Algorithmen und Datenstrukturen 1
WS 2010/2011

1. Rekursion

Gegeben ist die folgende Funktion f:

f(n)=f(n-1) + f(n-2) – f(n-3), für n>3,

f(n)=n für n≤ 3.

a) Berechnen Sie die fehlenden Funktionswerte:

n	1	2	3	4	5	6	7	8	9	10
f(n)										

b) Schreiben Sie eine rekursive Java-Methode int f(n), die den Funktionswert von f(n) berechnet.

c) Stellen Sie die Berechnung des Aufrufs von f(6) grafisch dar.

LÖSUNG:

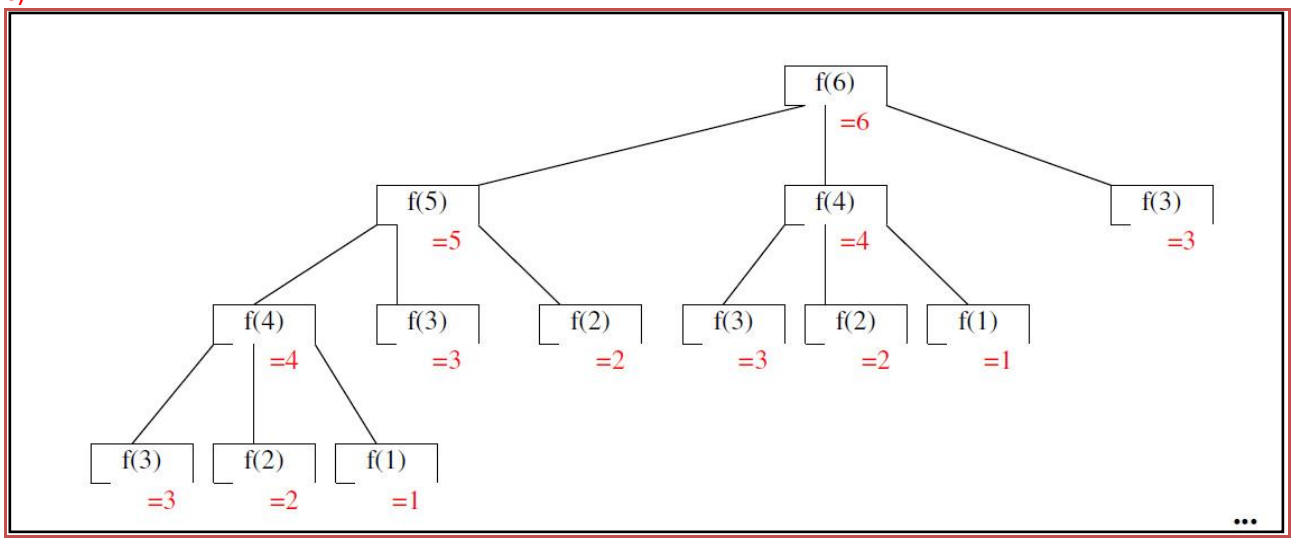
a)

n	1	2	3	4	5	6	7	8	9	10
f(n)	1	2	3	4	5	6	7	8	9	10

b)

```
public int f(int n) {
      if (n<=3) {
            return n;
      } else {
            return (f(n-1) + f(n-2) - f(n-3));
      } // if (else)
} //
```

c)



d) Schreiben Sie eine rekursive Java-Methode

```
boolean durchDreiTeilbar(int x)
```

die prüft, ob eine positive ganze Zahl `x` durch drei teilbar ist.

LÖSUNG:

```
      public boolean teilbarDurchDrei(int x) {
            // Aus http://www.java-forum.org/java-basics-anfaenger-themen/94307-
rekursion-rueckgabe.html
            if (x >= 0 && x <= 2) {
              return x == 0;
          }
          if (x < 0) {
              return teilbarDurchDrei(0 - x);
          }
          return teilbarDurchDrei(x - 3);
      } // teilbarDurchDrei
```

2. Queue / Liste

a) Implementieren Sie in Java eine Queue mit Hilfe einer einfach verketteten Liste. Es sollen folgende Methoden implementiert werden:

```
void enqueue(int data)
ListElement dequeue()
```

Beide Operationen sollen eine Laufzeit von O(1) besitzen. Eine Klasse `MyList` (vgl. Übungsblatt 9) stehe zur Verfügung. Dieses Klasse besitzt u.a. folgende Methoden:

```
public void insertListTail(int data);
// Anfügen eines Elements ans Ende der Liste
public void insertListHead(int data);
// Anfügen eines Elements an den Anfang der Liste
public ListElement deleteHead();
// Löschen des letzten Elements der Liste; das gelöschte Element wird übergeben
public ListElement deleteTail();
// Löschen des letzten Elements der Liste; das gelöschte Element wird übergeben
```

Die Klasse `ListElement` sieht wie folgt aus:

```
public class ListElement {
	int data;
	ListElement next;

	public ListElement(int data) {
		this.data = data;
		next = null;
	} // Konstruktor

	public ListElement(int data, ListElement next) {
		this.data = data;
		this.next = next;
	} // Konstruktor

	public int getData() {
		return data;
	} // getData
}
```

b) Schreiben Sie den Java-Code für die Klasse MyList eine Methode zur Ausgabe aller Elemente der Liste.

c) Stellen Sie die Datenstruktur (Liste) mit den Elementen 4, 5 und 7 als einfach verkettete Liste grafisch dar. Fügen Sie der Liste ein Element mit dem Schlüssel 6 hinzu. Das Element soll vor dem Element mit dem Schlüssel 7 eingefügt werden. Stellen Sie dies durch Veränderung der Grafik dar (andere Farbe o.ä.).

LÖSUNG:

```
public void enqueue(int data) {
	queue.insertListTail(data);
} // enqueue

public int dequeue() {
	ListElement x = queue.deleteHead();
	if (x != null) {
		return x.data;
	} else {
		return -1;
	} // if (else)
} // dequeue
```

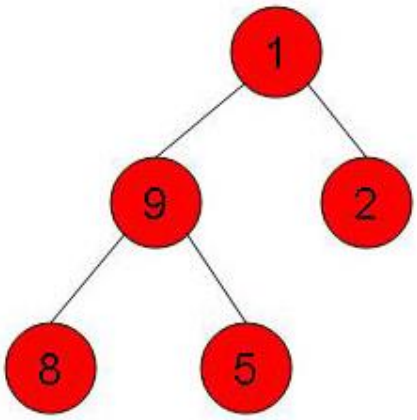
b)

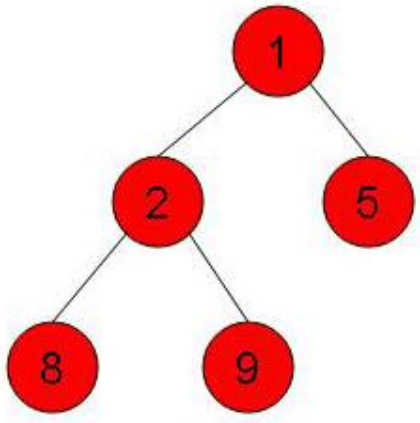
```
public void output() {
	ListElement x = head;
	int i=0;
	while (x != null) {
		i++;
		System.out.println("Liste, Element " + i + ": " + x.data);
		x = x.next;
	} // while
} // output
```

4. **Sortieren**

a) Sortieren Sie folgende Zahlenfolge (1 – 9 – 2 – 8 – 5) aufsteigend mit dem Heapsort-Algorithmus. Tragen Sie die Zwischenstände in folgende Tabelle ein.
Zeichnen Sie zudem den entsprechenden Heap-Baum vor und nach der Sortierung.

1	**2**	**3**	**4**	**5**	**Array-Index**
1	9	2	8	5	Ursprungs-Array
9	1	2	8	5	Array nach Herstellung des Max-Heaps
5	1	2	8	9	Array nach Sortierung der ersten Zahl
8	5	2	1	9	Array nach Wiederherstellung des Max-Heaps
1	5	2	8	9	Array nach Sortierung der zweiten Zahl
5	1	2	8	9	Array nach Wiederherstellung des Max-Heaps
2	1	5	8	8	Array nach Sortierung der dritten Zahl
2	1	5	8	9	Array nach Wiederherstellung des Max-Heaps
1	2	5	8	9	Array nach Sortierung der vierten Zahl
1	2	5	8	9	Array nach Wiederherstellung des Max-Heaps
1	2	5	8	9	Array nach Sortierung der fünften Zahl





b) Sortieren Sie die gleiche Zahlenfolge mit dem Quicksort-Algorithmus. Schreiben Sie jeden Aufruf von Quicksort(A, left, right) auf und die dabei aktuelle Zeichenfolge. Sie können hierzu die folgende Tabelle verwenden.

c) Sortieren Sie die Zahlenfolge {9, 8, 7, 4, 5, 6, 3, 2, 1} mit dem Insertionsort-Algorithmus. Schreiben Sie die Zahlenfolge nach jedem Schleifendurchlauf (äußere Schleife).
LÖSUNG: Siehe Zettel vom 30.01.2011.

9	8	7	4	5	6	3	2	1

8	9	7	4	5	6	3	2	1
7	8	9	4	5	6	3	2	1
4	7	8	9	5	6	3	2	1
4	5	7	8	9	6	3	2	1
4	5	6	7	8	9	3	2	1
3	4	5	6	7	8	9	2	1
2	3	4	5	6	7	8	9	1
1	2	3	4	5	6	7	8	9

d)

e) Sortieren Sie die Zahlenfolge aus c) mit dem **Mergesort**-Algorithmus. Zeichnen Sie den Rekursionsbaum.
LÖSUNG:

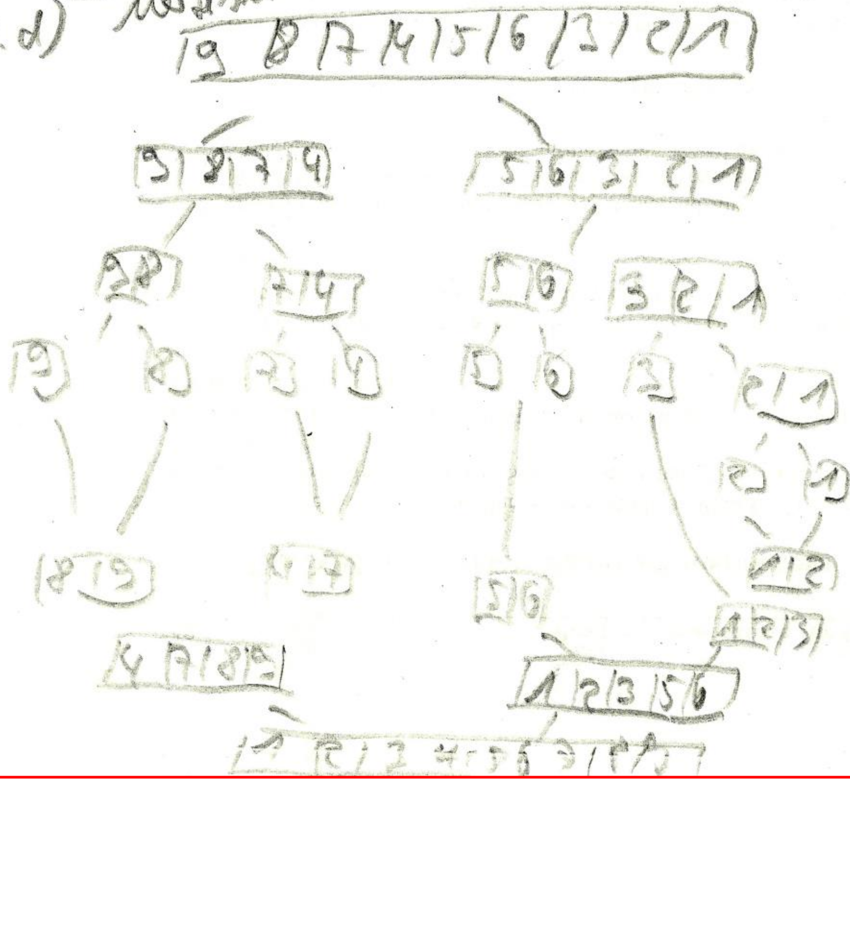
4. c

9	8	7	4	5	6	3	2	1
8	9	7	4	5	6	3	2	1
7	8	9	4	5	6	3	2	1
4	7	8	9	5	6	3	2	1
4	5	7	8	9	6	3	2	1
4	5	6	7	8	9	3	2	1
3	4	5	6	7	8	9	2	1
2	3	4	5	6	7	8	9	1
1	2	3	4	5	6	7	8	9

24
3Um

R : 20

4.d)

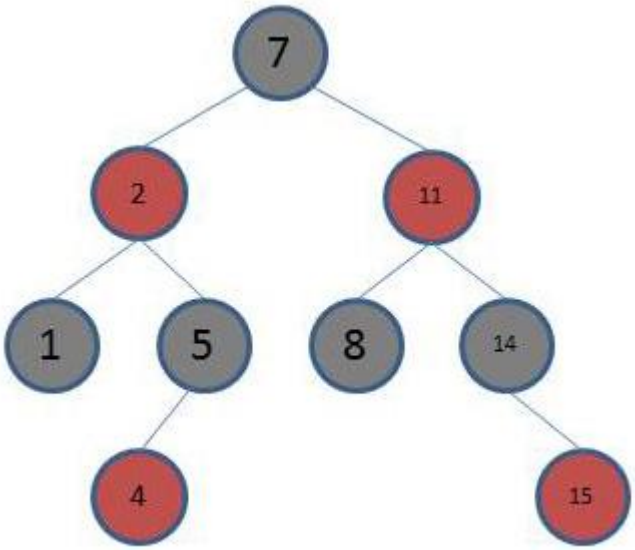


3:00

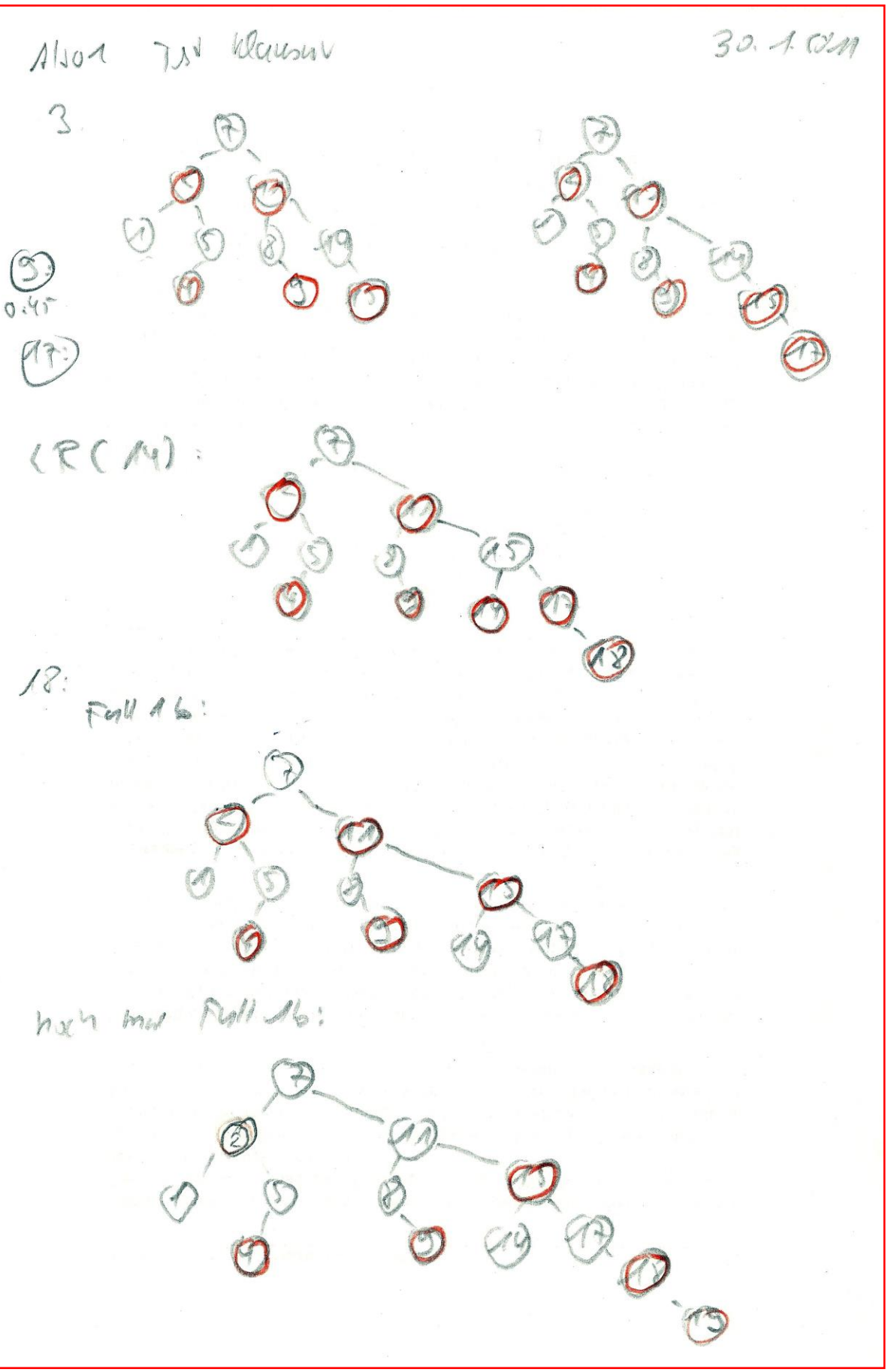
5:10
+ 5:33
+ 2:20
+ 3:00

3. Rot-Schwarz-Baum

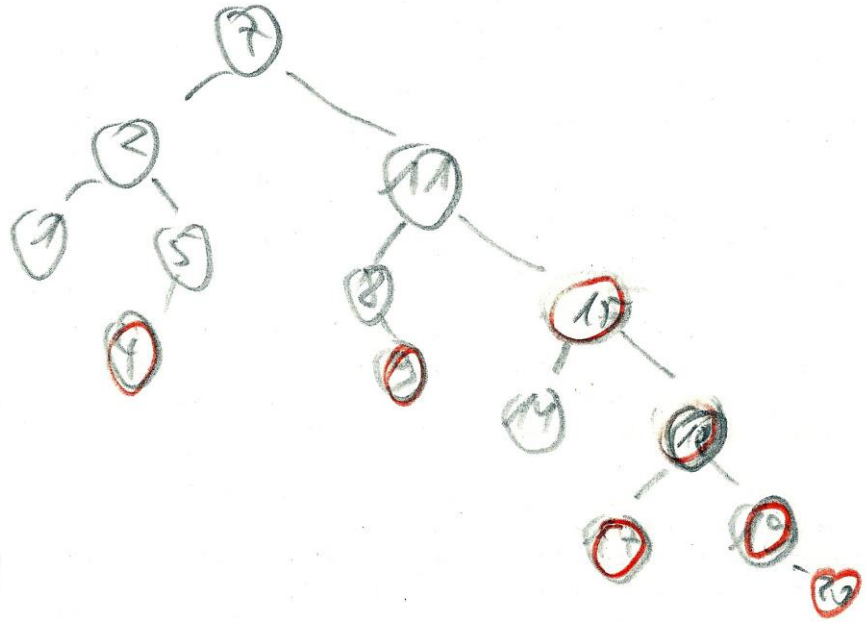
Fügen Sie in folgendem Rot-Schwarz-Baum nacheinander Knoten mit den Schlüsseln 9, 17, 18, 19 und 20 ein. Zeichnen Sie jeweils den resultierenden Baum. Geben Sie jeweils den Zustand direkt nach dem Einfügen und ggf. nach einer Wiederherstellung der Rot-Schwarz-Eigenschaft an. Beschreiben Sie, welche Baum-Operationen Sie ggf. durchführen.



Lösung



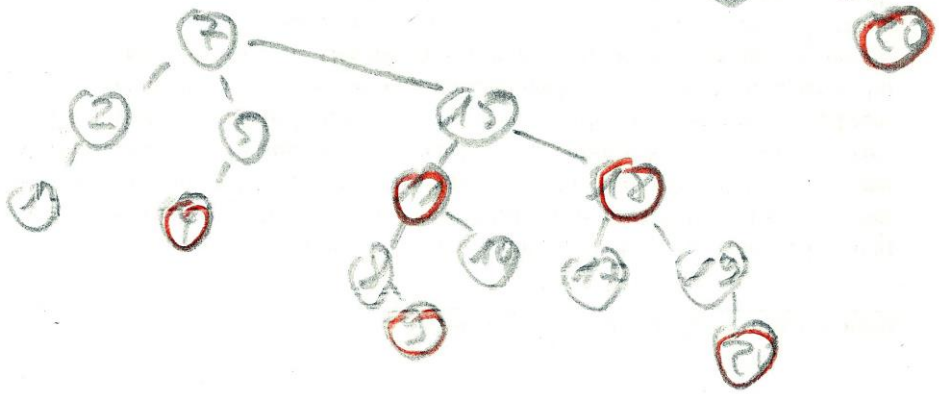
15: (R(14)) :



(...)

Fall 13:

(R(11))



Transfer-Aufgabe: Quicksort

a) Der Quicksort-Algorithmus teilt das zu sortierende Feld in zwei Teile: Ein Teil der alle Elemente ≤ dem Pivot-Element enthält und ein Teil der alle Elemente ≥ dem Pivot-Element enthält. Diese Teile werden wiederum rekursiv mit Quicksort sortiert. *Das Pivot-Element muss dabei nicht notwendigerweise ein Element des zu sortierenden Feldes sein.*

Programmieren Sie eine Quicksort-Variante, die als Pivot-Element stets das ganzzahlige arithmetische Mittel aus dem ersten und dem letzten Element verwendet.

Beispiel: In der Zahlenfolge

25 4 5 12 49 37 5 121

wäre das Pivot-Element das arithmetische Mittel aus 25 und 121, also73 (man beachte, dass diese Zahl nicht in der ursprünglichen Zahlenfolge auftritt).

Schreiben Sie eine Java-Methode

`void sort(int[] array, int from, int to)`,

die das Integerfeld `array` vom linken Index `from` bis zum rechten Index `to` mittels dieser Quicksort-Variante sortiert.

Geben Sie einen Pseudocode oder eine Java-Implementierung an.

b) Wenden Sie diesen Algorithmus auf die obige Beispielfolge (25 … 121) an, geben Sie die Zeichenfolge nach dem zweiten Aufruf an.

Lösung

a) Ist sogar einfacher zu implementieren als das "reguläre" Quicksort, da man sich hier keine Gedanken um das richtige Platzieren des Pivot-Elementes nach Aufteilen des Feldes machen muss und sich so das Abfangen einiger Sonderfälle spart:

```
public class Quick {
public static void sort(int[] array, int from, int to) {
if (from < to) {
int pivValue = (array[from] + array[to]) / 2;
int up = from;
int down = to;
while (true) {
while (up < to && array[up] <= pivValue)
up++;
while (down > up && array[down] > pivValue)
down--;
if (up >= down) break;
Quick.swap(array,up, down);
}
sort(array,from,up-1);
sort(array,up, to);
}
}
}
```

Java-Code:

```
public void quicksort3(int left, int right) {
	int x, i, j;
	if (right <= left) {
		return;
	} // if
	x = (A[left] + A[right]) / 2;
	i = left;
	j = right;
	while (true) {
		while ((i < right) && (A[i] <= x)) {
			i++;
		} // while
		while ((j > i) && (A[j] > x)) {
			j--;
		} // while
		if (i >= j) {
			break;
		} // if
		swap(i, j);
	} // while
	quicksort3(left, i - 1);
	quicksort3(i, right);
} // quicksort3
```

b)

5 4 5 12 49 37 25 121

c) Begründen Sie, weshalb diese Variante für sortierte Zahlenfolgen schneller ist als die in der Vorlesung vorgestellte Quicksort-Variante.

LÖSUNG: Es finden kaum Vergleiche statt. Die Partitionen sind etwas gleich groß.

d) Wie ist die Laufzeit von Quicksort im schlimmsten Fall? Wann tritt der schlimmste Fall ein?

LÖSUNG: $O(n^2)$; Schlimmster Fall: Partionierung des Arrays ein Teilarray mit n-1 Elementen und ein Teilarray mit einem Element.

Algorithmus

Transfer (http://algo2.iti.kit.edu/documents/AlgorithmenI_SS09/nachklausur.pdf)

Gegeben sein ein Array A={A1, …, An} mit n Zahlen in beliebiger Reihenfolge. Für ein gegebenes x soll ein Paar (A[i], A[j]), 1 ≤ i, j ≤ n, gefunden werden, für das gilt A[i]+A[j]=x.

a) Geben Sie eine Lösung für x=29 und A={1, 7, 21, 3, 22, 9, 11} an.

b) Geben Sie einen effizienten Algorithmus an, der das Problem in O(n log n) löst und das Paar (A[i], A[j]) ausgibt, sofern eine Lösung existiert, sonst NIL.
(Bei einem Algorithmus mit quadratischer Komplexität gibt es maximal die Hälfte der Punkte.)

LÖSUNG:

a) A[2] = 15, A[5] = 18.

b)

```
sort(A)                                          // sortiere Feld aufsteigend
i=1                                              // linker Rand
j=n                                              // rechter Rand
while i ≤ j do
    if A[i]+A[j] < x then i++                    // Summe größer machen
    else
        if A[i]+A[j] > x then j--                // Summe kleiner machen
        else return (A[i],A[j])                  // Lösung gefunden
return NIL
```

Eine alternative Lösung mit Hashing läuft sogar in linearer Zeit: Im ersten Schritt alle Elemente in die Hash-Tabelle einfügen, im zweiten Schritt zu jeder Zahl das "Gegenstück" suchen.