Fachbereich Mathematik & Informatik
Freie Universität Berlin
Prof. Dr. Ralf Kornhuber, Prof. Dr. Christof Schütte, Lasse Hinrichsen

9. Übung zur Vorlesung

Computerorientierte Mathematik I

WS 2020/2021

`http://numerik.mi.fu-berlin.de/wiki/WS_2020/CoMaI.php`

Abgabe: Do., 4. Februar 2021, 12:15 Uhr

1. Aufgabe (4 TP)
Sei $\mathcal{M}$ eine Menge, auf der eine Ordnungsrelation $\leq$ definiert ist. Für ein Tupel $(x_1, \ldots, x_N) \in \mathcal{M}^N$ ist eine Abbildung $\pi : \{1, \ldots, N\} \to \{1, \ldots, N\}$ ein Sortieralgorithmus, wenn π bijektiv ist und

$$x_{\pi(i)} \leq x_{\pi(j)} \quad (i \leq j)$$

gilt. Das Tupel $(x_1, \ldots, x_N)$ wird also in das sortierte Tupel $\left(x_{\pi(1)}, \ldots, x_{\pi(N)}\right)$ überführt. Ein Sortieralgorithmus heißt *stabil*, wenn aus

$$x_i \leq x_j \wedge x_j \leq x_i \quad (i < j)$$

(wir schreiben in der Regel $x_i = x_j$) folgt, dass

$$\pi^{-1}(i) < \pi^{-1}(j)$$

gilt. Die relative Ordnung gleicher Elemente bleibt also erhalten.

- Zeigen Sie, dass der *Mergesort*-Algorithmus (aus der Vorlesung bekannt) stabil ist.

2. Aufgabe (8 PP)

a) Implementieren Sie eine Funktion

`mergesort(x)`

in Python, die einen Vektor `x` von Zahlen erhält und sie mittels *Mergesort* sortiert. Der Rückgabewert sei ein Tupel `(x_sortiert, n_vergleiche)`, wobei `x_sortiert` der sortierte Vektor und `n_vergleiche` die Anzahl der angewandten Vergleichsoperationen sei.

b) Analog zur vorherigen Unteraufgabe, implementieren Sie das sortieren mittels *Bubblesort* als

`bubblesort(x)`

in Python.

c) Konstruieren Sie jeweils 100 Listen von Zufallszahlen aus dem Intervall $[0, 1]$ der Länge $N = 10^k$, $k = 1, \ldots, 4$, und wenden Sie darauf die beiden Sortieralgorithmen an. Protokollieren Sie für jedes N jeweils den kleinsten Aufwand, den größten Aufwand und den durchschnittlichen Aufwand des jeweiligen Algorithmus. Plotten Sie mit geeigneter Skalierung diese Werte gegen N.

3. Aufgabe (4 Bonus TP + 4 Bonus PP)
Der *Quicksort*-Algorithmus für eine Liste $x = (x_1, \ldots, x_n)$ lässt sich grob erklären durch

i) Wähle ein Pivotelement $\tilde{x} \in x$. Beispielsweise

$$\tilde{x} = x_1.$$

ii) Partitioniere die Liste x in drei Teillisten, so dass gilt:
- Alle Elemente der ersten Teilliste sind kleiner als das Pivotelement.
- Die zweite Teilliste enhält nur das Pivotelement.
- Alle Elemente der dritten Teilliste sind größer (oder gleich) als das Pivotelement.

iii) Wende Quicksort rekursiv auf die Teillisten an.

Für eine detailiertere Einführung des Algorithmus können Sie natürlich auf externe Quellen zurückgreifen.

a) Zeigen Sie, dass die Partionierung der Liste gemäß ii) in $\mathcal{O}(n)$ möglich ist.

b) Konstruieren Sie einen Fall in dem durch ungünstige Wahl des Pivotelements die Laufzeit des Algorithmus $\mathcal{O}(n^2)$ ist.

c) Implementieren Sie den *Quicksort*-Algorithmus als

`quicksort(x)`

mit den gleichen Rückgabewerten wie in Aufgabe 2). Testen Sie Ihre Implementierung wie in Teilaufgabe 2c) und plotten Sie Ihre Ergebnisse entsprechend.

4. Bonusaufgabe (Quiz) (1 Bonus TP/PP)
Formulieren Sie eine Frage zur Vorlesung. Falls Sie die Antwort wissen, geben Sie die richtige Antwort und 3 falsche Antwortmöglichkeiten an.

Allgemeine Hinweise

Die Punkte unterteilen sich in Theoriepunkte (TP) und Programmierpunkte (PP). Bitte beachten Sie die auf der Vorlesungshomepage angegebenen Hinweise zur Bearbeitung und Abgabe der Übungszettel, insbesondere der Programmieraufgaben.

Von: Sina Irmak
Jonny Lam
Thore Brehmer

1. Aufgabe (4 TP)

Sei $\mathcal{M}$ eine Menge, auf der eine Ordnungsrelation $\leq$ definiert ist. Für ein Tupel $(x_1, \ldots, x_N) \in \mathcal{M}^N$ ist eine Abbildung $\pi : \{1, \ldots, N\} \to \{1, \ldots, N\}$ ein Sortieralgorithmus, wenn π bijektiv ist und

$$x_{\pi(i)} \leq x_{\pi(j)} \quad (i \leq j)$$

gilt. Das Tupel $(x_1, \ldots, x_N)$ wird also in das sortierte Tupel $\left(x_{\pi(1)}, \ldots, x_{\pi(N)}\right)$ überführt. Ein Sortieralgorithmus heißt *stabil*, wenn aus

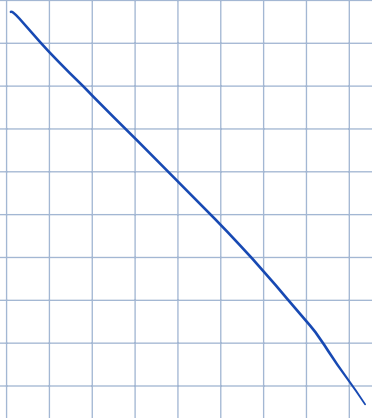
$$x_i \leq x_j \wedge x_j \leq x_i \quad (i < j)$$

(wir schreiben in der Regel $x_i = x_j$) folgt, dass

$$\pi^{-1}(i) < \pi^{-1}(j)$$

gilt. Die relative Ordnung gleicher Elemente bleibt also erhalten.

- Zeigen Sie, dass der *Mergesort*-Algorithmus (aus der Vorlesung bekannt) stabil ist.



2. Aufgabe (8 PP)

a) Implementieren Sie eine Funktion

`mergesort(x)`

in Python, die einen Vektor `x` von Zahlen erhält und sie mittels *Mergesort* sortiert. Der Rückgabewert sei ein Tupel `(x_sortiert, n_vergleiche)`, wobei `x_sortiert` der sortierte Vektor und `n_vergleiche` die Anzahl der angewandten Vergleichsoperationen sei.

b) Analog zur vorherigen Unteraufgabe, implementieren Sie das sortieren mittels *Bubblesort* als

`bubblesort(x)`

in Python.

c) Konstruieren Sie jeweils 100 Listen von Zufallszahlen aus dem Intervall $[0, 1]$ der Länge $N = 10^k$, $k = 1, \ldots, 4$, und wenden Sie darauf die beiden Sortieralgorithmen an. Protokollieren Sie für jedes N jeweils den kleinsten Aufwand, den größten Aufwand und den durchschnittlichen Aufwand des jeweiligen Algorithmus. Plotten Sie mit geeigneter Skalierung diese Werte gegen N.

a, b, c) im code

Wir haben k auf 1,...,3 begrenzt, da die Ausführung sonst zu lange dauert

3)

a) Zeigen Sie, dass die Partionierung der Liste gemäß ii) in $\mathcal{O}(n)$ möglich ist.

b) Konstruieren Sie einen Fall in dem durch ungünstige Wahl des Pivotelements die Laufzeit des Algorithmus $\mathcal{O}(n^2)$ ist.

c) Implementieren Sie den *Quicksort*-Algorithmus als

quicksort(x)

a)

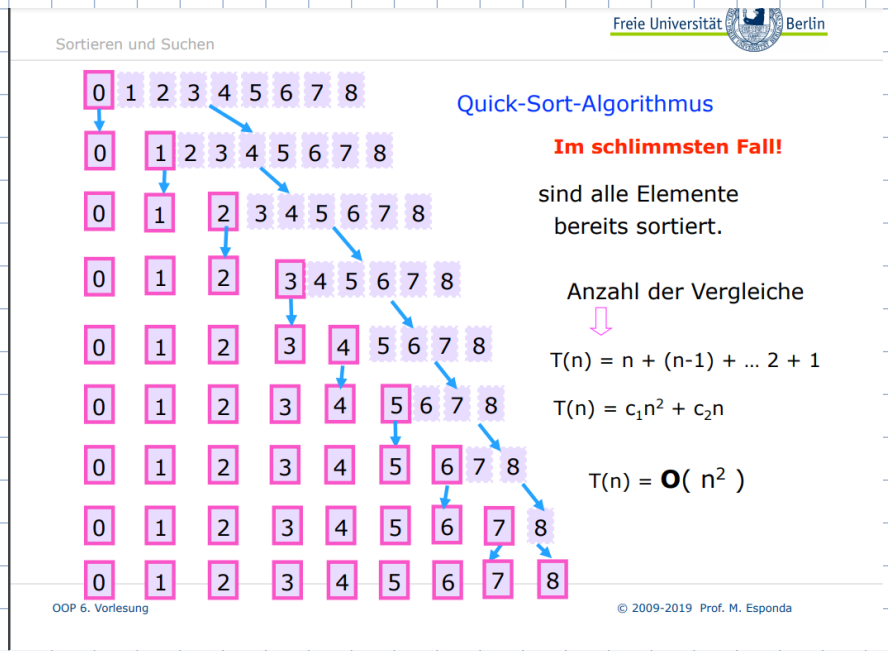
```
def partition(liste):
O(1)    global quick_counter
O(1)    middl = liste[0]
O(1)    left =  []
O(1)    right = []
O(n)    for i in range(1,len(liste)):
O(1)        item = liste[i]
O(1)        quick_counter += 1
O(1)        if middl <= item:
O(1)            right.append(item)
            else:
O(1)            left.append(item)
O(1)    return (left, [middl], right)
```

$\Rightarrow \mathcal{O}(n)$

b)

Wir wählen pivot := erstes Element der Liste

Worst case: Liste ist bereits Sortiert



Quelle: ALP 3 von Prof. M. Esponda

c) im Code