Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dr. Hanjo Täubig
Tobias Lieber

Sommersemester 2011
Übungsblatt 4
9. September 2011

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: 9. September 2011

Hausaufgabe 1

Implementieren Sie den QuickSelect-Algorithmus.
Implementieren Sie in der Klasse `UIsqsArray` in der Funktion `quickSelect` den QuickSelect-Algorithmus der das i-te Elemente in dem unsortierten Feld `A` liefert.
Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die Klasse `UIsqsArray`.

Hausaufgabe 2

Implementieren Sie in der Klasse `UIaHeap` einen adressierbaren binären Heap, der neben den Basisoperationen auch die Operationen `inserth`, `remove` und `decreaseKey` implementiert.
Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die Klasse `UIaHeap`.
Hinweis: Hier macht die Benutzung eines Dynamischen Arrays Sinn. Falls Sie die früher benutzte Klasse `UIsArray` *benutzen wollen, so fügen Sie den Quellcode Ihrer Abgabe hinzu. Sie können aber auch das in Java implemntierte Dynamische Array benutzen, welches deutlich flexibler ist.*

Aufgabe 1

Seien A und B nicht leere Mengen und $\leq_A$ und $\leq_B$ totale Ordnungen auf A bzw. B.
Definieren Sie eine totale Ordnung auf $A \times B$.
Wozu brauchen wir den Begriff der totalen Ordnung im Zusammenhang der aktuellen Vorlesungsthemen Sortieren und Selektieren?
Hinweis: Eine lineare Ordnung $\leq$ auf einer Menge X ist

- transitiv: $\forall x, y, z \in X: \quad x \leq y \ \wedge \ y \leq z \ \Rightarrow \ x \leq z$
- antisymmetrisch: $\forall x, y \in X: \quad x \leq y \ \wedge \ y \leq x \ \Rightarrow \ x = y$
- linear (total): $\forall x, y \in X: \quad x \leq y \ \vee \ y \leq x$
- reflexiv: $\forall x \in X: \quad x \leq x$ (folgt aus der Linearität)

Aufgabe 2

Veranschaulichen Sie das Vorgehen des RadixSort-Algorithmus anhand der folgenden Wörter:

`alt, hort, kalt, bar, sport, spalt, ort, stall, spart, speer, bart`

Benutzen Sie die lexikographische Sortierung (Dabei sei das Leerzeichen ␣ kleiner als jeder andere Buchstabe). Es reicht aus, wenn sie als Alphabet die Buchstaben ␣, a, b, e, h, k, l, o, p, r, s, t verwenden und daher die Schlüssel 0,...,11 verwenden.

Aufgabe 3

Wir wollen in dieser Aufgabe den `QuickSort` Algorithmus näher analysieren.

a) Geben Sie eine Familie von Eingabearrays an, so dass der `QuickSort` Algorithmus für jedes n Laufzeit $\Omega(n^2)$ benötigt.

b) Zeigen Sie: Wird in dem Algorithmus als Pivot-Element das k-größte Element ausgewählt (wobei $k \in \mathbb{N}$ eine beliebige aber feste Konstante ist), so liegt die Worst-Case Laufzeit des modifizierten Algorithmus immer noch bei $\Omega(n^2)$.

   Ist $|A| < k$, so wird das Pivot-Element wie im Algorithmus der Vorlesung gewählt.

c) Beweisen Sie: Wählen wir als Pivot-Element das $\lfloor \frac{9}{10} n \rfloor$-größte Element aus `A` ($|A| = n$), so ist die Worst-Case Laufzeit $\mathcal{O}(n \log n)$ (wenn die Selektion des Pivot Elements in $\mathcal{O}(n)$ realisiert ist).

Aufgabe 4

Beweisen Sie folgende in der Vorlesung benötigte Aussage: Ein fast vollständiger Binärbaum der Höhe h hat $2^{h-1} \leq n \leq 2^h - 1$ Knoten (Die Wurzel hat Höhe 1).
Definition; Ein fast vollständiger Binärbaum der Höhe h ist ein vollständiger Binärbaum der Höhe $h-1$ dessen Knoten so angeordnet werden können, dass es auf dem Level h einen Knoten e mit dem Vaterknoten v gibt so dass gilt:

- Alle Knoten auf dem Level $h-1$ die „links“ von v stehen zwei Kinder haben
- und alle Knoten auf dem Level $h-1$ die „rechts“ von v stehen keine Kinder haben
- oder e ist die Wurzel.

Aufgabe 5

Führen Sie auf einem anfangs leeren Binären Heap folgende Operationen aus und stellen Sie die Zwischenergebnisse graphisch dar:

- build({15, 20, 9, 1, 11, 8, 4, 13, 17}),
- insert(7),
- delMin(),
- decreaseKey(20, 3),
- delete(8).

Aufgabe 6

Analysieren Sie die Anzahl der Element-Vergleiche, die der in der Vorlesung angegebene `siftDown`-Algorithmus im Worst-Case benötigt.
Geben Sie ausgehend von dem Algorithmus in der Vorlesung einen Algorithmus an, der mit $\log n + \mathcal{O}(\log \log n)$ Vergleichen auskommt.