Aufgabe 1: Eigenschaften von Algorithmen

(a) Terminiertheit: Terminiert dieser Algorithmus?

Dieser Algorithmus terminiert, da n *immer* mindestens um 1 vermindert wird. Somit muss n - 1 immer irgendwann < 1 sein, wodurch die Intervalllänge nicht > 0 ist und der Algorithmus terminiert.

(b) Determinismus: Ist dieser Algorithmus deterministisch?

Dieser Algorithmus ist nicht deterministisch, weil z eine *zufällige* Zahl aus dem Intervall 1 .. n-1 ist.

(c) Determiniertheit: Ist dieser Algorithmus determiniert?

Dieser Algorithmus ist determiniert, da n immer um mindestens 1 und höchstens n - 1 vermindert wird. Da dieser Schritt wiederholt wird solange die Länge des Intervalls > 0 ist und die Intervalllänge 1 .. n - 1 ist, ist die einzige mögliche Ausgabe 1.

Aufgabe 2: Nim-Spiel

(a) Optimale Strategie

Die Optimale Strategie besagt, dass man immer gewinnen kann, wenn der Gegner beginnt und der aktuelle Stapel die Größe „n = 1 + 4 * h (h = 1, 2, 3 …..)“ hat. Zieht nun der Gegner 1 Holz, zieht man 3. Zieht der Gegner 2, so zieht man auch 2. Zieht er hingegen 3, zieht man nur eines. Man zieht also immer genau so viel, dass die Summe der in der letzten Runde gezogenen Hölzer 4 ergibt.
Durch diese Strategie ergibt sich immer wieder das Muster „n = 1 + 4*h (h = 1, 2, 3 …)“ bis nur noch 1 Holz übrig ist (h = 0). Da der Gegner jede Runde begonnen hat muss er das letzte Holz ziehen und verliert somit.
Will man nun gewinnen wenn der Stapel nicht die vorgegebene Größe hat und man selbst beginnt, so muss man ihn in seinem ersten Zug auf “n = 1 + 4 * h (h = 1, 2, 3….)” bringen. Als Beispiel ist der Stapel zu Beginn 23, so muss man im ersten Zug 2 ziehen damit er die Größe 21 hat (n = 1 + 4 * 5). Von da an kann man die oben beschriebene Strategie nutzen.

Marcel Herd (1527440), Firas Romdhane (1525044), Manuel Schwalm (1521130)

(b) Pseudo-Code zur optimalen Strategie

```
proc zug (n)
   Initialisiere maxHoelzer mit 3;
   if n < 3
         then maxHoelzer = n;
   fi
   Initialisiere bZug mit einer Zahl zwischen 1 und maxHoelzer;
   Ziehe von n bZug ab;
   drucke("Spieler b Zieht " + bZug + " Hölzer. Es bleiben " + n + " Hölzer übrig.");
   if n = 0
         then
               drucke("Der Stapel ist leer. Da Spieler b den Letzten Zug machte, hat er
verloren.");
               return;
   fi
   Initialisiere aZug mit 4;
   Ziehe von aZug bZug ab;
   Ziehe von n aZug ab;
   drucke("Spieler a Zieht " + aZug + " Hölzer. Es bleiben " + n + " Hölzer übrig.");
   if n = 0
         then
               drucke("Der Stapel ist leer. Da Spieler a den Letzten Zug machte, hat er
verloren.");
               return;
   fi
   zug(n);
corp
```

(c)

		Zug Spieler	
Runde	Stapel	A	B
1	23	2	-
	21	-	1
2	20	3	-
	17	-	2
3	15	2	-
	13	-	3
4	10	1	-
	9	-	1
5	8	3	-
	5	-	2
6	3	2	-
	1	-	1
7	0		

Aufgabe 3: Rekursiver Algorithmus

0. Gegeben: Aufgabe der Größenordnung n
   Bsp. Quersumme(155); n = 3
1. Finde eine kleinere Aufgabe
   Quersumme(15) + 5
2. Rekursiver Fall
   letzteZiffer + Quersumme(restlicheZiffern)
3. Trivialer Fall
   Quersumme(i) bei n = 1
4. Fallunterscheidung
   i < 10
5. Funktion

```
funct Quersumme (n) returns integer
   if n < 10
   then
         return n;
   else
          return letzteZiffer(n) + Quersumme(restlicheZiffern(n));
   fi
tcnuf
```

Marcel Herd (1527440), Firas Romdhane (1525044), Manuel Schwalm (1521130)

Aufgabe 4: Iteration und Rekursion

(a) Rekursive Funktion

```
[1]     funct Multipliziere (a, b) returns Integer
[2]             if b = 1
[3]             then
[4]                     return a;
[5]             else
[6]                     return a + Multipliziere(a, b-1);
[7]             fi
[8]     tcnuf
```

(b) Protokoll

Aufruf	Zeile	Aufführende Operation	Hinweis wo das Ergebnis weiterverwendet wird
1	1	Multipliziere(3, 4);	
2	2	if 4 = 1;	
3	4	else return 3 + Multipliziere (3, 3);	
4	1	Multipliziere(3, 3);	Ergebnis wird in 3 verwendet
5	2	if 3 = 1;	
6	4	else return 3 + Multipliziere (3, 2);	
7	1	Multipliziere(3, 2);	Ergebnis wird in 6 verwendet
8	2	if 2 = 1;	
9	4	else return 3 + Multipliziere (3, 1);	
10	1	Multipliziere(3, 1);	Ergebnis wird in 8 verwendet
11	2	if 1 = 1;	
12	3	then return 3;	

(c) Iterative Funktion

```
[1]     funct Multipliziere (a, b) returns Integer
[2]             if b = 1
[3]             then
[4]                     return a;
[5]             else
[6]                     Initialisiere summe mit 0;
[7]                     while b >= 1
[8]                     do
[9]                             addiere a auf die Summe;
[10]                            vermindere b um 1;
[11]                    od
[12]                    return summe;
[13]            fi
[14]    tcnuf
```

(d) Protokoll

Aufruf	Zeile	Auführende Operation	Hinweis wo das Ergebnis weiterverwendet wird
1	1	Multipliziere(3, 4)	
2	2	if 4 = 1	
3	5	else	
4	6	Initialisiere summe mit 0	Summe = 0
5	7	while 4 >= 1 do	
6	9	addiere 3 auf die Summe	Summe = 3
7	10	vermindere b um 1	b = 3
8	7	while 3 >= 1 do	
9	9	addiere 3 auf die Summe	Summe = 6
10	10	vermindere b um 1	b = 2
11	7	while 2 >= 1 do	
12	9	addiere 3 auf die Summe	Summe = 9
13	10	vermindere b um 1	b = 1
14	7	while 1 >= 1 do	
15	9	addiere 3 auf die Summe	Summe = 12
16	10	vermindere b um 1	b = 0
17	11	od	

Marcel Herd (1527440), Firas Romdhane (1525044), Manuel Schwalm (1521130)

18	12	return summe;	12 wird als ergebnis zurückgegeben