Vorname	Name	Matr.-Nr.

Aufgabe 1 O-Notation (2+3+5=10 Punkte)

Zeigen oder widerlegen Sie die folgenden Aussagen:

a) $n^2 + log\ n \in \Theta(n^2)$

Lösung:

Ist die Aussage wahr, so muss gelten: $0 < \lim_{n\to+\infty} \frac{n^2+\log n}{n^2} < \infty$.

$$\lim_{n\to+\infty} \frac{n^2+\log n}{n^2} = \lim_{n\to+\infty} \frac{2n+\frac{1}{n}}{2n} = \lim_{n\to+\infty} (1+\frac{1}{2n^2}) = 1$$

q.e.d.

b) $n^3 \in O(2^n)$

Lösung:

Nach Definition gilt für $n^3 \in O(2^n)$:
$\exists c > 0 \in \mathbb{R}, n_0 \in \mathbb{N}$ mit $n^3 > 2^n \forall n \geq n_0$. Sei $c = 1$:

$$\begin{array}{lrcl} & n^3 & \leq & 2^n \\ \Leftrightarrow & ld\ n^3 & \leq & ld\ 2^n \\ \Leftrightarrow & 3 \cdot ld\ n & \leq & n \end{array}$$

Für $n \geq 16$ gilt $3 \cdot ld\ n \leq n$. q.e.d.

Vorname	Name	Matr.-Nr.

c) $ln\ n! \in \Theta(ln\ n^n)$

Lösung:

Wir zeigen, dass die folgenden beiden Aussagen gelten:

$$ln\ n! \in O(ln\ n^n)\ (1)$$

$$ln\ n^n \in O(ln\ n!)\ (2)$$

(1)

$$\ln n! = \ln(1 \cdot 2 \cdot \dots \cdot n) \leq \ln(\overbrace{n \cdot n \cdot \dots \cdot n}^{n-mal}) = \ln n^n$$

somit gilt: $\ln n! \in O(\ln n^n)$.

(2)

Es gilt:

$$n! = 1 \cdot 2 \cdot \dots \cdot \lceil \frac{n}{2} \rceil \cdot \dots \cdot n \geq \overbrace{\lceil \frac{n}{2} \rceil \cdot \dots \cdot \lceil \frac{n}{2} \rceil}^{\lfloor \frac{n}{2} \rfloor - mal}$$

Wegen $(\lceil \frac{n}{2} \rceil)^{\lfloor \frac{n}{2} \rfloor} \geq (\frac{n}{2})^{\frac{n}{2}-1}$ reicht es zu zeigen, dass $\ln(n^n) \in O(\ln((\frac{n}{2})^{\frac{n}{2}-1}))$ gilt.

$$\begin{aligned}
\lim_{n\to+\infty} \frac{\ln(n^n)}{\ln((\frac{n}{2})^{\frac{n}{2}-1})} &= \lim_{n\to+\infty} \frac{n \cdot \ln n}{(\frac{n}{2}-1) \cdot \ln \frac{n}{2}} \\
&= \lim_{n\to+\infty} \frac{n \cdot \ln n}{(\frac{n}{2}-1) \cdot (\ln n - \ln 2)} \\
&= \lim_{n\to+\infty} \frac{\ln n}{(\frac{1}{2}-\frac{1}{n}) \cdot (\ln n - \ln 2)} \\
&= \lim_{n\to+\infty} \frac{1}{(\frac{1}{2}-\frac{1}{n}) \cdot (1 - \frac{\ln 2}{\ln n})} \\
&= \frac{1}{(\frac{1}{2}-0) \cdot (1-0)} \\
&= 2
\end{aligned}$$

Vorname	Name	Matr.-Nr.

Aufgabe 2 Sortieren (4+2+4 = 10 Punkte)

In dieser Aufgabe betrachten wir einen Sortieralgorithmus für einfach verkettete Listen.

Die Methode `void append(List list, int x)` fügt in konstanter Zeit den Wert `x` an das Ende der Liste `list` ein.

Weiterhin ist die von Mergesort bekannte Methode `List merge(List list1, List list2)` gegeben, die zwei sortierte Listen `list1`, `list2` in linearer Zeit zu einer sortierten Liste zusammenführt.

Betrachten Sie nun folgenden Sortieralgorithmus:

```
1  List strandSort(List list){
2    if (list.head == null || list.head.next == null) return list;
3    (rest, subList) = split(list);
4    List sortedList = merge(strandSort(rest), subList);
5    return sortedList;
6  }
7
8  (List, List) split(List list){
9    List rest, subList;
10   int value = 0;
11   Element pos = list.head;
12   while (pos!=null){
13     if (value <= pos.value){
14       append(subList, pos.value);
15       value = pos.value;
16     }
17     else
18       append(rest, pos.value);
19     pos = pos.next;
20   }
21   return (rest, subList);
22 }
```

Alle Operationen außer `merge` werden in konstanter Zeit $O(1)$ ausgeführt.

Vorname	Name	Matr.-Nr.

4

a) Sortieren Sie die folgende Liste anhand des `strandSort` Algorithmus. Notieren Sie dabei den Inhalt von `rest` und `subList` nach jedem Aufruf von `split` sowie den Inhalt von `sortedList` nach jedem Aufruf von `merge`.

$$4, 3, 1, 7, 5, 2, 6$$

```
1   List strandSort(List list){
2     if (list.head == null || list.head.next == null) return list;
3     (rest, subList) = split(list);
4     List sortedList = merge(strandSort(rest), subList);
5     return sortedList;
6   }
7
8   (List, List) split(List list){
9     List rest, subList;
10    int value = 0;
11    Element pos = list.head;
12    while (pos!=null){
13      if (value <= pos.value){
14        append(subList, pos.value);
15        value = pos.value;
16      }
17      else
18        append(rest, pos.value);
19      pos = pos.next;
20    }
21    return (rest, subList);
22  }
```

Vorname	Name	Matr.-Nr.

5

rest	subList	sortedList
3 1 5 2 6	4 7	
1 2	3 5 6	
–	1 2	
		1 2
		1 2 3 5 6
		1 2 3 4 5 6 7

Vorname	Name	Matr.-Nr.

b) Für welche Eingaben ergibt sich die Worst-Case Zeitkomplexität von `strandSort`? Geben Sie für Eingabelisten der Länge n die asymptotische Worst-Case Laufzeit als Rekursionsgleichung $T(n)$ an.

Lösung:

Im schlechtesten Fall ist die Liste invers sortiert. Die Laufzeit beträgt dann:

$$T(n) = n + T(n-1) + n \text{ mit } T(1) = 1$$

c) Lösen Sie die Rekursionsgleichung $T(n)$ aus b) auf.

Lösung:

$$\begin{aligned} T(n) &= 2n + T(n-1) \\ &= 2n + 2(n-1) + T(n-2) \\ &= T(1) + \Sigma_{i=2}^{n} 2i \\ &= 1 + 2\Sigma_{i=2}^{n} i \\ &= 2(\Sigma_{i=1}^{n} i) - 1 \\ &= n(n+1) - 1 \\ &\in O(n^2) \end{aligned}$$

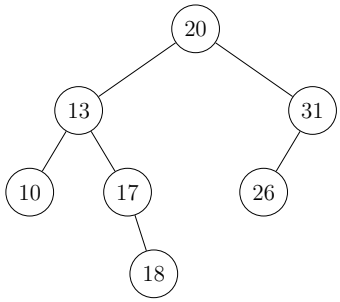
Vorname	Name	Matr.-Nr.

Aufgabe 3 Bäume (4+6=10 Punkte)

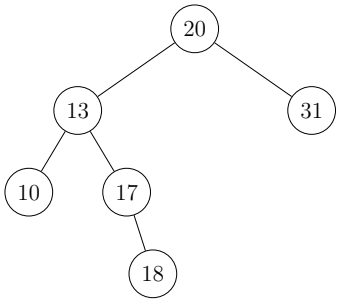
Ein AVL-Baum ist ein balancierter binärer Suchbaum, bei dem sich für jeden Knoten die Höhe seiner Teilbäume höchstens um eins unterscheiden darf.

Beispiel:

AVL-Baum:



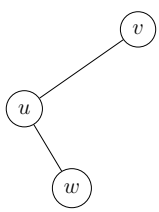
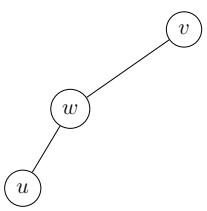
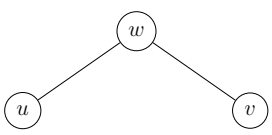
Kein AVL-Baum:



Falls durch das Einfügen eines Knotens die AVL-Eigenschaft verletzt wird, so muss diese durch eine Rebalancierung wiederhergestellt werden. Beim Einfügen können vier verschiedene Fälle von AVL-Verletzungen auftreten. Im Folgenden sind die zwei Fälle erläutert, in denen die Verletzung im Knoten v darin besteht, dass die Höhe des linken Nachfolgers um zwei größer ist als die des rechten. Die beiden anderen Fälle sind hierzu symmetrisch.

Fall 1**:** Tritt in einem Knoten v eine Verletzung der AVL-Eigenschaft auf, wobei die Höhe des linken Nachfolgers u um zwei größer ist als die des rechten und hat darüber hinaus w, der rechte Nachfolger von u, eine größere Höhe als sein Bruder (vergleiche die untenstehende Skizze), so überführen wir den Fall 1 mittels einer Linksrotation auf u in den 2. Fall, der dann noch aufgelöst werden muss.

Fall 2**:** Tritt eine Verletzung in einem Knoten v auf, wobei die Höhe des linken Nachfolgers w um zwei größer ist als die des rechten und hat darüber hinaus der linke Nachfolger von w eine größere Höhe als sein Bruder, so stellen wir die AVL-Eigenschaft mittels einer Rechtsrotation auf den Knoten v wieder her (vergleiche die untenstehende Skizze).

Fall 1:	Fall 2:	Rebalanciert:
		

Vorname	Name	Matr.-Nr.

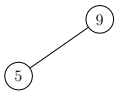
8

a) Geben Sie den AVL-Baum an, der entsteht, wenn die folgenden Zahlen in gegebener Reihenfolge in einen leeren Baum eingefügt werden: $9, 5, 2, 7, 8, 6$. Zeichnen Sie den Baum nach jeder Einfüge- bzw. Rotationsoperation.

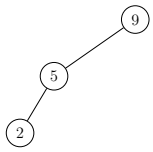
Lösung:

Einfügen 9:

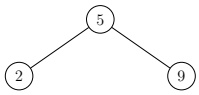
Einfügen 5:



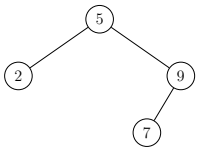
Einfügen 2:



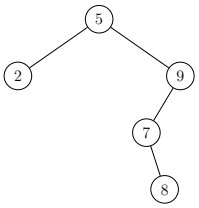
Fall 2:



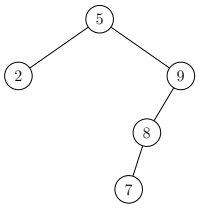
Einfügen 7:



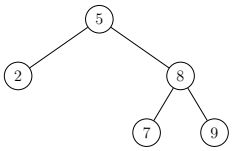
Einfügen 8:



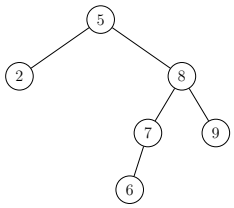
Rotation Fall 1:



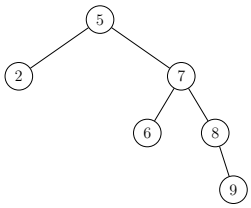
Rotation Fall 2:



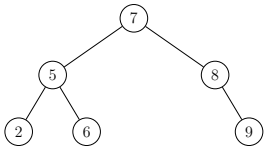
Einfügen 6:



Fall 1 (gespiegelt):



Fall 2 (gespiegelt):



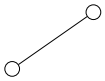
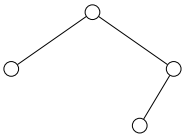
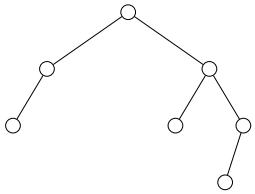
Vorname	Name	Matr.-Nr.

b) Zeigen Sie per Induktion, dass ein AVL-Baum der Höhe $h \geq 0$ mindestens $B(h) = Fib(h+1)$ Blätter hat.

Hinweise:

- Die Höhe eines Baums ist definiert als die Länge des längsten Pfades von der Wurzel bis zu einem Blatt.
- $Fib(n) = Fib(n-1) + Fib(n-2)$ für $n > 1$, $Fib(0) = 0$ und $Fib(1) = 1$

Lösung:

$B(0) = 1$:	$B(1) = 1$:	$B(2) = 2$:	$B(3) = 3$:
			

IA. $B(0) = 1 = Fib(1)$, $B(1) = 1 = Fib(2)$ ✓

IV. Angenommen $B(h) = Fib(h+1)$. Z.z.: $B(h+1) = Fib(h+2)$

IS. Ein AVL-Baum der Höhe $h+1$ mit minimaler Anzahl von Blättern lässt sich konstruieren, indem zwei Bäume der Höhe h und $h-1$ an einen neuen Wurzelknoten angehängt werden. Die Anzahl der Blätter entspricht dann der Summe der Blätter in den beiden Unterbäumen:

$$B(h+1) = B(h) + B(h-1) \overset{\text{nach IV.}}{=} Fib(h+1) + Fib(h) = Fib(h+2)\checkmark$$

Vorname	Name	Matr.-Nr.

Aufgabe 4 Graphen (6+2+2=10 Punkte)

a) Sei $G = (V, E)$ ein gerichteter Graph mit weiß oder blau gefärbten Knoten. Geben Sie einen auf Tiefensuche basierenden Algorithmus

```
boolean erfuellt(List adjLst[n], Color farbe[n], int n, int v)
```

an, der zu dem als Adjazenzliste gegebenen Graphen G und einem Knoten $v \in V$ entscheidet, ob von v aus ein blauer Knoten erreichbar ist, der mindestens einen Nachfolger besitzt und dessen *direkte* Nachfolger *alle* blau sind.

Lösung:

```
1  public boolean erfuellt(List adjList[n], Color farbe[n], int n, int v) {
2      Color color[n];
3
4      // Initialisierung
5      for(int i = 0; i < n; i++)
6          color[i] = WHITE;
7
8      // Suche starten
9      erfuellt(adjList, farbe, color, n, v)
10 }
11
12 public boolean erfuellt(List adjList[n], Color farbe[n],
13     Color color[n], int n, int v) {
14
15     // trifft nicht zu wenn es keine Nachfolger gibt
16     if (adjList[v].isEmpty)
17         return false;
18
19     // Knoten v wird bearbeitet
20     color[v] = GRAY;
21
22     // damit die Bedingung erfüllt ist muss v blau sein
23     boolean allBlue = farbe[v] == BLUE;
24
25     foreach (next in adjList[v]) {
26       allBlue = allBlue && farbe[next] == BLUE;
27
28       if (color[next] == WHITE)
29         if(erfuellt(adjList, farbe, color, n, next))
30             return true;
31     }
32     // Knoten v wurde abgearbeitet
33     color[v] = BLACK;
34
35     return allBlue;
36 }
```

Vorname	Name	Matr.-Nr.

b) Geben Sie die Laufzeitkomplexität Ihres Algorithmus aus a) an.

Lösung:

Die Laufzeit des Algorithmus entspricht dem der Tiefensuche: $O(|V| + |E|)$, da für das Überprüfen der Eigenschaft keine weiteren Schleifen benötigt werden.

c) Lösen Sie die Rekursionsgleichung $T(n) = 4T(\frac{n}{2}) + 3n^2$ mit Hilfe des Mastertheorems.

Lösung:

$E = \log_2 4 = 2$

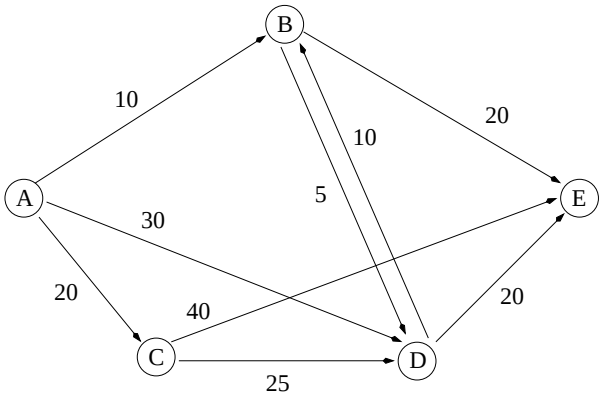
Mit $3n^2 \in \Theta(n^E)$ handelt es sich um den Fall 2 des Mastertheorems und somit gilt:

$$T(n) \in \Theta(n^2 \cdot \log n)$$

Vorname	Name	Matr.-Nr.

Aufgabe 5 Gewichtete Graphen (3+3+4=10 Punkte)

Der folgende gewichtete Graph G sei gegeben:



a) Ermitteln Sie mit Hilfe des Dijkstra-Algorithmus die kürzesten Wege von Startknoten A zu allen anderen Knoten des Graphen. Verwenden Sie hierzu die untenstehende Tabelle und notieren Sie für jeden Rechenschritt den aktuell gewählten Knoten zur Verbesserung der Wege sowie die Länge der bis zu diesem Zeitpunkt möglichen kürzesten Wege für jeden noch nicht abgeschlossenen Knoten ($D[\dots]$). Streichen Sie die Felder der Tabelle durch, die nicht mehr benötigt werden.

Lösung:

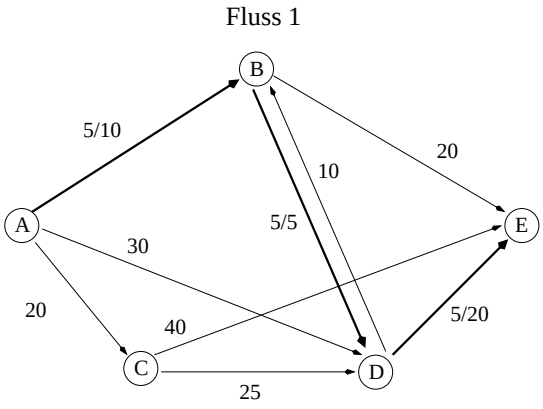
Schritt	0	1	2	3	4
Knoten	A	B	D	C	E
$D[A]$	0	–	–	–	–
$D[B]$	10	10	–	–	–
$D[C]$	20	20	20	20	–
$D[D]$	30	15	15	–	–
$D[E]$	∞	30	30	30	30

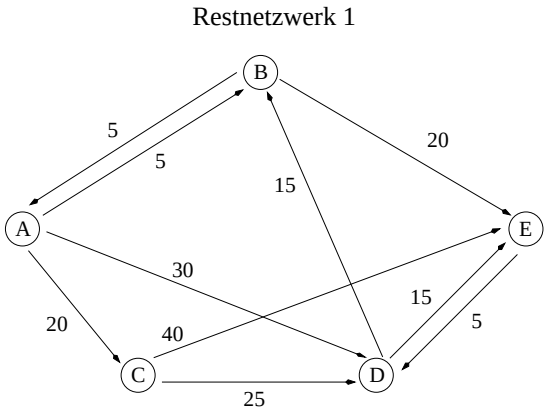
Vorname	Name	Matr.-Nr.

b) Wenden Sie die Ford-Fulkerson-Methode an, um zu G einen maximalen Fluss von $s = A$ nach $t = E$ zu berechnen.

(i) Erweitern Sie den noch leeren Fluss in der folgenden Abbildung entlang des augmentierenden Pfades $p = A \to B \to D \to E$ und geben Sie das zugehörige Restnetzwerk an.

Lösung:





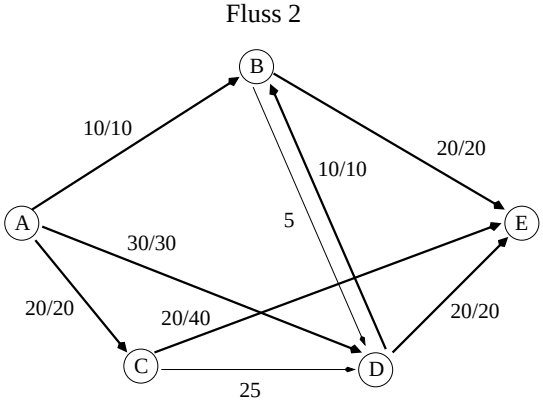
(ii) Bestimmen Sie weitere augmentierende Pfade ausgehend von obigem Fluss. Geben Sie sowohl die Pfade als auch ihre Restkapazitäten an und tragen Sie den resultierenden maximalen Fluss in die folgende Abbildung ein. Geben Sie zusätzlich das Restnetzwerk zu dem von Ihnen bestimmten maximalen Fluss an.

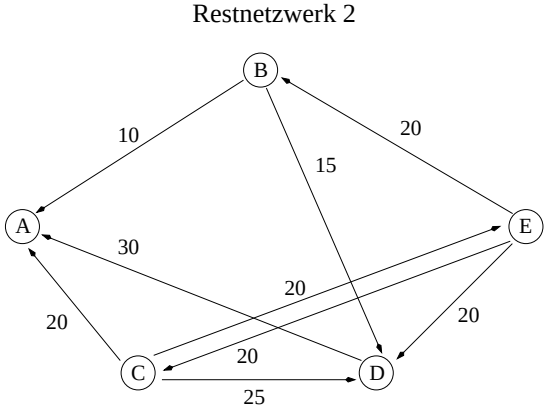
Lösung:

Neue Pfade sind:

$A \to B \to E$ (5), $A \to D \to E$ (15)

$A \to D \to B \to E$ (15), $A \to C \to E$ (20)





Es gibt nun im Restnetzwerk 2 keinen flussvergrößernden Weg mehr. Der Wert des maximalen Flusses beträgt 60.

Vorname	Name	Matr.-Nr.

c) Gegeben sei ein ungerichteter, gewichteter Graph $G = (V, E, w)$ und eine echte Teilmenge V' von V mit $V' \neq \emptyset$ und $V \neq V'$. Sei e eine Kante in G mit minimalem Gewicht, welche die Mengen V' und $V \setminus V'$ verbindet. Zeigen Sie, dass ein minimaler Spannbaum von G existiert, der e enthält.

Lösung:

Sei T ein MST in G und $e \notin T$. Fügen wir e in T ein, so erhalten wir einen Kreis in $T \cup \{e\}$. Dieser Kreis muss neben e eine zusätzliche Kante e' enthalten welche die Menge V und V' verbindet. Löschen wir nun e' aus $T \cup \{e\}$, so erhalten wir einen Baum $T' = T \cup \{e\} \setminus \{e'\}$. T' ist offensichtlich ein Spannbaum von G. Es sind zwei Fälle zu unterscheiden. Ist das Gewicht von e kleiner als das von e', so ist die Summe der Gewichte von T' kleiner als die von T. Dies widerspricht der Annahme, das T ein MST in G ist! Somit kann e kein kleineres Gewicht als e' besitzen.

Ist das Gewicht von e nicht kleiner als das von e', so müssen beide Gewichte gleich sein, da e eine minimale Kante zwischen V und V' ist. Ist dies der Fall, so handelt es sich offensichtlich auch bei T' um einen minimalen Spannbaum, da er das gleiche Gesamtgewicht besitzt.

Vorname	Name	Matr.-Nr.

Aufgabe 6 Baumstamm-Zerlegung (5+3+2=10 Punkte)

Ein Baumstamm der Länge L soll so zerlegt werden, dass möglichst wenig Verschnitt entsteht. Gehen Sie davon aus, dass beim Sägen kein Material verloren geht. Gegeben sind $n \geq 0$ Baumstücke der Längen $k_1, k_2, \ldots, k_n$, wobei $k_j > 0$ für alle $0 < j \leq n$. Gesucht ist eine Teilmenge von $\{1, \ldots, n\}$, so dass der Baumstamm mit möglichst wenig Rest zerlegt werden kann. Jedes Stück wird hierbei nur einmal benötigt.

Beispiel: Sei $L = 10$ und 3,4,6,5 die Längen der gewünschten Baumstücke. Schneidet man von dem Baumstamm Stücke der Länge 3 und 4 ab, so erhält man ein Reststück der Länge 3, welches nicht weiter verwendet werden kann. Zerlegt man den Baumstamm hingegen in Stücke der Längen 4 und 6, so bleibt kein Rest. Dies wäre eine optimale Zerlegung für das gegebene Problem.

a) Geben Sie eine rekursive Gleichung für das Teilproblem $C(i, l)$ an, wobei $C(i, l)$ der minimale Rest für einen Baumstamm der Länge $l \leq L$ und den Baumstücken $k_1, \ldots, k_i$, $0 \leq i \leq n$ ist.

   Lösung:

$$C(i,l) = \begin{cases} l & \text{für } i = 0, \\ min\{C(i-1,l), C(i-1,l-k_i)\} & \text{für } i > 0, l \geq k_i, \\ C(i-1,l) & \text{sonst.} \end{cases}$$

b) Geben Sie eine Implementierung nach dem Prinzip der dynamischen Programmierung an, die den minimalen Rest für eine gegebene Baumstammlänge $L \geq 0$ und Baumstücke $k_1, k_2, \ldots, k_n$ berechnet.

Lösung:

```
1   int Baumsaegen(int k[n], int L){
2     int C[n+1,L+1];
3     for (int l = 0; l <= L; l++)
4         C[0,l] = l;
5     for (int i = 1; i <= n; i++){
6         for (l = 0; l <= L; l++){
7             if (l - k[i-1] < 0)
8                 C[i,l] = C[i-1,l];
9             else
10                C[i,l] = min(C[i-1,l], C[i-1,l-k[i-1]);
11        }
12    }
13    return C[n,L];
14  }
```

c) Geben Sie die Worst-Case Zeit- und Speicherkomplexität des von Ihnen entworfenen Algorithmus an.

Lösung: Der angegebene Algorithmus enthält zwei Schleifen. Die erste wird L-fach durchlaufen. Bei der zweiten handelt es sich um eine verschachtelte Schleife, wobei die äußere n-fach und die innere L-fach durchlaufen wird. Somit kann die erste, einfache Schleife vernachlässigt werden und es ergibt sich eine Gesamtlaufzeit in $\Theta(n \cdot L)$.

Neben einigen Hilfsvariablen, die einen konstanten Platzbedarf verursachen, wird ein Array der Größe $(n+1) \cdot (L+1)$ genutzt. Der Platzbedarf liegt somit in $\Theta(n \cdot L)$