Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Stefan Kratsch
Dr. K. Ahrens, F. Hegerfeld, F. Nelles, Dr. P. Schäfer

Berlin, den 14.05.2018

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Übungsblatt 3

Abgabe: Montag den 28.05.2018 bis 11:10 Uhr vor der Vorlesung im Hörsaal oder bis 10:45 Uhr im Briefkasten (RUD 25, Raum 3.321). Die Übungsblätter sind in Gruppen von 2 Personen zu bearbeiten. 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 Ihre Namen, Ihre **CMS-Benutzernamen**, Ihre Abgabegruppe (z.B. AG123) aus Moodle, und Ihren Übungstermin (z.B. Do 13 Uhr bei Florian Nelles), zu dem Sie Ihre korrigierten Blätter zurückerhalten werden.
Beachten Sie die Informationen auf der Übungswebseite (https://hu.berlin/algodat18).

Konventionen:

- Für ein Array A ist $|A|$ die Länge von A, also die Anzahl der Elemente in A. Die Indizierung aller Arrays auf diesem Blatt beginnt bei 1 (und endet also bei $|A|$).
- Mit der Aufforderung "Analysieren Sie die Laufzeit" ist gemeint, dass Sie eine möglichst gute obere Schranke der Zeitkomplexität angeben und diese begründen sollen.

Aufgabe 1 (Schreibtischtests) — 3 + 3 + 3 + 4 = 13 Punkte

In den folgenden Unteraufgaben sollen Sie jeweils einen *Schreibtischtest* durchführen. Das heißt, Sie führen auf Papier einen gegebenen Algorithmus für gegebene Eingaben aus. In der Unteraufgabe steht, welche Zwischenergebnisse Sie als Lösung einreichen sollen. Notieren Sie ein Array entweder als Liste in eckigen Klammern ($[a_1, \ldots, a_n]$) oder als Tabelle der Form | a_1 | $\ldots$ | a_n |.

a) Führen Sie einen Schreibtischtest für den Algorithmus **InsertionSort** aus der VL für das Eingabe-Array $A = [7, 5, 1, 4, 9, 6, 3, 8, 2]$ durch, wobei Sie als Ordnung die natürliche Ordnung auf den natürlichen Zahlen annehmen. Geben Sie Zwischenergebnisse jeweils nach Ausführung der Zeile 8 an.

InsertionSort(Array A)

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

```
1: for i = 2 to n do
2:     new_value = A[i];
3:     j = i;
4:     while j > 1 and new_value < A[j − 1] do
5:         A[j] = A[j − 1];
6:         j = j − 1;
7:     end while
8:     A[j] = new_value;
9: end for
```

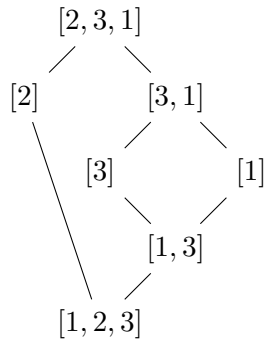
b) Führen Sie einen Schreibtischtest für den Algorithmus **MergeSort** aus der VL für das Eingabe-Array $A = [\mathrm{i}, \mathrm{n}, \mathrm{d}, \mathrm{k}, \mathrm{q}, \mathrm{b}, \mathrm{f}, \mathrm{v}, \mathrm{m}]$ durch, wobei Sie als Ordnung die alphabetische Ordnung auf den Buchstaben annehmen. Geben Sie die Zwischenergebnisse in Form eines Graphen an (siehe Beispiel für $A = [2, 3, 1]$ unten).

MergeSort(Array A)

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

1: **if** $n > 1$ **then**
2: $s = \lfloor \frac{n}{2} \rfloor$;
3: MergeSort($A[1, \ldots, s]$);
4: MergeSort($A[s+1, \ldots, n]$);
5: A = Merge($A[1, \ldots, s]$,$A[s+1, \ldots, n]$);
6: **end if**

Schreibtischtest für $A = [2, 3, 1]$:



Merge(Array A_1, A_2)

Input: Array A_1 von x Elementen, aufsteigend sortiert.
Array A_2 von y Elementen, aufsteigend sortiert.
Output: Array B aus Elementen von A_1 und A_2, aufsteigend sortiert.

1: B: Array der Länge $n = x + y$;
2: $k = i = j = 1$;
3: **while** $i \leq x$ and $j \leq y$ **do**
4: **if** $A_1[i] \leq A_2[j]$ **then**
5: $B[k] = A_1[i]$;
6: $i = i + 1$;
7: **else**
8: $B[k] = A_2[j]$;
9: $j = j + 1$;
10: **end if**
11: $k = k + 1$;
12: **end while**
13: **if** $j \leq y$ **then**
14: $B[k, \ldots, n] = A_2[j, \ldots, y]$;
15: **else if** $i \leq x$ **then**
16: $B[k, \ldots, n] = A_1[i, \ldots, x]$;
17: **end if**
18: **return** B

c) Führen Sie einen Schreibtischtest für den Algorithmus **QuickSort** aus der VL für das Eingabe-Array $A = [2, 10, 6, 7, 13, 4, 1, 12, 5, 9]$ 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 die aktuelle Belegung von A nach jeder Swap-Operation (Zeile 13,17,20) an. Unterstreichen Sie jeweils das in diesem Aufruf betrachtete Teil-Array.

QuickSort(Array A, Integer ℓ, r)

Input: Array A von n Elementen.
Zahlen ℓ und r mit $1 \leq \ell, r \leq n$.
Output: Array A mit $A[\ell, \ldots, r]$ sortiert.

```
1: if r > ℓ then
2:     pos = Divide(A, ℓ, r);
3:     QuickSort(A, ℓ, pos − 1);
4:     QuickSort(A, pos + 1, r);
5: end if
```

Divide(Array A, Integer ℓ, r)

Input: Array A von n Elementen.
Zahlen ℓ und r mit $1 \leq \ell \leq r \leq n$.
Output: Array A und Index k mit $A[i] \leq A[k]$ für $\ell \leq i \leq k$ und $A[j] \geq A[k]$ für $k \leq j \leq r$.

```
1: k =ChoosePivot(A, ℓ, r);
2: p = A[k];
3: i = ℓ;
4: j = r;
5: while i < j do
6:     while A[i] ≤ p and i < r do
7:         i = i + 1;
8:     end while
9:     while A[j] ≥ p and j > ℓ do
10:        j = j − 1;
11:    end while
12:    if i < j then
13:        Swap(A[i], A[j]);
14:    end if
15: end while
16: if k > i then
17:    Swap(A[i], A[k]);
18:    return i;
19: else if k < j then
20:    Swap(A[j], A[k]);
21:    return j;
22: end if
23: return k;
```

d) Führen Sie einen Schreibtischtest für den Algorithmus **CountingSort** für das Eingabe-Array $A = [2, 4, 2, 1, 5, 2, 5, 2]$ durch, wobei Sie als Ordnung die natürliche Ordnung auf den natürlichen Zahlen annehmen. Geben Sie die aktuelle Belegung von C sowohl nach der ersten als auch nach der zweiten for-Schleife an. Geben Sie außerdem die aktuellen Belegungen von B und C nach jeder Iteration der dritten for-Schleife an.

CountingSort(Array A, Integer m)

Input: Array A von n natürlichen Zahlen aus $\{1, \ldots, m\}$.
Output: Array B aus den Zahlen von A, aufsteigend sortiert .

```
1: initiiere Array C der Länge m (alle Werte auf 0);
2: initiiere Array B der Länge n (alle Werte auf 0);
3: for i = 1 to n do
4:     C[A[i]] = C[A[i]] + 1;
5: end for
6: for i = 2 to m do
7:     C[i] = C[i] + C[i − 1];
8: end for
9: for i = n down to 1 do
10:    B[C[A[i]]] = A[i];
11:    C[A[i]] = C[A[i]] − 1;
12: end for
13: return B;
```

Aufgabe 2 (Stabilität) **3 + 3 + 3 + 3 = 12 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 unten.) 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) **InsertionSort**

b) **BubbleSort**

c) **MergeSort**

d) **QuickSort** mit dem letzten Element im (Teil-)Array als Pivotelement

Bubblesort(Array A)

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

```
1:  swap = TRUE;
2:  while swap do
3:     swap = FALSE;
4:     for i = 1 to n − 1 do
5:        if A[i] > A[i + 1] then
6:           Swap(A[i], A[i + 1]);
7:           swap = TRUE;
8:        end if
9:     end for
10: end while
```

Aufgabe 3 (Listen) **8 + 2 = 10 Punkte**

Gegeben seien zwei einfach verkettete Listen A und B, die mindestens einen gemeinsamen Knoten enthalten (vgl. Abbildung 1).

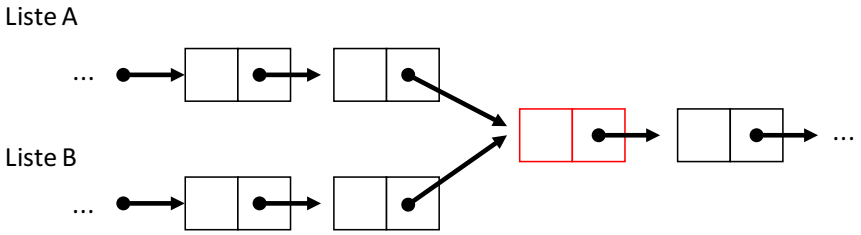


Abbildung 1: Zwei einfach verkettete Listen A und B. Es existiert ein Knoten (rot markiert), ab dem die Listen übereinstimmen.

1. Erläutern Sie, wie Sie den ersten gemeinsamen Knoten finden können, ab dem die beiden einfach verketteten Listen übereinstimmen. Beachten Sie, dass die beiden Listen nicht die gleiche Länge haben müssen. Sie können davon ausgehen, dass die beiden Listen keine Zyklen enthalten. (Es ist nicht notwendig und nicht ausreichend hier Pseudocode anzugeben.).

   Hinweis: Die Punktevergabe erfolgt gestaffelt nach Effizienz bezüglich der Laufzeit ihrer Lösung. Es dürfen keine weiteren Datenstrukturen und außer den beiden Listen ausschließlich primitive Datentypen verwendet werden.

2. Analysieren Sie die Laufzeit Ihres vorgestellten Algorithmus.

Aufgabe 4 (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) ...die ersten 90% aller Elemente im Array in sortierter Reihenfolge vorliegen.

b) ...die Elemente sind vom Anfang des Feldes beginnend aufsteigend sortiert bis zu einem maximalen Element und ab diesem bis zum Ende des Feldes absteigend sortiert.

c) ... die Länge des Array durch 10 teilbar ist, und jeweils 10 aufeinanderfolgende Elemente (vom Anfang beginnend) aufsteigend sortiert sind.

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.