<table>
<tr><th>Teilaufg.</th><th>Aufgabe 1</th><th>Pkt</th></tr>
<tr><td>1.1.1</td><td>

Lösung in Java:

```
int mult1(int a, int b){
            int sum=0;
            int i;
            for(i=1;i<=a;i=i+1){
                  sum=sum+b;
            }
            return sum;
}
```

Lösung in Delphi:

```
function mult1(a,b:integer):integer;
var i:integer;
begin
      result:=0;
      for i:=1 to a do result:=result+b;
end;
```

</td><td>3</td></tr>
<tr><td>1.1.2</td><td></td><td></td></tr>
<tr><td>a)</td><td>$3 \cdot 7=7+2 \cdot 7=7+7+1 \cdot 7=7+7+7+0=21$</td><td>1</td></tr>
<tr><td>b)</td><td>

Lösung in Java:

```
int mult2(int a, int b){
            if(a>0){
                  return b+mult2(a-1,b);
            }
            else{
                  return 0;
            }
}
```

Lösung in Delphi:

```
function mult2(a,b:integer):integer;
begin
      if a>0 then result:=b+mult2(a-1,b)
               else result:=0;
end;
```

</td><td>3</td></tr>
</table>

1.1.3 — 3

Lösung in Java:

```
int mult3(int a, int b){
            if(a>=0){
                  return mult2(a,b);
            }
            else{
                  return -mult2(-a,b);
            }
}
```

Lösung in Delphi:

```
function mult3(a,b:integer):integer;
begin
      if a<0 then result:=-1*mult2(-1*a,b)
               else result:=mult2(a,b);
end;
```

Korrekt ist auch die Betrachtung einer anderen Fallunterscheidung.

1.2.1 — 2

Lösung in Java:

```
class Knoten{
            public int data;
            public Knoten next;
}
```

Lösung in Delphi:

```
type
      TZeiger=^TKnoten;
      TKnoten=record
            data:integer;
            next:TZeiger;
      end;
```

1.2.2

a) — 1

Lösung in Java:

```
L=L.next;
```

Lösung in Delphi:

```
L:=L^.next;
```

b) — 3

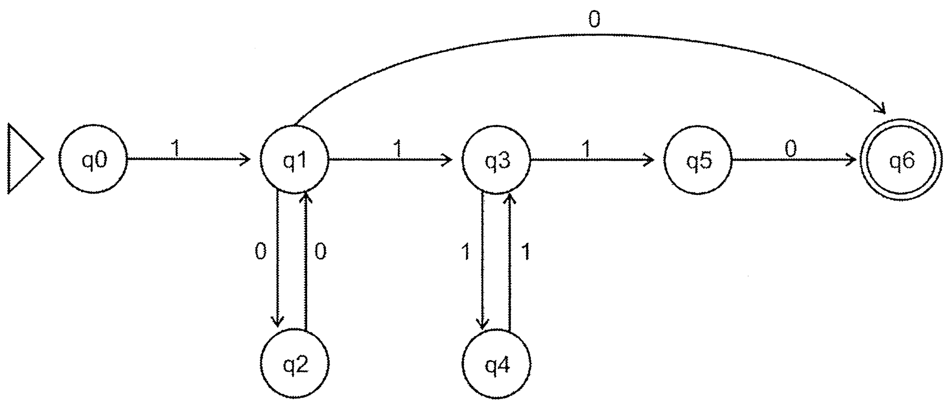
Lösung in Java:

```
Knoten kneu=new Knoten();
kneu.data=45;
kneu.next=L.next;
L.next=kneu;
```

Lösung in Delphi:

```
new(pneu);
pneu^.data:=45;
pneu^.next:=L^.next;
L^.next:=pneu;
```

<table>
<tr><td>1.2.3</td><td>Lösung in Java:
<pre>
int summe(Knoten k){
      Knoten kneu=new Knoten();
      int erg=0;
      kneu=k;
      while(kneu!=null){
            erg=erg+kneu.data;
            kneu=kneu.next;
      }
      return erg;
}
</pre>
Lösung in Delphi:
<pre>
function summe(pL:TZeiger):integer;
var paktuell:TZeiger;
begin
      result:=0
      paktuell:=pL;
      while paktuell<>nil do
            begin
                  result:=result+paktuell^.data;
                  paktuell:=paktuell^.next;
            end;
end;
</pre></td><td>4</td></tr>
</table>

Teilaufg.	**Aufgabe 2**	Pkt
2.1.1	Es ist $R = 1(00)^*(11)^*0$	2
2.1.2	Eine mögliche Grammatik lautet: $P = \{S \to 1A;\ A \to 00A \mid B;\ B \to 11B \mid 0\}$	2,5
2.1.3		3,5

2.2 (5 Pkt)

$\delta*$	1	2
$z0 := \{q0\}$	$\{q0, q1\}$	$\{\}$
$z1 := \{q0, q1\}$	$\{q0, q1\}$	$\{q0, q2\}$
$z2 := \{q0, q2\}$	$\{q0, q1\}$	$\{\}$

Startzustand= $z0$, Endzustand= $\{z2\}$, Alphabet= $\{1,2\}$, Zustände= $\{z0, z1, z2\}$, Übergangsfunktion $\delta*$ s. Tabelle

2.3.1 (3 Pkt)

Eingangsalphabet= $\{a, b, c\}$, Ausgabealphabet= $\{x, y, z\}$, Startzustand= $q0$, Zustände= $\{q0, q1, q2\}$, Endzustände= $\{\}$, Übergangsfunktion $\delta*$ s. Tabelle

$\delta*$	a	b	c
$q0$	$q1 \mid x$	$q2 \mid y$	$q0 \mid z$
$q1$	$q2 \mid z$	$q1 \mid y$	$q1 \mid x$
$q2$	$q0 \mid y$	$q0 \mid z$	$q2 \mid x$

2.3.2 (1,5 Pkt)

$$\rightarrow q0 \xrightarrow[x]{a} q1 \xrightarrow[y]{b} q1 \xrightarrow[x]{c} q1 \xrightarrow[z]{a} q2 \xrightarrow[z]{b} q0 \xrightarrow[z]{c} q0$$

2.3.3 (2,5 Pkt)

Das erste Zeichen ist ein `z` gewesen. Da sich der Automat im Startzustand befindet, muss das Eingabewort mit einem `c` begonnen haben. Der Automat befindet sich dann immer noch im Zustand $q0$.
Das zweite Ausgabezeichen ist ein `x`, daher muss das Eingabezeichen ein `a` gewesen sein.
Man fragt also rückwärts, wie ist die Ausgabe entstanden und in welchem Zustand befindet sich dann der Automat. Man erhält insgesamt die folgende Rückübersetzung `w=cabaca`, denn

$$\rightarrow q0 \xrightarrow[z]{c} q0 \xrightarrow[x]{a} q1 \xrightarrow[y]{b} q1 \xrightarrow[z]{a} q2 \xrightarrow[x]{c} q2 \xrightarrow[y]{a} q0$$

Teilaufg.	**Aufgabe 3**:	Pkt
3.1.1	Gegeben ist eine zufällige Anordnung einer vorgegebenen Menge von n Zahlen. In dieser Anordnung wird zunächst von vorne angefangen das größte Element gesucht, die Position und der Wert gespeichert und mit dem letzten Element der Anordnung getauscht. Damit ist der 1. Durchlauf beendet und das größte Element der Anordnung steht an letzter Stelle. Im 2. Durchlauf wird die neue Anordnung nur bis zum vorletzten Element durchlaufen. Auch hier wird nach dem größten Element gesucht und mit dem Element an vorletzter Stelle vertauscht. Damit verringert sich die zu durchlaufende Anzahl der Elemente jeweils um 1. Das Verfahren wiederholt man nun solange, bis nur noch die beiden ersten Elemente verglichen und evtl. vertauscht werden müssen.	3

3.1.2 — 3

	1.	2.	3.	4.	5.	6.	7.
Ausgangsanordnung	40	4	38	13	2	23	8
Nach dem 1. Durchlauf	4	38	13	2	23	8	**40**
Nach dem 2. Durchlauf	4	13	2	23	8	**38**	**40**
Nach dem 3. Durchlauf	4	2	13	8	**23**	**38**	**40**
Nach dem 4. Durchlauf	2	4	8	**13**	**23**	**38**	**40**
Nach dem 5. Durchlauf	2	4	8	13	23	38	40
Nach dem 6. Durchlauf	**Schluss**						

3.2.1

a) Es sind 7 Zahlen vorgegeben. Beim ersten Durchlauf müssen 6 Vergleiche durchgeführt werden. Beim 2. Durchlauf nur noch 5 Vergleiche, beim 3. Durchlauf dann 4, usw. Beim 6. Durchlauf ist dann nur noch ein Vergleich durchzuführen, so dass man folgende Summe erhält: 6+5+4+3+2+1=21. — 1

b) Für eine große Anzahl von n Elementen werden nach dem Maxsort-Verfahren also $\sum_{i=1}^{n-1} i = (n-1)+(n-2)+\ldots+2+1 = \frac{n\cdot(n-1)}{2} = \frac{1}{2}\left(n^2-n\right)$ Vergleiche durchgeführt. Daher ist die Laufzeit von der Ordnung $O(n^2)$. — 2

3.2.2

a) Im günstigsten Fall ist die Anordnung in der richtigen Reihenfolge. Das Bubblesort-Verfahren läuft dann die Anordnung nur ein Mal durch. Es werden dann (n-1) Vergleiche durchgeführt. Das Laufzeitverhalten ist also $O(n)$. — 2

Im ungünstigsten Fall ist die Liste falsch herum sortiert. Dann werden zunächst (n-1), dann (n-2), dann (n-3), ... Vergleiche pro Durchgang durchgeführt. Es ergeben sich somit n Durchläufe und daher ein Laufzeitverhalten der Ordnung $O(n^2)$.

b) Das Bubblesort-Verfahren hat in der Regel ein besseres Laufzeitverhalten als das Maxsort-Verfahren. Nur wenn die Reihe „falschherum" sortiert ist, stimmt das Laufzeitverhalten des Bubblesort-Verfahrens mit dem Laufzeitverhalten des Maxsort-Verfahrens überein. Während eine Vorsortierung auf das Laufzeitverhalten des Maxsortverfahrens keinen Einfluss hat, denn es wird von jeder verringerten Anordnung jeweils das Maximum gesucht, verbessert sich die Laufzeit beim Bubblesort-Verfahren. — 1,5

3.3.1		
a)	Durch Vergleich der Häufigkeitsverteilungen kann jedem Buchstaben des kodierten Textes ein entsprechendes Zeichen des ursprünglichen Textes zugeordnet werden. Eventuell unklare Zuordnungen können dann noch aus dem Kontext heraus gebildet werden.	1,5
b)	Jedem Geheimtextzeichen wird eindeutig ein Klartextzeichen zugeordnet und umgekehrt.	1
c)	Das RSA-Verfahren ist ebenfalls ein Substitutionsverfahren, da auch hier bestimmte Zahlen durch andere Zahlen ersetzt werden. Das RSA-Verfahren wehrt einen Angriff durch die Häufigkeitsanalyse durch die Bildung von Blöcken B einer bestimmten Länge l ab. Es sind die Blöcke, die dann durch andere Zeichen (andere Blöcke) substituiert werden. Durch die Blockbildung wird die Häufigkeitsanalyse zumindest erschwert.	2
3.3.2		
a)	Bei dem Automaten wird jeder Buchstabe des Eingangsalphabets durch einen anderen Buchstaben des Ausgabealphabets ersetzt. Daher stellt der Automat ein Substitutionsverfahren dar.	1
b)	Die Zuordnung der Buchstaben des Eingangsalphabets zu den Buchstaben des Ausgangsalphabets ist abhängig von dem Zustand, in dem sich der Automat gerade befindet. Es wird also nicht immer einem Buchstaben des Eingangsalphabets derselbe Buchstabe des Ausgangsalphabets zugeordnet.	2