Vorname Nachname Matrikel-Nr

Diese Klausur ist mein **letzter Prüfungsversuch** (bitte ankreuzen): Ja ○ Nein ○

Diese Klausur besteht aus 5 Aufgaben. Schreiben Sie jede Lösung einer Aufgabe auf die *Vorderseite* eines Ihrer Lösungsblätter (und lassen Sie die Rückseiten Ihrer Lösungsblätter **leer**).

Aufgabe 1 (20 Punkte): Schreiben Sie eine rekursive Methode ("Schleifen dürfen hier nicht rein") entsprechend der folgenden Spezifikation:

```
 1   static int anzahl(int n, int z) {
 2      // Verlaesst sich darauf, dass n groesser oder gleich 0 ist
 3      // und dass z zwischen 0 und 6 (einschliesslich) liegt
 4      // (d.h. "z ist eine Ziffer im 7-er-System").
 5      // Angenommen, wir stellen n als 7-er-Zahl dar. Wie oft kommt
 6      // in dieser Darstellung die Ziffer z vor?
 7      // Diese Methode liefert die Antwort.
 8      //
 9      // Beispiele:
10      // anzahl( 5, 3) ist gleich 0 (denn  5 als 7-er-Zahl:   5)
11      // anzahl( 3, 3) ist gleich 1 (denn  3 als 7-er-Zahl:   3)
12      // anzahl( 7, 1) ist gleich 1 (denn  7 als 7-er-Zahl:  10)
13      // anzahl( 8, 1) ist gleich 2 (denn  8 als 7-er-Zahl:  11)
14      // anzahl(10, 3) ist gleich 1 (denn 10 als 7-er-Zahl:  13)
15      // anzahl(21, 3) ist gleich 1 (denn 21 als 7-er-Zahl:  30)
16      // anzahl(24, 3) ist gleich 2 (denn 24 als 7-er-Zahl:  33)
17      // anzahl(57, 1) ist gleich 3 (denn 57 als 7-er-Zahl: 111)
18      ...
19   }
```

Tipp: Wir haben behandelt, wie man aus einer `int`-Variablen wie `n` die einzelnen Ziffern berechnen kann, und zwar in einem beliebigen Zahlensystem (z.B. im 7-er-System).

Aufgabe 2 (20 Punkte): Hier geht es um Bojen. Betrachten Sie die folgenden Befehle:

```
1      Object[] obr = new Object[4];
2
3      obr[1] = "ABC";
4      obr[2] = new int[]{10, 20};
5      obr[3] = obr;
```

Wie sieht die Variable `obr` nach Ausführung der Zeile 1 aus?
Wie sieht die Variable `obr` nach Ausführung der Zeile 5 aus?
Stellen Sie als Antworten auf diese Fragen die Variable `obr` zweimal *als Boje* dar.

Tipp: Die Zuweisung in Zeile 5 ist erlaubt, und "wie immer" weist sie der Variablen auf der linken Seite (`obr[3]`) den Wert des Ausdrucks (`obr`) auf der rechten Seite zu.

Aufgabe 3 (20 Punkte): In dieser Aufgabe geht es (wie in einigen SUs besprochen) um *Algorithmen* und dazu passenden *Schrittfunktionen*.

Zur Erinnerung: Ein Algorithmus wird hier durch eine Methode mit dem Rückgabetyp `void` und einem `int`-Parameter `n` dargestellt. Der Parameter `n` beschreibt die *Größe des Problems*, welches mit dem Algorithmus gelöst werden soll. Der Algorithmus macht nichts weiter, als eine Methode `schritt` aufzurufen. Die Anzahl dieser Schritte (Aufrufe der Methode `schritt`) hängt meistens von `n` ab, z.B. so wie bei dem folgenden Algorithmus `alg01`:

```
static void alg01(int n) {
   for (int i=1; i<=3*n; i++) schritt();
}
```

Eine zu diesem Algorithmus passende Schrittfunktion `stp01` sollte etwa so aussehen:

```
static in stp01(int n) {
   return 3*n;
}
```

Für jedes `n` soll `stp01(n)` die Anzahl der Schritte liefern, die der Aufruf `alg01(n)` auslöst. Wenn `stp01` das tut, dann
passt die Schrittfunktion `stp01` zum Algorithmus `alg01`
und der Algorithmus `alg01` passt zur Schrittfunktion `stp01`.

Teilaufgabe 3.1. Gegegen sei der Algorithmus `alg17`:

```
static void alg17(int n) {
   for (int i=1; i<=n; i++) {
      for (int j=1; j<=i; j++) {
         schritt();
      }
   }
   for (int i=1; i<=3*n; i++) {
      for (int j=i; j<=3*n; j++) {
         schritt();
      }
   }
}
```

Geben Sie eine zu diesem Algorithmus passende Schrittfunktion `stp17` an. Vereinfachen Sie die Schrittfunktion so weit wie möglich (**Beispiel**: statt `(6*n + 2*n)/4` sollten Sie `2*n` schreiben).

Teilaufgabe 3.2. Gegen sei die Schrittfunktion `stp18`:

```
static int stp18(int n) {
   return 3*n*n + 5*n + 100;
}
```

Geben Sie einen zu dieser Schrittfunktion passenden Algorithmus `alg18` an.

Aufgabe 4 (20 Punkte): Die folgenden Fragen haben mit den Projekten 1 bis 6 zu tun (d.h. mit den Klassen `LongSpeicher10` bis `Longspeicher60`), die im Laufe des Semester behandelt wurden. Beantworten Sie die Fragen möglichst kurz und einfach. Bei einigen Fragen genügt so etwas Ähnliches wie "`fuegeEin` ist langsamer" oder "`istDrin` ist schneller", bei anderen Fragen braucht man ein paar Worte mehr. Informelle Begriffe wie *Gummi* und *Beton*, die in den SUs mehrfach benutzt wurden, sind erlaubt.

4.1. Was ist bei einer *unsortierten Reihung* besser als bei einer *sortierten Reihung*?

4.2. Was ist bei einer *sortierten Reihung* besser als bei einer *unsortierten Reihung*?

4.3. Was ist bei einer *unsortierten Liste* besser als bei einer *unsortieren Reihung*?

4.4. Was ist bei einem *binären Baum* besser als bei einer *sortierten Liste*?

4.5. Was ist bei einer *Hash-Tabelle* besser als bei einem *binären Baum*?

4.6. Welche Zeitkomplexität hat die Methode `istDrin` bei einer *unsortierten Reihung*?

4.7. Welche Zeitkomplexität hat die Methode `istDrin` bei einer *sortierten Reihung*?

4.8. Welche Zeitkomplexität hat die Methode `istDrin` bei einem *binären Baum* ("wenn der Baum balanciert bleibt und nicht entartet")?

4.9. Welche Zeitkomplexität hat die Methode `istDrin` bei einer *Hash-Tabelle* ("wenn die Hash-Funktion gut funktioniert")?

4.10. Die Zeitkomplexität $O(n^3)$ ist "schlimmer" als die Zeitkomplexität $O(n^2)$ (ab einem bestimmten n wird ein $O(n^3)$-Algorithmus mehr Schritte machen müssen als ein $O(n^2)$ Algorithmus). Geben Sie zwei Zeitkomplexitäten an, die "schlimmer" sind als *alle* polynomialen Zeitkomplexitäten ($O(n^0)$, $O(n^1)$, $O(n^2)$, $O(n^3)$, ..., $O(n^{17})$, ...).

Aufgabe 5 (20 Punkte): Die folgenden Fragen haben mit dem Thema "Ist P gleich NP?" zu tun.

5.1. Geben Sie den Namen eines algorithmischen Problems an, von dem bewiesen wurde, dass man es prinzipiell *nicht* mit einem Computer lösen kann (egal wie groß und schnell Computer noch werden).

5.2. Mit **NP** bezeichnet man eine bestimte Klasse ("Gruppe") von algorithmischen Problemen. Welche algorithmischen Probleme gehören zu dieser Klasse? Hier sollen Sie eine (kurze) allgemeine *Definition* angeben (und nicht versuchen, die mehr als 3000 bekannten Elemente dieser Klasse aufzuzählen).

5.3. Geben Sie die Namen von drei algorithmischen Problemen an, die zur Klasse **NP** , aber (zumindest heute noch) nicht zur Klasse **P** gehören.

5.4. Sei AP1 eines dieser drei algorithmischen Probleme. Was müsste passieren, damit AP1 in Zukunft zur Klasse **P** gehört?

5.5. Wann ist eine *aussagenlogische Formel* (z.B. (A oder B) und ($\overline{A}$ oder $\overline{B}$)) *erfüllbar*?

Beurteilung der Klausur:

Punkte↓	
Aufgabe 1:	Note:
Aufgabe 2:	Datum: 24.07.2015
Aufgabe 3:	
Aufgabe 4:	
Aufgabe 5:	
Summe:	

Korrigierte Beurteilung der Klausur:

Punkte↓	
Aufgabe 1:	Note:
Aufgabe 2:	Datum:
Aufgabe 3:	
Aufgabe 4:	
Aufgabe 5:	
Summe:	

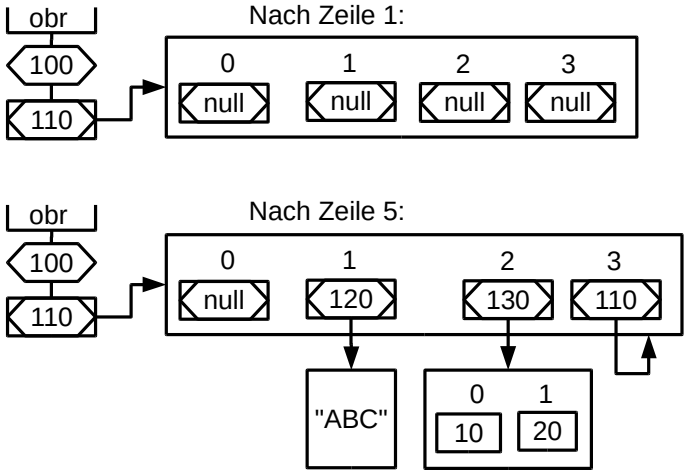
Lösung 1 (20 Punkte): Eine rekursive Methode namens `anzahl`:

```
 1   static int anzahl(int n, int z) {
 2      // Verlaesst sich darauf, dass n groesser oder gleich 0 ist
 3      // und dass z zwischen 0 und 6 (einschliesslich) liegt
 4      // (d.h. "z ist eine Ziffer im 7-er-System").
 5      // Angenommen, wir stellen n als 7-er-Zahl dar. Wie oft kommt
 6      // in dieser Darstellung die Ziffer z vor?
 7      // Diese Methode liefert die Antwort.
 8
 9      // Erste Version:
10//    if (n == z) return 1;                 // Einzige Ziffer   gleich z
11//    if (n < 7)  return 0;                 // Einzige Ziffer ungleich z
12//
13//    if (n%7==z) return anzahl(n/7, z) + 1; // Letzte Ziffer  gleich z
14//                return anzahl(n/7, z);     // Letzte Ziffer unglich z
15
16      // Zweite Version:
17      int letzt = (n%7 == z) ? 1 : 0;
18      if (n<7) return letzt;
19      return anzahl(n/7, z) + letzt;
20   }
```

Lösung 2 (20 Punkte): Hier geht es um Bojen. Betrachten Sie die folgenden Befehle:

```
1      Object[] obr = new Object[4];
2
3      obr[1] = "ABC";
4      obr[2] = new int[]{10, 20};
5      obr[3] = obr;
```

Wie sieht die Variable `obr` nach Ausführung der Zeile 1 aus?
Wie sieht die Variable `obr` nach Ausführung der Zeile 5 aus?
Stellen Sie als Antworten auf diese Fragen die Variable `obr` zweimal als Boje dar.



Lösung 3 (20 Punkte):

Teillösung 3.1.: Gegegen sei der Algorithmus `alg17`:

```
 1   static void alg17(int n) {
 2      for (int i=1; i<=n; i++) {
 3         for (int j=1; j<=i; j++) {
 4            schritt();
 5         }
 6      }
 7      for (int i=1; i<=3*n; i++) {
 8         for (int j=i; j<=3*n; j++) {
 9            schritt();
10         }
11      }
12   }
```

Geben Sie eine zu diesem Algorithmus passende Schrittfunktion `stp17` an.

```
13   static int stp17(int n) {
14      return 5*n*n + 2*n;
15   }
```

Teillösung 3.2.: Gegen sei die Schrittfunktion `stp18`:

```
 1   static int stp18(int n) {
 2      return 3*n*n + 5*n + 100;
 3   }
```

Geben Sie einen zu dieser Schrittfunktion passenden Algorithmus `alg18` an.

```
 4   static void alg18(int n) {
 5      for (int i=1; i<=3; i++) {
 6         for (int i1=1; i1<=n; i1++) {
 7            for (int i2=1; i2<=n; i2++) {
 8               schritt();
 9            }
10         }
11      }
12      for (int i=1; i<=5; i++) {
13         for (int i1=1; i1<=n; i1++) {
14            schritt();
15         }
16      }
17      for (int i=1; i<=100; i++) {
18         schritt();
19      }
20   }
```

Lösung 4 (20 Punkte): Die folgenden Fragen haben mit den Projekten 1 bis 6 zu tun (d.h. mit den Klassen `LongSpeicher10` bis `Longspeicher60`), die im Laufe des Semester behandelt wurden. Beantworten Sie die Fragen möglichst kurz und einfach. Bei einigen Fragen genügt so etwas Ähnliches wie "`fuegeEin` ist langsamer" oder "`istDrin` ist schneller", bei anderen Fragen braucht man ein paar Worte mehr. Informelle Begriffe wie *Gummi* und *Beton*, die in den SUs mehrfach benutzt wurden, sind erlaubt.

4.1. Was ist bei einer unsortierten Reihung besser als bei einer sortierten Reihung?

fuegeEin geht schneller

4.2. Was ist bei einer sortierten Reihung besser als bei einer unsortierten Reihung?

i**stDrin geht schneller**

4.3. Was ist bei einer unsortierten Liste besser als bei einer unsortieren Reihung?

Die Liste "ist aus Gummi", die Reihung "ist aus Beton".

4.4. Was ist bei einem binären Baum besser als bei einer sortierten Liste?

istDrin geht schneller

4.5. Was ist bei einer Hash-Tabelle besser als bei einem binären Baum?

fuegeEin, istDrin und loesche sind schneller

4.6. Welche Zeitkomplexität hat die Methode `istDrin` bei einer *unsortierten Reihung*?

O(n)

4.7. Welche Zeitkomplexität hat die Methode `istDrin` bei einer *sortierten Reihung*?

O(log(n))

4.8. Welche Zeitkomplexität hat die Methode `istDrin` bei einem *binären Baum*
("wenn der Baum balanciert bleibt und nicht entartet")?

O(log(n))

4.9. Welche Zeitkomplexität hat die Methode `istDrin` bei einer *Hash-Tabelle*
("wenn die Hash-Funktion gut funktioniert")?

O(1)

4.10. Die Zeitkomplexität $O(n^3)$ ist "schlimmer" als die Zeitkomplexität $O(n^2)$ (ab einem bestimmten n wird ein $O(n^3)$-Algorithmus mehr Schritte machen müssen als ein $O(n^2)$ Algorithmus).
Geben Sie zwei Zeitkomplexitäten an, die "schlimmer" sind als *alle* polynomialen Zeitkomplexitäten ($O(n^0)$, $O(n^1)$, $O(n^2)$, $O(n^3)$, ..., $O(n^{17})$, ...).

$O(2^n)$, O(n!), ...

Lösung 5 (20 Punkte): Die folgenden Fragen haben mit dem Thema "Ist P gleich NP?" zu tun.

5.1. Geben Sie den Namen eines algorithmischen Problems an, von dem bewiesen wurde, dass man es prinzipiell nicht mit einem Computer lösen kann (egal wie groß und schnell Computer noch werden).

Das Halte-Problem,
Das Problem, diophantische Gleichungen zu lösen.

5.2. Mit **NP** bezeichnet man eine bestimte Klasse ("Gruppe") von algorithmischen Problemen. Welche algorithmischen Probleme gehören zu dieser Klasse? Hier sollen Sie eine (kurze) allgemeine *Definition* angeben (und nicht versuchen, die mehr als 3000 bekannten Elemente dieser Klasse aufzuzählen).

Zu NP gehören alle algorithmischen Probleme von denen gezeigt wurde, dass man sie mit einem nicht-deterministischen Programm in polynomialer Zeit lösen kann.

5.3. Geben Sie die Namen von 3 algorithmischen Problemen an, die (zumindest heute noch) zur Klasse **NP** gehören.

SAT-Problem,
Hamilton-Problem,
Rucksack-Problem,
Problem des Handlungsreisenden,
Teilsummen-Problem,
Umwandlung von aussagenlogischen Formeln (und-Normalform ↔ oder-Normalform)
...

5.4. Wann ist eine *aussagenlogische Formel* (z.B. (A oder B) und ($\overline{A}$ oder $\overline{B}$)) *erfüllbar*?

Wenn es es eine Variablenbelegung (für die Variablen der Formel) gibt, mit der die Formel den Wert wahr beschreibt.

5.5. Sei AP1 eines dieser 3 algorithmischen Problme. Was müsste passieren, damit AP1 in Zukunft zur Klasse **P** gehört.

Jemand müsste ein deterministisches Programm schreiben, welches das Problem P in polynomialer Zeit löst.