RWTH Aachen **SS 2018**
Lehrgebiet Theoretische Informatik Blatt 7
Rossmanith–Dreier–Kuinke 11.6.2018

Übung zur Vorlesung Algorithmen und Datenstrukturen

Aufgabe T14

Füllen Sie die folgende Tabelle aus.

	Quicksort	Mergesort	Insertion-Sort
in-place?			
stabil?			
Laufzeit (worst-case)			
Laufzeit (Durchschnitt)			
vergleichsbasiert?			

Lösungsvorschlag:

	Quicksort	Mergesort	Insertion-Sort
in-place?	fast	nein	ja
stabil?	nein	ja	ja
Laufzeit (worst-case)	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$
Laufzeit (Durchschnitt)	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$
vergleichsbasiert?	ja	ja	ja

Aufgabe T15

Sehen Sie sich folgendes Programm, welches als Eingabe ein Array a der länge n bekommt, an und finden Sie heraus, was es tut. Was ist seine Laufzeit?

```
for(int k = 0; k < n − 1; k++) {
    int min = k;
    for(int i = k + 1; i < n; i++) {
        if(a[i] < a[min]) {
            min = i;
        }
    }
    if(min ≠ k) {
        int temp = a[k];
        a[k] = a[min];
        a[min] = temp;
    }
}
```

Lösungsvorschlag:
Bei diesem Algorithmus handelt es sich um Selection-Sort. Ähnlich zu Insertion-Sort ist als Invariante nach k Schritten das Teilarray $[0, k]$ bereits sortiert. In jeden Schritt wird im Teilarray $[k+1, n-1]$ das minimale Element gesucht und falls es kleiner als $a[k]$ ist, mit $a[k]$ getauscht. Am Ende ist das gesamte Array sortiert.
Genau wie bei Insertion-Sort beträgt die Worst-Case Laufzeit $\Theta(n^2)$, im Unterschied zu Insertion-Sort hat Selection-Sort allerdings immer eine Laufzeit von $\Theta(n^2)$.

Aufgabe H12 (5 Punkte)
Analysieren Sie die Laufzeit von Quicksort für den Spezialfall, daß *alle* Elemente identisch sind. Wie verhält sich Insertionsort in diesem Fall?

Lösungsvorschlag:
Quicksort braucht $\Theta(n \log n)$. Insertionsort ist in linearer Zeit fertig.

Aufgabe H13 (8 Punkte)
Erfinden Sie ein Sortierverfahren, daß ein Array der Größe n in linearer Zeit sortieren kann unter der Annahme, daß das Array ganze Zahlen zwischen -1000 und 1000 enthält. Ist ihr Verfahren In-place?
Erläutern Sie die Idee und stellen Sie den Algorithmus zusätzlich in Form von Pseudocode oder einer Implementierung in einer vernünftigen Programmiersprache vor.

Lösungsvorschlag:
Wir zählen, wie häufig jede Zahl zwischen -1000 und 1000 vorkommt. Dafür initialisieren wir zuerst ein Array der Länge 2001 mit 0-en. Dann einmal durch das Array und erhöhen in jedem Schritt den Eintrag an der entsprechenden Stelle. Dies geht in $O(n)$. Danach benutzen wir unser Zählarray um ein sortiertes Ausgabearray zu erzeugen. Dieses Verfahren is In-Place, da wir das Eingabearray überschreiben können und wir nur konstant viel extra Speicher brauchen (Array der Größe 2000 ist in $O(1)$). Eine Beispielimplementation in Python:

```
def SmartSort(a):
    count = [0] * 2001
    for i in a:
        count[i + 1000] += 1
    #k is next free cell of a
    k = 0
    for i in range(0, len(count)):
        for j in range(0, count[i]):
            a[k + j] = i - 1000
        #next free cell is after last insertion
        k = k + count[i]
    return a
```

Aufgabe H14 (10 Bonuspunkte)

Das Kapitel über Hashing liegt nun schon über eine Woche zurück. Auf den Folien gab es zu diesem Thema mehrere Lemmata mit zugehörigem Beweis. Eines dieser Lemmata ist aber falsch.

Lesen Sie alle Lemmata noch einmal in Ruhe durch und finden Sie das fehlerhafte. Geben Sie ein Gegenbeispiel an, das die fehlerhafte Aussage illustriert. Geben Sie schließlich die Stelle im Beweis an, welche fehlerhaft ist.

Vorsicht: Der Fehler im Beweis ist natürlich nicht leicht zu entdecken. Er ist subtil.

Hinweise: Die 10 erreichbaren Punkte zählen nicht für die Klausurteilnahmevoraussetzungsquote, aber werden natürlich — wenn die Aufgabe richtig bearbeitet wurde — zu Ihren Punkten dazugezählt. Das fehlerhafte Lemma hat keinerlei Konsequenzen auf andere Folien, Tutor- oder Hausaufgaben.

Lösungsvorschlag:

Das Lemma auf Folie 55 ist falsch. Ein Gegenbeispiel liefert T12: Dort müsste die Wahrscheinlichkeit $1/4$ sein, daß $h_{a,b}(2) = 0$ ist, laut falschem Lemma. Die Wahrscheinlichkeit ist aber wirklich $2/5$.

Im Beweis wird unterschieden, ob es ein x mit $h(x) = k$ gibt oder nicht. Das geht nicht, denn h ist ja eine Zufallsvariable und daher gibt es so ein x mit einer gewissen Wahrscheinlichkeit.