Prof. Dr. E. Ábrahám

F. Corzilius, S. Schupp, T. Ströder

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2 bis 3 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Montag, den 19.05.2014 um 9:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch in Ihrem Tutorium vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor abgeben.
- Sofern nicht näher spezifiziert, steht die Notation log ohne Angabe einer Basis für den natürlichen Logarithmus.
- Wenn Sie in einer Aufgabe einen **Algorithmus** erstellen müssen, so soll dieser in **Pseudo-Code** (gerne an `Java` bzw. `C++` orientiert) oder in `Java` bzw. `C++` verfasst werden. Reichen Sie den Algorithmus **nur in Maschinenschrift** ein, entweder Ihrem Tutor vor dem Abgabedatum per Email oder ausgedruckt an die Lösung geheftet. Einfache Syntaxfehler (z. B. ein fehlendes Semikolon) geben keine Punktabzüge. Inwieweit im Pseudo-Code abstrahiert werden darf, wird in der Aufgabenstellung bekannt gegeben. Ansonsten gilt es, sich möglichst nahe an den Möglichkeiten, die `Java` bzw. `C++` bieten, zu orientieren.
- Wir nennen die Folge `int tmp = E[i]; E[i] = E[j]; E[j] = tmp;` an Anweisungen für ein Array `E` und zwei Array-Indizes `i` und `j` eine Swap-Operation.

Tutoraufgabe 1 (Sortieren):

a) Sortieren Sie das folgende Array durch Anwendung des Selectionsort-Algorithmus. Geben Sie dazu das Array nach jeder Swap-Operation an.

```
[ 7,3,6,2,1 ]
```

b) Sortieren Sie das folgende Array durch Anwendung des Insertionsort-Algorithmus. Geben Sie dazu das Array nach jeder Iteration der äußersten Schleife an.

```
[ 7,2,3,6,1 ]
```

c) Sortieren Sie das folgende Array durch Anwendung des Mergesort-Algorithmus. Geben Sie dazu das Eingabe-Array nach jeder Merge-Operation an.

```
[ 4,7,2,9,1,8 ]
```

Aufgabe 2 (Sortieren): (3 + 3 + 8 = 14 Punkte)

a) Sortieren Sie das folgende Array durch Anwendung des Bubblesort-Algorithmus. Geben Sie dazu das Array nach jeder Swap-Operation an.

```
[ 6,2,3,7,1 ]
```

b) Sortieren Sie das folgende Array durch Anwendung des Quicksort-Algorithmus. Geben Sie dazu das Array nach jeder Partition-Operation an und markieren Sie das für die jeweilige Partitionierung genutzte Pivot-Element.

```
[ 4,7,2,8,9,6,1,3 ]
```

c) Sortieren Sie das folgende Array durch Anwendung des Heapsort-Algorithmus. Geben Sie dazu das Array nach jeder Swap-Operation an.

```
[ 4,7,2,3,9,6,1 ]
```

Tutoraufgabe 3 (Best- und Worstcase):

a) Geben Sie für ein beliebiges, aber festes $n \in \mathbb{N}$ ein Array der Länge n an, sodass der Bubblesort-Algorithmus so *viele* Swap-Operationen wie möglich benötigt (Worst-Case) und zusätzlich ein Array der Länge n, sodass der Bubblesort-Algorithmus so *wenig* Swap-Operationen wie möglich benötigt (Best-Case). Begründen Sie Ihre Antwort kurz.

b) Geben Sie für ein beliebiges, aber festes $n \in \mathbb{N}$ ein Array der Länge n an, sodass der Quicksort-Algorithmus mit der Pivot-Strategie aus der Vorlesung (als Pivot-Element wird immer das letzte Element des betrachteten Array-Bereichs gewählt) so *viele* Partition-Operationen wie möglich benötigt (Worst-Case). Begründen Sie Ihre Antwort kurz.

c) Geben Sie für ein beliebiges, aber festes $n \in \mathbb{N}$ ein Array der Länge n an, sodass der Heapsort-Algorithmus so *wenig* Swap-Operationen wie möglich benötigt (Best-Case). Begründen Sie Ihre Antwort kurz.

Aufgabe 4 (Best- und Worstcase): $(1 + 4 + 2 + 2^* = 7 + 2^*$ Punkte)

a) Geben Sie für ein beliebiges, aber festes $n \in \mathbb{N}$ ein Array der Länge n an, sodass der Selectionsort-Algorithmus so *viele* Swap-Operationen wie möglich benötigt (Worst-Case) und zusätzlich ein Array der Länge n, sodass der Selectionsort-Algorithmus so *wenig* Swap-Operationen wie möglich benötigt (Best-Case). Begründen Sie Ihre Antwort kurz.

b) Geben Sie für ein beliebiges, aber festes $n \in \mathbb{N}$ ein Array der Länge n an, sodass der Insertionsort-Algorithmus so *viele* Vergleiche wie möglich benötigt (Worst-Case) und zusätzlich ein Array der Länge n, sodass der Insertionsort-Algorithmus so *wenig* Vergleiche wie möglich benötigt (Best-Case). Begründen Sie Ihre Antwort kurz.

c) Geben Sie für ein beliebiges, aber festes $n \in \mathbb{N}$ ein Array der Länge n an, sodass der Mergesort-Algorithmus so *wenig* Vergleiche wie möglich benötigt (Best-Case). Begründen Sie Ihre Antwort kurz.

d)* Geben Sie für ein beliebiges, aber festes $n \in \mathbb{N}$ ein Array der Länge n an, sodass der Mergesort-Algorithmusso *viele* Vergleiche wie möglich benötigt (Worst-Case). Begründen Sie Ihre Antwort kurz.

Tutoraufgabe 5 (Iteratives Sortieren):

Implementieren Sie für den Sortieralgorithmus `void mergeSort(int[] E, int left, int right)` eine iterative Variante `void mergeSortIter(int[] E, int left, int right)` mit einer Zeitkomplexität in $\mathcal{O}(n \log n)$, wobei Sie für Schreibzugriffe nur die Operation `merge(int[] E, int left, int mid, int right)` verwenden dürfen.
Nutzen Sie die Datenstruktur `Stack` mit den Operationen `void push(int v)`, `int pop()` und `bool isEmpty()`, um Rekursionen aufzulösen.
(`void push(int v)` schiebt `v` auf den Stack, `int pop()` entfernt den zuletzt eingefügten Wert vom Stack und gibt ihn zurück, `bool isEmpty()` gibt zurück, ob der Stack leer ist.)

Aufgabe 6 (Iteratives Sortieren): (6 Punkte)

Implementieren Sie für den Sortieralgorithmus `void quickSort(int[] E, int left, int right)` eine iterative Variante `void quickSortIter(int[] E, int left, int right)` mit einer Zeitkomplexität in $\mathcal{O}(n^2)$, wobei Sie für Schreibzugriffe nur die Operation `swap(int[] E, int i, int j)` (definiert als `int tmp = E[i]; E[i] = E[j]; E[j] = tmp;`) verwenden dürfen.
Nutzen Sie die Datenstruktur `Stack` mit den Operationen `void push(int v)`, `int pop()` und `bool isEmpty()`, um Rekursionen aufzulösen. (`void push(int v)` schiebt `v` auf den Stack, `int pop()` entfernt den zuletzt eingefügten Wert vom Stack und gibt ihn zurück, `bool isEmpty()` gibt zurück, ob der Stack leer ist.)

Bonusaufgabe 7 (Algorithmenanalyse*): (2*+4*=6* Punkte)

Gegeben sei folgender Algorithmus:

```
1  int foo(int[] E, int k) {
2      int left = 0;
3      List<int> m; // generiere eine leere Liste
4      while (left < E.length()) { // fuer jede Teilsequenz der Laenge 5
5          int right = left + 5;
6          if (right > E.length()) right = E.length();
7          int[] tmp = copyOfRange(E, left, right);  // kopiere E[left..right-1] nach tmp
8          sort(tmp); // sortiere tmp mit Insertionsort
9          if (E.length() <= 5) return tmp[k-1];
10         // nehme das mittlere Element aus tmp
11         // roundDown rundet zur naechsten ganzen Zahl ab
12         // add fuegt m das gegebene Element hinten an
13         m.add(tmp[roundDown(tmp.length()/2)]);
14         left = right;
15     }
16     // toArray erstellt zu einer Liste ein Array mit der gleichen Sequenz
17     p = foo(toArray(m), roundDown(m.length()/2));
18     List<int> E1, E2;
19     for (int i=0; i<E.length(); i++) {
20         if (i < p) E1.add(E[i]);
21         else if (i > p) E2.add(E[i]);
22     }
23     if (E1.length() + 1 == k) return p;
24     else if (E1.length() >= k) return foo(toArray(E1), k);
25     else return foo(toArray(E2), (k - E1.length() - 1));
26 }
```

a) Beschreiben Sie den Rückgabewert von `foo(E, k)`, falls `E` ein nicht leeres Array von paarweise verschiedenen ganzen Zahlen ist und `k` eine positive (> 0) ganze Zahl kleiner oder gleich der Länge von `E` ist. Begründen Sie Ihre Antwort!

b) Geben Sie die Rekursionsgleichungen $T(n,\ k)$ für die Worst-Case Laufzeit von `foo(E, k)` für den Fall an, dass $n =$ `E.length()` und $k =$ `k`. Dabei sind die elementaren Operationen `+`, `-`, `/`, `roundDown(..)`, `add(..)`, `length(..)` $\in \mathcal{O}(1)$. Außerdem ist `toArray(E)` $\in \mathcal{O}($`E.length()`$)$, `copyOfRange(E, b, r)` $\in \mathcal{O}(r - b)$ und `sort(tmp)` $\in \mathcal{O}((\texttt{tmp.length()})^2)$. Beachten Sie, dass konstante Aufwände, die nicht in Verbindung mit einem rekursiven Aufruf stehen, durch eine Konstante c zusammengefasst werden können.