Übungsblatt 2

Algorithmen und Datenstrukturen (WS 2013, Ulrike von Luxburg)

Präsenzaufgabe 1 (Master Theorem) Die folgenden Terme beschreiben die rekursiven Laufzeiten $T(n) = \dots$ diverser Divide & Conquer Algorithmen. Wenden Sie das Master Theorem an, um diese Rekurrenzen asymptotisch aufzulösen, wobei e die Eulersche Zahl bezeichnet.

(a) $4 \cdot T(n/4) + n$ (b) $T(n/42) + \sqrt[3]{n}$ (c) $10^9 \cdot T(n/1000) + n^e$

Präsenzaufgabe 2 (Laufzeitanalyse) Die folgende Funktion FUN erhält als Eingabe ein Array $X = [x_1, x_2, \dots, x_n]$ der Länge n. FUN ruft sich rekursiv mit Teilarrays auf, wobei z.B. $X[5 \dots 42]$ das Teilarray der Indizes $5 \dots 42$ bezeichnet.

```
1:  function FUN(X)
2:      ℓ ← length(X)
3:      if ℓ ≤ 1 then
4:          PRINT("Hallo")
5:          return
6:      end if
7:      p ← ⌈ℓ/3⌉
8:      FUN(X[1 ... p])
9:      FUN(X[2p ... ℓ])
10:     x ← 0
11:     while x < ℓ do
12:         x ← x + √ℓ
13:     end while
14: end function
```

(a) Bestimmen Sie eine Rekurrenzgleichung für die Laufzeit von FUN in Abhängigkeit von n.

(b) Nutzen Sie das Master-Theorem, um die resultierende asymptotische Laufzeit zu ermitteln.

(c) Wie oft wird "Hallo" ausgegeben?

Präsenzaufgabe 3 (Heaps) Betrachten Sie die folgenden drei in Array-Schreibweise gegebenen Binärbäume (diese Reihenfolge der Indizierung heißt auch "Level-Order"):

$T_1 = \boxed{6}\boxed{3}\boxed{9}\boxed{2}\boxed{1}\boxed{7}$ $T_2 = \boxed{9}\boxed{7}\boxed{5}\boxed{8}\boxed{3}\boxed{6}\boxed{2}$ $T_3 = \boxed{9}\boxed{6}\boxed{8}\boxed{3}\boxed{5}\boxed{1}\boxed{4}\boxed{2}$

Nur einer davon (T_a) erfüllt die Max-Heap-Eigenschaft. Ein zweiter (T_b) verletzt die Max-Heap-Eigenschaft an genau einer Stelle. Der dritte (T_c) kann nur durch mehrere Operationen in einen gültigen Max-Heap umgewandelt werden.

(a) Bestimmen Sie $a, b, c \in \{1, 2, 3\}$.

(b) Führen Sie auf T_b die HEAPIFY-Operation an der Fehlstelle aus. Geben Sie das Resultat in Level-Order an.

(c) Führen Sie auf T_a nacheinander die folgenden Operationen aus: INSERT(7), EXTRACTMAX, DECREASE($8 \mapsto 0$). Geben Sie nach jeder Operation das Ergebnis in Level-Order an.

(d) Erstellen Sie aus T_c einen Max-Heap mittels der Operation BUILDMAXHEAP. Geben Sie das Resultat in Level-Order an.

(e) In BUILDMAXHEAP wird die HEAPIFY-Operation auf dem zugrundeliegenden Array von rechts nach links gehend angewandt. Kann man stattdessen auch von links nach rechts gehen? Beweisen oder widerlegen Sie!

(f) Erstellen Sie aus den originalen Array-Daten von T_c einen *ternären* Heap, indem sie ganz analog zum binären Fall die Einträge Level für Level von links nach rechts in einen ternären Baum schreiben. Erfüllt das Ergebnis die Max-Heap-Eigenschaft?

Hausaufgaben zum 6. November, 9:00 (Vorlesungsbeginn).

Handschriftliche Abgaben ab sofort bitte nicht mehr einscannen, sondern zusammengeheftet (!) in der Vorlesung abgeben. Nur elektronisch erstellte Abgaben (LaTeX, Office, etc.) können optional weiterhin als PDF per Email an den Tutor gesendet werden. Weitere wichtige Infos:

- **Wie angekündigt werden ab sofort keine Einzelabgaben mehr akzeptiert.**
- **Oben auf der ersten Seite müssen (1) Übungsgruppennummer und (2) alle Namen/Matrikelnummern angegeben werden.**
- **Bei elektronischer Abgabe bitte Email-Betreffzeile und Datei-Namen nach folgendem Muster: "AD-Gruppe_9_Mustermann_Schuster.pdf"**

Danke ☺

Aufgabe 1 (k-närer Baum (1+1+1+1)) In einem k-nären Baum hat jeder Knoten bis zu k Kinder. Der Wurzelknoten liegt in Level $\ell = 0$. Beantworten Sie mit kurzer Begründung:

(a) Wieviele Knoten liegen maximal in Level $\ell \geq 0$?

(b) Wieviele Knoten hat ein voller Baum der Tiefe ℓ insgesamt?

(c) Wieviele Knoten hat ein vollständiger Baum der Tiefe ℓ insgesamt?

(d) Wieviele Kanten hat ein k-närer Baum der Tiefe ℓ mit insgesamt n Knoten?

Aufgabe 2 (Tree-Walk (2+1+2+2+1)) Wir haben bereits die sogenannte "Level-Order" kennengelernt, bei welchem die Knoten eines Baumes Level für Level von links nach rechts indiziert werden. Betrachten Sie nun die folgenden drei alternativen Reihenfolgen. Jede erhält als Eingabe die Wurzel eines binären Baumes der Größe n:

```
function ORDER1(v)
    if v = null then return
    else
        PRINT(v)
        ORDER1(leftChild)
        ORDER1(rightChild)
    end if
end function
```

```
function ORDER2(v)
    if v = null then return
    else
        ORDER2(leftChild)
        PRINT(v)
        ORDER2(rightChild)
    end if
end function
```

```
function ORDER3(v)
    if v = null then return
    else
        ORDER3(leftChild)
        ORDER3(rightChild)
        PRINT(v)
    end if
end function
```

(a) Bestimmen Sie die Laufzeiten. Die PRINT-Operation benötige Zeit $\Theta(1)$.

(b) Inwieweit kann die jeweilige Laufzeit im best-case verbessert werden?

(c) Betrachten Sie den folgenden mit Buchstaben gelabelten Binärbaum, gegeben in "Level-Order": | N | A | U | O | M | S | R | E | F | R | L | G | A | T | H | I |. Die PRINT-Operation gebe den Buchstaben des an sie übergebenen Knotens aus. In welcher Reihenfolge werden die Buchstaben jeweils durch den Aufruf von ORDER1, ORDER2 und ORDER3 ausgegeben?

(d) Bestimmen Sie einen mit Buchstaben gelabelten vollständigen Binärbaum T, für den ein Aufruf von ORDER2 die Ausgabe "LOVELYTREE" erzeugt. Geben Sie Ihr Ergebnis in der üblichen Array-Schreibweise (Level-Order von T) an.

(e) Betrachten Sie nun auf dem Array von Aufgabenteil (c) den zugehörigen *ter*nären Baum. Starten Sie darauf eine verzwickte Variante von ORDER3 für ternäre Bäume, welche vor dem PRINT-Aufruf die rekursiven ORDER3-Aufrufe auf *rightChild*, *leftChild*, *middleChild* in dieser Reihenfolge (!) vornimmt. Was ist nun die Ausgabe?

Aufgabe 3 (k-närer Heap (2+1+1+2+1+1)) In dieser Aufgabe benötige die HEAPIFY-Operation an der Wurzel eines Array-basierten k-nären Heaps mit n Elementen im worst-case $\lceil k \log_k(n) \rceil$ Schritte. Dabei sind auf jedem seiner $\lceil \log_k(n) \rceil$ vielen Level zusätzlich k Schritte zum Finden (und ggf. Vertauschen) des Maximums der k Kinder des aktuellen Knotens berücksichtigt. Welches ist die beste Wahl für $k \in \mathbb{N}$? Diese Frage wollen wir nun beantworten.

(a) Zunächst suchen wir das Minimum der Funktion $f(x) = x \log_x(n)$ auf dem Definitionsbereich $\mathbb{R}_{>1}$ für festes aber beliebiges n. Bestimmen Sie dazu die Minima von f. (Tipp: Schreiben Sie mittels Basiswechsel $x \log_x(n) = x \cdot \log_e(n)/\log_e(x)$ und leiten Sie mittels Produkt- und Kettenregel ab.)

(b) Nutzen Sie die Einsichten aus (a) um die beste Wahl $k^* \in \mathbb{N}$ zu ermitteln. Wieviele Schritte benötigt diese im worst-case bei Heap-Größe $n = 10^\ell$ für $\ell \in \{1, \ldots, 9\}$? Vergleichen Sie dies mit dem Fall $k = 2$.

(c) Welche Gründe könnten dafür sprechen, wenn in der Praxis trotzdem $k = 2$ verwendet wird?

(d) Wie wirkt es sich auf die Gesamtlaufzeit von HEAPIFY aus, wenn jeder Knoten seine (bis zu k vielen) Kinder in einem separaten binären Max-Heap verwaltet? Bestimmen Sie hierfür insbesondere die notwendige Anzahl Schritte zum Finden (und. ggf. Austauschen) des Maximums der k Kinder des aktuellen Knotens.

(e) Führen Sie auf das Resultat von Aufgabenteil (d) und (f) der *Präsenz*aufgabe 3 jeweils die Operation DECREASE($9 \mapsto 1$) aus. Wieviele Vertauschungen wurden jeweils durchgeführt?

(f) Wir bezeichnen ein Array A mit paarweise unterschiedlichen Einträgen als 2/3-kompatibel, wenn sowohl der jeweils per Level-Order definierte binäre Heap H_2 als auch der ternäre Heap H_3 beide die Max-Heap-Eigenschaft erfüllen. Überdenken Sie folgende Aussage: *"Für alle 2/3-kompatiblen Arrays gilt: In H_3 benötigt keine* DECREASE*-Operation auf der Wurzel mehr Vertauschungen, als dieselbe* DECREASE*-Operation auf der Wurzel in H_2"*. Stimmt das? Beweisen oder widerlegen Sie!

Aufgabe 4 (Sortieren (1+1+2))

(a) Führen Sie MERGE(22579, 1248) Schritt für Schritt wie auf Folie 215 aus.

(b) Führen Sie MERGESORT(67834291) Parallelschritt für Parallelschritt wie auf Folie 217 aus.

(c) Betrachten sie die aus der Vorlesung bekannte MERGE-Funktion (Folie 213). Geben Sie zwei verschiedene Möglichkeiten an, wie durch kleine Änderungen an dieser Funktion insgesamt eine *absteigende* Sortierung durch MERGESORT geliefert wird.

Aufgabe 5 (Stacks und Queues (2+1)) Angenommen Sie möchten "nach außen" eine Queue-Datenstruktur mit den üblichen Funktionen ENQUEUE($\cdot$) und DEQUEUE($\cdot$) bereitstellen, wozu Ihnen "intern" allerdings nur genau 2 Stacks zur Verfügung stehen (also keine Arrays, Listen, oder anderes).

(a) Beschreiben Sie eine Implementierung der Queue (in Worten und in Pseudocode), die nur die beiden Stacks benutzt. Was koennen Sie über die worst-case-Laufzeit für eine ENQUEUE($\cdot$)- bzw. DEQUEUE($\cdot$)-Operation sagen?

(b) Nun wollten wir eine Methode betrachten, mit der man die Laufzeit mehrerer Operationen gemeinsam betrachtet:

- Betrachten Sie eine beliebige Folge von n ENQUEUE($\cdot$)/DEQUEUE($\cdot$)-Operationen.
- Berechnen Sie die worst-case-Laufzeit T_n für diese Folge von Operationen (wobei man beachtet, dass manche Operationen vielleicht sehr schnell, andere sehr wenig Zeit benötigen).
- Dann heißt T_n/n die *amortisierte Laufzeit* dieser Operationen.

Falls nicht bereits geschehen, finden Sie eine Implementierung einer Queue, die wie oben nur zwei Stacks benutzt und amortisierte Laufzeit $\mathcal{O}(1)$ hat. Begründung?