Informatik I **Wintersemester 2018/19**
Prof. Dr. Carsten Damm Georg-August-Universität Göttingen
Dr. Henrik Brosenne Institut für Informatik

Übungsblatt 08

E-Learning

Absolvieren Sie die Tests bis Di., 18.12., 23:55 Uhr.

Die Tests sind in der Stud.IP-Veranstaltung *Informatik I* unter *Lernmodule* hinterlegt.

Sie können einen Test **nur einmal durchlaufen**. Sobald Sie einen Test starten steht Ihnen nur eine **begrenzte Zeit** zu Verfügung, um den Test zu bearbeiten.

Alle Punkte, die Sie beim Test erreichen, werden ihnen angerechnet.

ILIAS – 10 Punkte

Binäre Suche

Absolvieren Sie den Test *Informatik I - ILIAS 08 Teil 1.*
(10 Punkte)

Hinweis

Wenn Sie den Test einmal vollständig durchlaufen haben kommen Sie auf die Seite *Testergebnisse*. Starten Sie den Test erneut aus Stud.IP, ist jetzt auch eine Schaltfläche *Testergebnisse anzeigen* vorhanden, die auf diese Seite führt.

Auf der Seite *Testergebnisse* können Sie sich unter *Übersicht der Testdurchläufe* zu jedem Testdurchlauf *Details anzeigen* lassen.

In der Auflistung der Aufgaben führt der Titel einer Aufgabe zu einer **Musterlösung** für die jeweilige Aufgabe.

ILIAS 4–10-Minuten-Aufgaben – 12 Punkte

Absolvieren sie die Tests *Informatik I - ILIAS 08 Teil 2* und *Teil 3.*
(12 Punkte)

Übung

Abgabe bis Di., 18.12., 18 Uhr.

Werfen Sie Ihre Lösung in den Zettelkästen Ihrer Gruppenübung. Für die Übungen im Nordbereich stehen die Zettelkästen im Sockelgeschoß (Ebene -1) **oder** auf dem Flur vor dem Seminarraum auf Ebene 0 des Instituts für Informatik.

Wenn Ihre Übung im Südbereich stattfindet, klären Sie mit Ihrem Tutor wo die Lösungen abzugeben sind.

Achten Sie darauf, dass Ihr **Name**, Ihre **Gruppe** und Ihre **Matrikelnummer** auf **jedem** Blatt stehen!

Falls Ihre Lösung mehrere Blätter umfasst, heften Sie diese bitte zusammen.

Aufgabe 1 – 18 Punkte

Rekursion/Iteration

Betrachten Sie folgende Methoden.

```
public static int compute(int n) {
  return computeInternal(n,1,1);
}

private static int computeInternal(int n, int i, int res) {
  if(i == n)
    return res;
  return computeInternal(n, i+1, res+i+i+1);
}
```

1. Betrachten Sie den Aufruf `compute(5)` und erstellen Sie ein rekursives Ablaufprotokoll für den zugehörigen Aufruf der Methode `computeInternal`.
   (8 Punkte)

2. Welche Funktion $f(n)$ berechnet `compute(int n)`?
   (4 Punkte)

3. Formulieren Sie `compute(int n)` iterativ, geben Sie dazu Java-Code für die Methode an. Die Idee zur Berechung von $f(n)$ soll dabei erhalten bleiben.
   (6 Punkte)

Aufgabe 2 – 25 Punkte

Suchen

Formulieren Sie jeweils eine Idee und eine Methode in Java-Code zur Lösung der folgenden speziellen Suchprobleme.

1. Gegeben ist eine Folge ganzer Zahlen $(f_0, f_1, \ldots, f_{n-1})$ mit mindestens einem Folgenglied.

   Gesucht ist ein $i \in \{0, \ldots, n-1\}$ mit $f_i \geq f_j$ für alle $j \in \{0, \ldots, n-1\}$.
   (6 Punkte)

2. Gegeben ist eine Folge ganzer Zahlen $(f_0, f_1, \ldots, f_{n-1})$ mit mindestens einem Folgenglied.

   Gesucht ist ein $i \in \{0, \ldots, n-1\}$ mit $|f_i - f_{((i+1) \mod n)}| \geq |f_j - f_{((j+1) \mod n)}|$ für alle $j \in \{0, \ldots, n-1\}$.
   (8 Punkte)

3. Gegeben ist eine Folge ganzer Zahlen $(f_0, f_1, \ldots, f_{n-1})$ mit mindestens einem Folgenglied.

   Gesucht ist ein $i \in \{0, \ldots, n-1\}$ mit $\sum_{j=0}^{n-1} |f_i - f_j| \leq \sum_{j=0}^{n-1} |f_k - f_j|$ für alle $k \in \{0, \ldots, n-1\}$.
   (11 Punkte)

Hinweis

Sie können sich an Bubblesort und an der Idee für die sequentiellen Suche orientieren.

Praktische Übung

Abgabe der Prüfsumme bis Di., 18.12., 23:55 Uhr.
Testat von Di., 08.01.2019., 8-10 Uhr bis Fr., 11.01.2019, 18-20 Uhr.

Hilfe zum Bearbeiten der praktischen Übungen können Sie grundsätzlich jeden Tag in den Rechnerübungen bekommen, insbesondere in den Rechnerübungen am **Montag**, in denen **keine Testate** stattfinden.

Hinweise zu den praktischen Übungen, insbesondere zur **Abgabe der Prüfsumme** und zur **Durchführung der Testate**, sind in der Stud.IP-Veranstaltung *Informatik I* unter *Dateien→Übungsblätter* hinterlegt.

Aufgabe 1 – 15 Punkte

Omitted Numbers

Die unendlich vielen Studierenden des autonomen Wohnheims *Infinitiy* müssen die Erledigung bestimmter regelmäßiger Aufgaben organisieren, die jeden zweiten, jeden dritten, . . . , jeden k-ten Tag, jeden k+1-ten Tag, . . . anfallen.

Anfangs wird eine Nummerierung 1, 2, 3, 4, . . . der Studierenden festgelegt. Dann wird jeder Zweite (Abzählen beim ersten Studierenden beginnen) Müllbeauftragter (das sind die Studierenden mit den Nummern 2, 4, 6, 8, . . .), von den Übrigen (Studierende 1, 3, 5, 7, . . .) wird jeder Dritte Kühlschrankbeauftragter (das sind die Studierenden 5, 11, 17, 23, . . .), von den Übrigen (Studierenden 1, 3, 7, 9, . . .) jeder Vierte . . . , von den dann noch Übrigen fällt uns für jeden k-ten auch noch irgendetwas ein und von den Übrigen für jeden (k+1)-ten usw. Offenbar werden einige der Bewohner (z.B. die Studierenden 1, 3, 7) bei der Verteilung ausgelassen und müssen keine der anfallenden Aufgaben erledigen. Die Nummern dieser Studierenden nennt man *Omitted Numbers*.

Programmieren Sie eine Applikation `OmittedNumbers`, die genau die Omitted Numbers eines Bereichs $1, \ldots, N$ ermittelt und ausgibt, wobei N auf der Kommandozeile übergeben wird. Verwenden Sie nur Methoden, die iterativ, d.h. ohne sich selbst direkt oder indirekt aufzurufen, arbeiten.

Sie können den Quelltext `https://www.stud.informatik.uni-goettingen.de/info1/java/Eratosthenes.java` benutzen und entsprechend anpassen.

Versehen Sie die Klasse mit ausführlichen Kommentaren, die den Programmablauf erläutern.
(15 Punkte)

Aufgabe 2 – 20 Punkte

Josephus-Permutation

Eine *Permutation* der Zahlen $1, \ldots, N$ ist eine beliebige Anordnung, ohne Wiederholungen und Auslassungen, dieser Zahlen.

Beispiel

`4 1 3 2` ist eine Permutation der Zahlen `1 2 3 4`.

Manchmal ist es sinnvoll die beiden Anordnungen direkt untereinander zu schreiben.

```
1 2 3 4
4 1 3 2
```

Eine *Josephus-Permutation* der Länge N ist eine Permutation der Zahlen $1, \ldots, N$, für deren Bildung noch eine Schrittweite S benötigt wird.

- Die Zahlen werden in einem Kreis angeordnet und die Position der Zahl 1 wird als aktuelle Position gesetzt.
- Bis der Kreis keine Zahlen mehr enthält entfernt man rundenweise jeweils die nächste Zahl aus dem Kreis, die S Schritte von der aktuellen Position entfernt ist, die aktuelle Position wird dabei mitgezählt.
- Die Josephus-Permutation entsteht, indem an die Position einer Zahl, in der ursprünglichen Anordnung `1 ... N`, die Runde eingetragen wird, in der diese Zahl aus dem Kreis entfernt wurde.

Beispiel

Das oben angegebene Beispiel ist eine Josephus-Permutation mit Länge 4, Schrittweite 2 und wird wie folgt erzeugt.

```
Init       1          2          3          4          Runde
[1 2 3 4]  [1 x 3 4]  [1   3 x]  [1   x  ]  [x      ]  Kreis
[       ]  [  1    ]  [  1   2]  [  1 3 2]  [4 1 3 2]  Josephus-Permutation
```

Programmieren Sie eine Applikation `Josephus`, die Länge `N` und Schrittweite `S` auf der Kommandozeile übergeben bekommt und die Josephus-Permutation **rekursiv** berechnet. Die Applikation gibt die Anordnung `1 ... N` und die zugehörige Josephus-Permutation untereinander aus.

Implementieren Sie eine Methode

```
int[] josephusPermutation(int length, int step)
```

die Länge und Schrittweite übergeben bekommt und eine Feld für die Josephus-Permutation erzeugt und initialisiert. Dieses Feld wird von einer rekursiven Methode

```
void josephusRecursive(int[] circle, int last, int step)
```

manipuliert und mit der Josephus-Permutation gefüllt.

Beachten Sie Folgendes.

- Die Methode `josephusPermutation` initialisiert alle Feldelemente mit einem passenden Startwert s, wobei $s < 1$ oder $s > N$ gilt.
- Das Feldelement `circle[k]` – nicht der darin gespeicherte Wert – repräsentiert die Zahl `k+1`.
- Aus den im Feld `circle` gespeicherten Werten kann man den aktuellen Kreis und die bereits berechneten Teile der Josephus-Permutation ablesen.
  - Genau dann, wenn der Wert von `circle[k]` gleich dem Startwert ist, befindet sich die Zahl `k+1` noch im Kreis.
  - Sonst gibt der Wert von `circle[k]` an in welcher Runde $(1, \ldots, N)$ die Zahl `k+1` aus dem Kreis entfernt wurde.
- Die Methode `josephusRecursive` entfernt eine Zahl aus dem Kreis, indem der Wert des Feldelements, das diese Zahl repräsentiert, mit der Runde überschrieben wird, in der die Zahl aus dem Kreis entfernt wurde.
- Mit Ausnahme des ersten Aufrufs von `josephusRecursive` gibt der übergebene Wert `last` an, das zuletzt in der Runde `circle[last]` die Zahl `last+1` aus dem Kreis entfernt wurde, d.h. gilt `circle[last] == circle.length` ist die Josephus-Permutation vollständig berechnet.
- Versehen Sie die Klasse mit ausführlichen Kommentaren, die den Programmablauf erläutern.

(20 Punkte)

Hinweis
Bei *WolframMathWorld* finden Sie im Artikel

`http://mathworld.wolfram.com/JosephusProblem.html`

eine Motivation für den Namen der Permutation und auch einige Beispiele.