TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Übungsblatt 3

PDF erstellt am: 4. April 2022

Deadline für dieses Übungsblatt ist **Montag, 02.05.2022, 20:00 Uhr**. Um Aufgaben für diese Übung anerkannt zu bekommen, gehen Sie folgendermaßen vor:

1. Öffnen Sie den TUWEL-Kurs der Lehrveranstaltung *186.866 Algorithmen und Datenstrukturen (VU 5.5)* und navigieren Sie zum Abschnitt *Übungsblätter*.
2. Teilen Sie uns mit, welche Aufgaben Sie gelöst haben **und** welche gelösten Aufgaben Sie gegebenenfalls in der Übungseinheit präsentieren können. Gehen Sie dabei folgendermaßen vor:
   - Laden Sie Ihre Lösungen in einem einzigen PDF-Dokument in TUWEL hoch.
     Link *Hochladen Lösungen Übungsblatt 3*
     Button *Abgabe hinzufügen*
     PDF-Datei mit Lösungen hochladen und *Änderungen sichern*.
   - Kreuzen Sie an, welche Aufgaben Sie gegebenenfalls in der Übung präsentieren können. Die Lösungen der angekreuzten Aufgaben müssen im hochgeladenen PDF enthalten sein.
     Link *Ankreuzen Übungsblatt 3*
     Button *Abgabe bearbeiten*
     Bearbeitete Aufgaben anhaken und *Änderungen speichern*.

Bitte beachten Sie:

- Bis zur Deadline können Sie sowohl Ihr hochgeladenes PDF, als auch Ihre angekreuzten Aufgaben beliebig oft verändern. Nach der Deadline ist keine Veränderung mehr möglich. Es werden ausnahmslos keine Nachabgabeversuche (z.B. per E-Mail) akzeptiert.
- Sie können Ihre Lösungen entweder direkt in einem Textverarbeitungsprogramm erstellen, oder aber auch gut leserliche Scans bzw. Fotos von handschriftlichen Ausarbeitungen hochladen (beachten Sie die maximale Dateigröße).
- Beachten Sie die Richtlinien für das An- und Aberkennen von Aufgaben (Details finden Sie in den Folien der Vorbesprechung).

Aufgabe 1. Führen Sie eine Internetrecherche zu Heapsort durch und machen Sie sich mit dem Algorithmus vertraut.

(a) Ist ein Max-Heap besser für aufsteigendes oder absteigendes Sortieren geeignet?

(b) Gegeben ist der folgende **Max-Heap** als Array, kodiert wie in der Vorlesung beschrieben:

0	8	5	6	2	1	3

Stellen Sie den **Max-Heap** als Baum dar. Beachten Sie die Reihenfolge der Kinder.

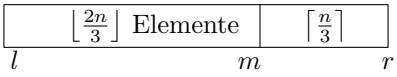
(c) Sortieren Sie den in Unteraufgabe (b) gegebenen **Max-Heap** mithilfe von Heapsort. Stellen Sie nach jedem Schritt den resultierenden (teilsortierten) Heap als Baum dar. Erklären Sie dabei, wie Sie **Heapify-down** korrekt anwenden.

Aufgabe 2.

(a) Führen Sie **Mergesort** auf dem folgenden Array aus und geben Sie Zwischenschritte in Form eines Ablaufdiagrammes wie im Foliensatz „Divide-and-Conquer“ an.

54	21	93	39	89	71	20

(b) In der Vorlesung wurde die klassische **Mergesort** Variante vorgestellt. Das Array wird in zwei ungefähr gleich große Hälften geteilt. Nehmen Sie nun an, dass die Aufteilung ein anderes Verhältnis besitzt, nämlich $\frac{2}{3}$ zu $\frac{1}{3}$.



Beweisen Sie die Laufzeit von $O(n \log n)$ für diese Aufteilung. Folgen Sie dabei den Ideen aus der Vorlesung. Zeigen Sie, dass für die Anzahl der Vergleiche $C(n) \leq n \log_{\frac{3}{2}} n$ gilt.

Hinweis: Beachten Sie, dass für $n > 1$ gilt: $1 \leq \lceil \frac{n}{3} \rceil \leq \lfloor \frac{2n}{3} \rfloor \leq \frac{2n}{3}$. Dadurch vereinfacht sich die Rechnung aus der Vorlesung. Es ist sinnvoll mit $\log_{\frac{3}{2}}$ zu rechnen, da z.B. $\log_{\frac{3}{2}} \frac{2n}{3} = \log_{\frac{3}{2}} n - 1$ gilt.

Aufgabe 3. Gegeben ist das Array $[1, 6, 3, 8, 4, 2, 7]$. Führen Sie **Quicksort** auf dem oben gegebenen Array aus. Wählen Sie stets das erste Element als Pivot-Element aus und implementieren Sie den Schritt des Aufteilens so, dass die Elemente einer Subfolge immer in derselben Reihenfolge angeordnet werden wie in der Originalfolge. Geben Sie die Zwischenschritte wie im Foliensatz „Divide-and-Conquer“ an.

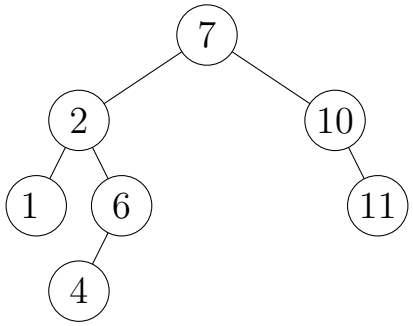
Aufgabe 4. Erfinden Sie ein Sortierverfahren, welches ein Array der Größe n in linearer Zeit sortieren kann unter der Annahme, dass das Array ausschließlich ganze Zahlen zwischen -1000 und 1000 enthält. Geben Sie Ihren Algorithmus in detailliertem Pseudocode an und achten Sie dabei auf die Array-Indices. Ist der zusätzliche Speicher, den Ihr Algorithmus benötigt, durch $O(1)$ beschränkt? Falls ja, nennt man ein solches Sortierverfahren *in-place*.

Aufgabe 5. Beantworten Sie die folgenden Fragen zu binären Suchbäumen:

(a) Wie sieht ein anfangs leerer binärer Suchbaum aus, nachdem die Schlüssel 4, 8, 6, 3, 9 und 5 in dieser Reihenfolge eingefügt wurden? Wie sieht der Baum aus, wenn wir dann erst die 4 löschen und sie anschließend wieder einfügen? Für die zweite Frage gibt es zwei gültige Möglichkeiten. Geben Sie beide an.

(b) Wir betrachten nun binäre Suchbäume, die die Elemente 1 bis 20 enthalten. Beschreiben Sie wie die binären Suchbäume mit (maximaler) Tiefe 19 aussehen. Was ist die Wahrscheinlichkeit, dass bei einer zufälligen Permutation der Eingabeelemente ein Suchbaum maximaler Tiefe entsteht?

Aufgabe 6. Gegeben ist folgender **AVL-Baum**:



(a) Fügen Sie zuerst den Schlüssel 3 und dann 12 in den oben gegebenen AVL-Baum ein. Falls notwendig rebalancieren Sie den Baum nach jedem Einfügen mit einer geeigneten Operation (siehe Foliensatz „Suchbäume“), um wieder einen gültigen AVL-Baum zu erhalten. Markieren Sie dabei den unbalancierten Knoten mit maximaler Tiefe.

(b) Löschen Sie aus dem ursprünglichen AVL-Baum die Schlüssel 1 und 10 in dieser Reihenfolge. Falls notwendig, so rebalancieren Sie den Baum nach jedem Löschvorgang in geeigneter Weise (siehe Foliensatz „Suchbäume“), um wieder einen gültigen AVL-Baum zu erhalten. Markieren Sie dabei den unbalancierten Knoten mit maximaler Tiefe.
