max planck institut informatik

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Benjamin Doerr, Dr. Reto Spöhel **Wintersemester 2011/12**
Übungsleitung: Alexander Kobel

Übung zu Grundzüge von Algorithmen und Datenstrukturen

http://www.mpi-inf.mpg.de/departments/d1/teaching/ws11/grads/

Übungsblatt 4 Abgabe: Donnerstag, 17. November 2011, 12:00 Uhr

Hinweise zur Abgabe der Aufgaben: Schreiben Sie klar und deutlich in gebührendem Abstand zu weiterem Text Ihren **Namen** und **Matrikelnummer**, ihre **Übungsgruppe als arabische Ziffer** sowie den **Namen Ihres Tutors** auf Ihre Abgabe. **Heften** Sie mehrere Blätter geeignet zusammen. Legen Sie ihre Lösung bis zum Abgabetermin in den **unteren rechten Briefkasten** im Erdgeschoss von Gebäude E1 3 (neben Hörsaal 001). Achtung: Verwechseln Sie den Briefkasten nicht mit dem für die *Stammvorlesung* Algorithms and Data Structures!

Aufgabe 1 (*Schriftliche Übung, 5 Punkte*)

Sei $A[1..n]$ ein Heap wie in der Vorlesung behandelt, $n \geq 10$. Wir nehmen der Einfachkeit halber an, dass $A[1..n]$ lauter verschiedene Elemente enthält. Der *Rang* eines Elementes ist die Anzahl größerer Elemente plus eins; der Rang des größten Elementes ist also 1, derjenige des zweitgrößten 2, und so weiter.

Wie in der Vorlesung besprochen folgt aus der Heap-Eigenschaft, dass das größte der n Elemente an der Position $A[1]$ zu finden ist.

a) An welchen Positionen kann das zweitgrößte Element sein?

b) An welchen Positionen kann das drittgrößte Element sein?

c) An welchen Positionen kann das kleinste Element sein?

d) Nehmen Sie zusätzlich an, dass $n = 2^k - 1$ für ein $k \in \mathbb{N}$, $k \geq 4$. Was sind die möglichen Ränge für das Element an Position $A[2]$?

e) Nehmen Sie zusätzlich an, dass $n = 2^k - 1$ für ein $k \in \mathbb{N}$, $k \geq 4$. Was sind die möglichen Ränge für das Element an Position $A[n]$?

Begründen Sie Ihre Antworten in jeweils ein bis zwei Sätzen.

Bitte wenden...

Aufgabe 2 (*Schriftliche Übung, 6 Punkte*)

Sei $A[1..n]$ ein Heap. Wir betrachten die in der Vorlesung diskutierte Baumstruktur von $A[1..n]$.

a) Beweisen Sie (z.B. per Induktion über n), dass für beliebige Werte von n der Baum genau $\lceil n/2 \rceil$ Blätter hat. [3P.]

b) Überlegen Sie sich, wie für $i \in [n]$ die Position von $A[i]$ in der Baumstruktur mit der Binärdarstellung von i zusammenhängt. Was beobachten Sie? [1P.]
Beweisen Sie Ihre Beobachtung. Erinnern Sie sich dafür daran, wie wir die Funktionen Left(i) und Right(i) in der Vorlesung definiert haben. [2P.]

Aufgabe 3 (*Schriftliche Übung, 4 Punkte*)

Es bezeichne $T(n)$ die Anzahl Elementvergleiche, die ein Aufruf von Heapify (Algorithmus 10 im Skript) für die Wurzel eines (Teil-)Baums der Größe n maximal benötigt. (Mit „Elementvergleichen“ sind Vergleiche der Form „$A[i] \leq A[j]$?“ gemeint, im Unterschied zu Vergleichen von Indices in der Form „$i \leq j$?“.)

Wir haben in der Vorlesung etwas informell diskutiert, dass $T(n) \in O(\log n)$ gilt. In dieser Aufgabe leiten wir eine genauere obere Schranke für $T(n)$ her.

a) Wie in Aufgabe 3 auf Übungsblatt 3 betrachten wir erst nur „schöne“ Werte von n. Überlegen Sie sich, was „schön“ hier heißen muss, und stellen Sie eine rekursive Ungleichung auf für $T(n)$, „n schön“. [2P.]

b) Verwenden Sie a), um zu zeigen, dass $T(n) \leq 2(\log_2(n+1) - 1)$ für „schöne“ Werte von n. [1P.]

c) Ähnlich zu Aufgabe 3 auf Übungsblatt 3 kann gezeigt werden, dass $T(n)$ monoton ist. Setzen Sie dies voraus, und folgern Sie aus b), dass $T(n) \leq 2\log_2 n$ für beliebige Werte von n. [1P.]

Aufgabe 4 (*Schriftliche Übung, 1 Punkt*)

Geben Sie ein explizites Beispiel einer Instanz der Größe n, für die Quick-Sort $\Omega(n^2)$ Vergleiche benötigt, falls immer das erste Element des zu sortierenden (Teil-)Arrays als Pivot genommen wird.

Bonusaufgabe (*4 Bonuspunkte*)

In der Vorlesung haben wir *binäre* Heaps behandelt. Ein ternärer Heap ist analog definiert, mit dem einzigen Unterschied, dass ein innerer Knoten i im Normalfall drei Kinder Left(i), Middle(i), Right(i) hat. Es ist nicht schwer einzusehen, das Heap-Sort auch mit ternären Heaps implementiert werden kann.

a) Sei nun $T'(n)$ wie $T(n)$ in Aufgabe 3 aber für *ternäres* Heap-Sort definiert. Was ist $\lim_{n\to\infty} \frac{T'(n)}{T(n)}$? In welcher der beiden Varianten von Heap-Sort benötigt ein Aufruf von Heapify also im worst-case asymptotisch weniger Vergleiche? [3P.]

b) Was geschieht für k-äres Heap-Sort, $k \geq 4$? [1P.]