Übungsblatt 3

Abgabe bis 10.05.2018
Besprechung: 14.05.2018 – 17.05.2018

Aufgabe 1: Sortieren (2 + 2 + 2 + 1 + 1 Punkte)

Betrachten Sie die Algorithmen `Insertionsort` und `Minimumsuche + Austausch` (Schematisch erklärt in Foliensatz 4, Seite 2).

(a) Geben Sie jeweils Pseudocode für `Insertionsort` und `Minimumsuche + Austausch` an.

(b) Argumentieren Sie, dass beide Sortierverfahren korrekt sortieren.

(c) Geben Sie die Anzahl der Vergleiche und die Anzahl der Vertauschungen auf einer vorsortierten Eingabe der Länge n an.

(d) Konstruieren Sie für allgemeines $n \in \mathbb{N}$ je ein Beispiel, auf denen die Algorithmen eine maximale Anzahl von *Vergleichen* benötigt. Geben Sie diese Anzahl auch an.

(e) Konstruieren Sie für allgemeines $n \in \mathbb{N}$ je ein Beispiel, auf denen die Algorithmen eine maximale Anzahl von *Vertauschungen* benötigt. Geben Sie diese Anzahl auch an.

Aufgabe 2: Heapsort (3 + 1 + 1 + 3 Punkte)

Gegeben sei das Array $A = \langle 4, 2, 12, 10, 18, 14, 6, 16, 8 \rangle$.

(a) Bilden Sie schrittweise (Element für Element) den Min-Heap S für das Array A. Benutzen Sie dabei die Heap-Eigenschaft: Jeder Baumknoten u ist mit einem Element $S[u]$ beschriftet und es gilt: Ist u Elternknoten von v, so ist $S[u] \leq S[v]$. Veranschaulichen und kommentieren Sie alle Schritte.

(b) Wie sieht der Heap aus, wenn Sie eine ExtractMin Operation ausgeführt und dann die Heapeigenschaft wieder hergestellt haben?

(c) Fügen Sie das neue Element 3 zu dem Heap (aus b) hinzu.

(d) Analysieren Sie in $\mathcal{O}$-Notation die Laufzeit der Methode ExtractMax, die das maximale Element aus einem Min-Heap S der Größe n löscht.

Aufgabe 3: Eine Erweiterung von Heapsort (2 + 1 + 4 Punkte)

In der Vorlesung haben Sie das Sortierverfahren Heapsort kennengelernt. Wir betrachten nun eine Erweiterung dieses Verfahrens, das sogenannte *k-Heapsort*. Dabei ist $k \geq 2$ eine natürliche Zahl. Bei diesem Verfahren benutzt man statt einem Binärbaum einen k-nären Baum, bei dem jeder Knoten höchstens k Kinder hat. Deshalb heißt der korrespondierende Heap k-Heap.

(a) Wie kann man einen k-Heap als ein Array repräsentieren? Wie effizient ist es, die Kinder bzw. den Elternknoten eines gegebenen Knoten zu finden?

(b) Geben Sie die Höhe eines k-Heaps an, wenn dieser n Elemente enthält.

(c) Geben Sie Pseudocode für effiziente Implementierungen der Methoden `Insert` und `ExtractMin` an. Analysieren Sie die Komplexität der beiden Methoden in Abhängigkeit von n und k.

Aufgabe 4: Zwei-Drittel-Sortieren (3 + 2 + 2 Punkte)

Eine alternative Methode, um ein Array A der Länge n zu sortieren, ist die Folgende:

Algorithm 1: `ZweiDrittelSortieren`(A, $left$, $right$)

```
if A[left] > A[right] then
    exchange A[left] and A[right] ;
end
if left + 1 ≥ right then
    return;
end
k ⟵ ⌊(right − left + 1) / 3⌋ ;
ZweiDrittelSortieren(A, left, right − k);
ZweiDrittelSortieren(A, left + k, right);
ZweiDrittelSortieren(A, left, right − k);
```

(a) Argumentieren Sie, dass `ZweiDrittelSortieren`(A, 1, n) das Array $A[1..n]$ korrekt sortiert.

(b) Analysieren Sie die Laufzeit von `ZweiDrittelSortieren` im worst-case. Geben Sie Ihre Angaben in $\mathcal{O}$-Notation an.

(c) Ist Zwei-Drittel-Sortieren im worst-case effizienter als Insertsort, Minimumsuche+Austauschen, Quicksort oder Heapsort? Alle Antworten sollten jeweils ausreichend begründet werden.