Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Ulf Leser
M. Bux, B. Grußien, L. Heimberg, P. Schäfer

Berlin, den 22.05.2017

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Übungsblatt 3

Abgabe: Mittwoch den 07.06.2017 bis 11:10 Uhr vor der Vorlesung im Hörsaal oder bis 10:45 Uhr im Briefkasten neben Raum 3.321, RUD25.

Die Übungsblätter sind in Gruppen von zwei (in Ausnahmen drei) Personen zu bearbeiten. Jedes Übungsblatt muss bearbeitet werden. (Sie müssen mindestens ein Blatt für wenigstens eine Aufgabe jedes Übungsblattes abgeben.) Die Lösungen sind auf nach Aufgaben getrennten Blättern abzugeben. Heften Sie bitte die zu einer Aufgabe gehörenden Blätter vor der Abgabe zusammen. Vermerken Sie auf allen Abgaben die Namen und **CMS-Benutzernamen** aller Gruppenmitglieder, Ihre Abgabegruppe (z.B. AG123) aus Moodle, und den Übungstermin (z.B. Di 13 Uhr bei Marc Bux), zu dem Sie Ihre korrigierten Blätter zurückerhalten werden.

Beachten Sie die Informationen auf der Übungswebseite (https://hu.berlin/algodat17).

Konventionen:

- Die Indizierung aller Arrays auf diesem Blatt beginnt bei 1. Bitte beginnen Sie die Indizierung der Arrays in Ihren Lösungen auch bei 1.

Aufgabe 1 (Schreibtischtests) — 4 + 4 + 4 = 12 Punkte

In den folgenden Teilaufgaben sollen Sie jeweils einen *Schreibtischtest* für ein Sortierverfahren durchführen. Das heißt, Sie führen auf Papier einen gegebenen Algorithmus für gegebene Eingaben aus. In der jeweiligen Teilaufgabe steht, welche Zwischenschritte Sie als Lösung einreichen sollen. Notieren Sie ein Array als Liste in eckigen Klammern (also in der Form $[a_1, \ldots, a_n]$).

a) Führen Sie einen Schreibtischtest für den Algorithmus **Mergesort** aus der VL für das Eingabe-Array

$$A = [w,\ b,\ x,\ a,\ g,\ c,\ h,\ e,\ r]$$

durch, wobei Sie als Ordnung die alphabetische Ordnung auf den Buchstaben annehmen.

Geben Sie die Zwischenschritte in Form eines Graphen wie in der VL (Folie 7) an.

b) Führen Sie einen Schreibtischtest für den Algorithmus **Quicksort** aus der VL für das Eingabe-Array

$$A = [8,\ 6,\ 2,\ 7,\ 1,\ 4,\ 3,\ 5]$$

durch, wobei Sie als Ordnung die natürliche Ordnung auf den natürlichen Zahlen annehmen.

Als Pivot-Element wählen Sie das am weitesten rechts stehende Element des aktuellen Teil-Arrays.

Geben Sie den aktuellen Wert von A nach jeder Swap-Operation (Folie 26, Zeilen 14, 17) an. Unterstreichen Sie jeweils das in diesem Aufruf von **divide**(A, l, r) betrachtete Teil-Array $A[l..r]$.

c) In dieser Teilaufgabe nutzen wir den Algorithmus **Bucketsort**, um Arrays von Zeichenketten gleicher Länge über einem festen endlichen Alphabet Σ zu sortieren. Wir nehmen also an, dass die Einträge des übergebenen Arrays alles Elemente von Σ^m für eine bestimmte Zahl $m \in \mathbb{N}_{>0}$ sind, wobei Σ^m wie üblich die Menge aller Zeichenketten über Σ der Länge m ist. Wir nehmen weiterhin an, dass das Alphabet Σ linear geordnet ist, und dass die Ordnung auf den Zeichenketten die *lexikographische* Ordnung ist. Es gilt also: $a_1 \dots a_m < b_1 \dots b_m$ g.d.w. ein i mit $1 \leq i \leq m$ existiert, so dass $a_i < b_i$ und $a_j = b_j$ für alle $j < i$.

Führen Sie einen Schreibtischtest für $\Sigma = \{0, 1, 2, 3\}$, $m = 3$, und das Array

$$A \;=\; [203,\ 202,\ 100,\ 123,\ 121,\ 323,\ 103,\ 211,\ 320]$$

durch.

Notieren Sie nach jedem Durchlauf der Schleife mit Laufvariable i den Inhalt des Arrays A. Markieren Sie wie in der VL (Folie 23) mit vertikalen Strichen, welche Elemente von A sich in dieser Iteration gemeinsam in einem Bucket befanden.

Aufgabe 2 (Stabilität) **3 + 3 + 3 + 3 + 3 = 15 Punkte**

In dieser Aufgabe sortieren wir Arrays mit Einträgen des abstrakten Datentyps *Element*. Ein Element e hat einen *Schlüssel* e.key aus einer Menge $\mathbb{K}$ und einen *Wert* e.val aus einer Menge $\mathbb{V}$. Elemente werden anhand ihrer Schlüssel sortiert. Dazu ist auf der Menge $\mathbb{K}$ der Schlüssel eine lineare Ordnung $\leq$ definiert. Für beliebige Elemente e_1 und e_2 schreiben wir

$$e_1 \;\leq\; e_2 \quad \text{genau dann, wenn} \quad e_1.\text{key} \;\leq\; e_2.\text{key}.$$

Ein Sortierverfahren heißt *stabil*, wenn Elemente mit gleichen Schlüsseln nach der Sortierung in der gleichen Reihenfolge aufeinander folgen wie vor der Sortierung.

Beispiel: Wir notieren ein Element e auch als Paar $(e.\text{key}, e.\text{val})$. Sei $\mathbb{K} = \mathbb{N}$ und $\mathbb{V} = \{a, b, c\}$. Ein Sortierverfahren, welches bei Eingabe des Arrays $[(3, a), (1, c), (1, b)]$ das sortierte Array $[(1, b), (1, c), (3, a)]$ ausgibt, ist nicht stabil.

In der Vorlesung haben Sie verschiedene Sortierverfahren kennengelernt. (Den Pseudocode für **Bubblesort** finden Sie auf diesem Übungsblatt.) Entscheiden Sie, ob die folgenden Verfahren stabil sind. Falls das Verfahren Ihrer Meinung nach nicht stabil ist, geben Sie eine (bitte möglichst kleine) Instanz als Gegenbeispiel an, andernfalls begründen Sie, weshalb das Verfahren stabil ist.

a) **Bubblesort**

b) **Insertionsort**

c) **Quicksort** mit dem letzten Element im (Teil-)Array als Pivotelement

d) **Mergesort**

e) **Bucketsort**

Bubblesort(Array A)

Input: Array A von n Elementen.
Output: Array A aufsteigend sortiert.

```
1:  repeat
2:      swapped := false
3:      for i := 1 to n − 1 do
4:          if A[i] > A[i + 1] then
5:              temp := A[i]
6:              A[i] := A[i + 1]
7:              A[i + 1] := temp
8:              swapped := true
9:          end if
10:     end for
11: until not swapped
```

Aufgabe 3 (Sortierung spezieller Arrays) **3 + 3 + 3 + 3 + 3 = 15 Punkte**

Ein *allgemeines Sortierverfahren* ist ein Sortierverfahren, welches die zu sortierenden Elemente nur vergleichen kann und sonst keinerlei Eigenschaften der Elemente ausnutzt.
Beweisen oder widerlegen Sie: Es gibt ein allgemeines Sortierverfahren, welches ein Array von n beliebigen (in konstanter Zeit vergleichbaren) Elementen im Worst Case in Laufzeit $\mathcal{O}(n)$ sortiert, falls...

a) ... 50% aller Elemente im Array gleich sind.

b) ... die erste Hälfte des Array bereits aufsteigend und die zweite Hälfte des Array bereits absteigend sortiert ist.

c) ... die Länge des Array durch 10 teilbar ist, und die ersten 10 Elemente aufsteigend sortiert sind, die zweiten 10 Elemente aufsteigend sortiert sind, usw.

d) ... das Array so vorsortiert ist, dass alle Elemente der ersten Hälfte des Arrays kleiner sind als alle Elemente der zweiten Hälfte.

e) ... im Array nur höchstens $l \in \mathbb{N}_{>0}$ viele unterschiedliche Elemente vorkommen. Hierbei sei l eine Konstante.

Hinweise: Für den Fall, dass es ein solches allgemeines Sortierverfahren gibt, können Sie als Beweis die Idee eines konkreten Verfahrens beschreiben. Falls kein solches allgemeines Sortierverfahren existiert, können Sie die in der Vorlesung gezeigte untere Schranke für allgemeine Sortierverfahren nutzen.

Aufgabe 4 (Allgemeine Sortierverfahren) **8 Punkte**

Sei A ein Array mit n unterschiedlichen Elementen. Es gibt genau $n!$ unterschiedliche Möglichkeiten, wie die Elemente in A angeordnet sind. Jede der $n!$ Permutationen ist eine mögliche Eingabe für ein allgemeines Sortierverfahren.

a) Zeigen Sie, dass es kein allgemeines Sortierverfahren gibt, welches für mindestens die Hälfte aller $n!$ vielen Eingaben eine Laufzeit in $\mathcal{O}(n)$ hat.

b) Beweisen oder widerlegen Sie, dass es ein allgemeines Sortierverfahren gibt, welches immerhin für einen Anteil von $\frac{1}{2^n}$ aller $n!$ vielen Eingaben in Zeit $\mathcal{O}(n)$ sortiert.

Hinweis: Überlegen Sie sich analog zu dem Beweis der unteren Schranke für allgemeine Sortierverfahren, wie ein Entscheidungsbaum in den jeweiligen Fällen aussehen muss.