```

Laufzeit: $O(n^2)$

(In Anlehung an http://cg.informatik.uni-freiburg.de/course_notes/info2_06_teile_und_herrsche.pdf)