Algorithmen und Datenstrukturen INF3/ICS3

Wintersemester 2023/24

Prof. Dr. Georg Schied

Aufgabenblatt 1

Abgabetermin: Do. 19. Oktober 2023, 23:59 Uhr
10 von 20 Punkten erforderlich

- Bearbeitung in 2er-Gruppen ist erlaubt. In Moodle muss jeder in einer Gruppe sein. Ggf. 1er-Gruppe bilden, wer alleine abgegeben möchte.
- Den Programmcode bitte in vollständiger, übersetzbarer Form über Moodle hochladen.
- Ein Projekt für alle Aufgaben in diesem Semester anlegen. Pro Aufgabenblatt gibt es ein Java-Paket (`blatt01`, `blatt02`, …). Dann das Paket-Verzeichnis für den Java-Sourcecode des Aufgabenblatts als zip-Datei zusammenpacken und hochladen (auch zusammen mit den vorgegebenen JUnit-Testdateien etc.). Die Programmvorlagen sind schon entsprechend vorbereitet.

Aufgabe 1.1

a) $\sum_{k=1}^{100} 5k$

b) $\sum_{i=0}^{10} 3^i$

c) $\sum_{j=n}^{m} 2j$

Aufgabe 1.2 - Scheinaufgabe (2 P)

Gegeben sind folgende geschachtelte Schleifen:

```
for (int j = 2*n; j >= 1; j--) {
   for (int v = 0; v < j; v++) {
       …
   }
}
```

Wie oft wird insgesamt der Rumpf der inneren Schleife ausgeführt, abhängig vom Wert der Variable $n \geq 0$? Begründen Sie kurz Ihr Ergebnis.

Aufgabe 1.3 - Scheinaufgabe (6 P)

Zwei Zeichenketten sind **Anagramme**, wenn die eine Zeichenkette durch Umordnung der Zeichen aus der anderen gebildet werden kann. Beispiele dafür sind:

WIEN / WEIN
LAMPE / PALME
PERMUTATION / TRAUMPOETIN
SCHUTZUMSCHLAG / UMZUGSSCHLACHT
CORONAVIRUS / CARNIVOROUS

a) Implementieren Sie eine möglichst effiziente Methode

```
public static boolean areAnagrams(String s1, String s2)
```

die prüft, ob die beiden Strings `s1` und `s2` Anagramme sind. Die Methode soll auch für sehr lange Zeichenketten (z.B. mit 1 Mio. Zeichen) einsetzbar sein. Sie können dabei davon ausgehen, dass nur die ersten 256 Zeichen des Unicode-Zeichensatzes in den Strings vorkommen (d.h. die Zeichen \u0000 bis \u00FF). Sie dürfen Klassen und Methoden der Java-Standardbibliothek verwenden.

b) Messen Sie die Laufzeit der Methode für die Längen *n* = 100, 1 000, 10 000, 100 000, 1 000 000. Achten Sie darauf, dass für die Ausführung bei der Java-VM die Option `-Xint` gesetzt ist (d.h. ohne Just-in-time-Compiler). Welches Laufzeitverhalten ist bei Ihrer Implementierung zu erkennen?
Die Tabelle mit den Messwerten mit abgeben!

In Moodle finden Sie als Vorlage eine Klasse `Aufg13_Anagramme`, die schon Hilfsmethoden zum Testen und zum Messen der Laufzeit enthält, so dass Sie nur noch die Methode `sindAnagramme` ergänzen müssen. Außerdem gibt es eine JUnit-Testklasse `JuTest_Aufg13_Anagramme` dafür.

Aufgabe 1.4

a) Programmieren Sie eine Methode `contains`, die *rekursiv* prüft, ob ein Wert `x` im Teilarray `arr[0..endIndex]` enthalten ist.

```
public static boolean contains(double x, double[] arr,
                               int endIndex)
```

b) Programmieren Sie eine Methode

```
public static boolean isPalindrome(char[] sequence)
```

die *rekursiv* prüft, ob die angegebene Zeichenfolge ein Palindrom ist, d.h. ob die Zeichenfolge von vorne nach hinten gleich ist wie von hinten nach vorne.
Tipp: Verwenden Sie eine rekursive Hilfsmethode.

In Moodle finden Sie die Klasse `Aufg14_Rekursion` als Programmvorlage und Klasse `JuTest_Aufg14_Rekursion` mit JUnit-Tests.

Aufgabe 1.5 - Scheinaufgabe (6 P)

a) Programmieren Sie rekursiv eine Methode `containsOdd`, die prüft, ob mindestens ein ungerader Wert im Teilarray `arr[0..endIndex]` enthalten ist.

```
public static boolean containsOdd(int[] arr, int endIndex)
```

b) Implementieren Sie eine Methode `allOdd`, die rekursiv prüft, ob alle Werte im Array `arr` ungerade sind.

```
public static boolean allOdd(int[] arr)
```

Sie können eigene Hilfsmethoden einführen. Verwenden Sie aber keine Methoden/Klassen der Standardbibliothek.

In Moodle finden Sie als Vorlage die Klasse `Aufg15_Odd` sowie eine JUnit-Testklasse `JuTest_Aufg15_Odd`.

Aufgabe 1.6

Beweisen Sie mittels **Berechnungsinduktion**, dass für alle Argumente *n* das Resultat der folgenden rekursiven Methode ein String ist, der gleich viele 'a' wie 'b' enthält.

```
public static String myMeth(int n) {
     if (n < 0) {
          return myMeth(-n);
     } else if (n == 0) {
          return "";
     } else if (n < 10) {
          return "bb" + myMeth(3*n) + "aa";
     } else if (n < 40) {
          return myMeth(2*n) + myMeth(n+1);
     } else {
          return "baab";
     }
}
```

Aufgabe 1.7 - Scheinaufgabe (6 P)

Beweisen Sie mittels Berechnungsinduktion, dass folgende Methode `funnyFun(n)` für alle Werte *n* ein Ergebnis kleiner als 5 liefert.

```
public static int funnyFun(int n) {
     if (n < 10) {
          return n - 6;
     } else if (n % 3 == 0) {
          return 4;
     } else if (n > 19) {
          return 10 * funnyFun(n-10) - 47;
     } else {
          return ( funnyFun(n+1) + funnyFun(n/3) ) / 2;
     }
}
```