Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

27. Oktober 2016

Datenstrukturen & Algorithmen — Blatt 6 — HS 16

Abgabe: Am Donnerstag, den 3.11.2016, vor Beginn der Vorlesung um 10 Uhr im Eingangsbereich vor ML D28. Bitte heften Sie Ihre Blätter zusammen und benutzen Sie dieses Blatt als Deckblatt. Füllen Sie auch die ersten zwei der untenstehenden Felder aus.

Übungsstunde (Raum & Zeit): _____

Abgegeben von: _____

Korrigiert von: _____

erreichte Punkte: ____

Aufgabe 6.1 *Vergleich von Sortieralgorithmen.*

Gegeben seien ein Array $A[1..n]$ und die folgenden Java-Implementationen der Sortieralgorithmen *Bubblesort*, *Sortieren durch Einfügen*, *Sortieren durch Auswahl* und *Quicksort*. Diese werden mit den Parametern $l = 1$ und $r = n$ aufgerufen, um A in aufsteigender Reihenfolge zu sortieren.

```
public void bubbleSort(int[] A, int l, int r)
{
    for(int i=r; i>l; i--)
        for(int j=l; j<i; j++)
            if(A[j]>A[j+1])
                swap(A, j, j+1);
}

public void selectionSort(int[] A, int l, int r)
{
    for(int i=l; i<r; i++) {
        int minJ = i;
        for(int j=i+1; j<=r; j++)
            if(A[j]<A[minJ])
                minJ = j;
        if(minJ != i)
            swap(A, i, minJ);
    }
}
```

```
public void insertionSort(int[] A, int l, int r)
{
    for(int i=l; i<=r; i++)
        for(int j=i-1; j>=l && A[j]>A[j+1]; j--)
            swap(A, j, j+1);
}

public void quicksort(int[] A, int l, int r)
{
    if(l<r) {
        int i=l+1, j=r;
        do {
            while(i<j && A[i]<=A[l]) i++;
            while(i<=j && A[j]>=A[l]) j--;
            if(i<j) swap(A, i, j);
        } while(i<j);
        swap(A, l, j);
        quicksort(A, l, j-1);
        quicksort(A, j+1, r);
    }
}
```

Der Aufruf von `swap(A, i, j)` vertauscht die Elemente $A[i]$ und $A[j]$. Geben Sie für jeden der obigen Algorithmen asymptotisch an, wie viele Vertauschungen und Vergleiche von Elementen aus A jeweils mindestens und höchstens ausgeführt werden. Untersuchen Sie, wie viel Speicherplatz mindestens und höchstens benutzt wird. Geben Sie auch Folgen aus den Zahlen $1, 2, \ldots, n$ an, bei denen diese Fälle jeweils eintreten. Die Folgen sollen möglichst so angegeben werden, dass n beliebig gewählt werden kann. Beispielsweise kann die absteigend sortierte Folge durch $n, n-1, \ldots, 1$ beschrieben werden.

Aufgabe 6.2 *Verschiedene Fragen zu Sortieralgorithmen.*

Beantworten Sie die folgenden Fragen und begründen Sie kurz Ihre Antwort.

a) Wenn alle Zahlen in einem Max-Heap unterschiedlich sind, an welchen Positionen kann dann das kleinste Element stehen?

b) Ein vergleichsbasiertes Sortierverfahren heisst *stabil*, wenn die relative Reihenfolge identischer Elemente nicht verändert wird. Wenn der Schlüssel 5 beispielsweise zweimal vorkommt, wird die erste 5 niemals hinter die zweite 5 verschoben. Welche der Ihnen bekannten vergleichsbasierten Sortierverfahren sind stabil, bzw. einfach entsprechend anpassbar?

c) Ein Sortierverfahren heisst *in-situ*, falls es zusätzlich zur Eingabefolge nur konstanten Speicherplatz benötigt. Welche der Ihnen bekannten Sortierverfahren sind in-situ, bzw. einfach entsprechend anpassbar?

d) Quicksort hat im schlechtesten Fall eine Laufzeit von $\Theta(n^2)$, Mergesort hingegen braucht immer höchstens $\Theta(n \log n)$ Schritte. Geben Sie zwei stichhaltige Gründe an, warum in der Praxis dennoch meist Quicksort bevorzugt wird.

Aufgabe 6.3 *3-Median-Quicksort.*

Wird Quicksort mit der *3-Median-Strategie* (siehe Buch, Abschnitt 2.2.2 für eine genauere Beschreibung) zur Sortierung von $A[l..r]$ verwendet, dann wird zunächst der Median der Elemente a_l, a_m und a_r für $m = \lfloor (l + r)/2 \rfloor$ berechnet, mit a_r vertauscht und danach a_r als Pivotelement benutzt. Geben Sie für ein beliebiges n_0 eine Folge der Länge $n \geq n_0$ an, für die auch die 3-Median-Strategie $\Omega(n^2)$ Schlüssel vergleicht. Beweisen Sie anschliessend, dass diese Folge wirklich zu quadratisch vielen Schlüsselvergleichen führt.

Aufgabe 6.4 *Algorithmenentwurf und untere Schranke.*

Gegeben seien zwei aufsteigend sortierte Arrays $A = (A[1], \ldots, A[n])$ und $B = (B[1], \ldots, B[n])$. Diese sollen zu einem Array der Länge $2n$ verschmolzen werden, das genau die in A und B gespeicherten Schlüssel enthält.

a) Analysieren Sie die exakte Anzahl der Schlüsselvergleiche, die die Verschmelz-Routine von Mergesort ausführt.

b) Zeichnen Sie den Ablauf der Verschmelz-Routine von Mergesort auf zwei Arrays der Länge 2 als Entscheidungsbaum.

c) Zeigen Sie, dass *jedes* allgemeine deterministische Verfahren zum Verschmelzen zweier Folgen der Längen n im schlimmsten Fall mindestens $2n - \mathcal{O}(\log n)$ viele Schlüssel vergleichen muss. Überlegen Sie dazu, wie viele mögliche Arrays beim Verschmelzen entstehen können. Die in Aufgabe 2.2 bewiesene Abschätzung

$$n \ln n - n \leq \ln(n!) \leq n \ln n - n + \mathcal{O}(\ln n)$$

ist möglicherweise hilfreich.

d) **Knifflige Extraaufgabe:** Zeigen Sie, dass jedes allgemeine deterministische Verfahren zum Verschmelzen zweier Folgen im schlimmsten Fall sogar immer mindestens $2n - 1$ viele Schlüssel vergleichen muss. Betrachten Sie dazu die Arrays $A = (1, 3, 5, \ldots)$ und $B = (2, 4, 6, \ldots)$.

Hinweis: Es reicht nicht aus, zu zeigen, dass die Verschmelz-Routine von Mergesort auf diesen beiden Arrays $2n-1$ viele Vergleiche benötigt. Der Beweis soll für allgemeine Verfahren geführt werden.