Institut für Informatik
der Universität München
Prof. M. Hofmann
Dr. Andreas Abel
Dipl. Inf. Roland Axelsson

SS 2009

10. Juni 2009

Lösung der Zwischenklausur
Algorithmen und Datenstrukturen

Aufgabe 1:

a) $a = 16, b = 2, \log_b a = 4, f(n) = 40n - 6$
$f(n) = O(n)$, also ist $f(n) = O(n^{4-\epsilon})$ für $\epsilon = 3$. Nach Master-Theorem gilt deshalb $T(n) = \Theta(n^4)$.

b) $a = 27, b = 3, \log_b a = 3, f(n) = 3n^3 \log n$
Zwar gilt, daß $n^3 \log n$ asymptotisch größer ist als n^3, aber nicht polynomiell größer. $\frac{f(n)}{n^{\log_b a}} = \frac{n^3 \log n}{n^3} = \log n$ ist asymptotisch kleiner als n^ϵ für alle $\epsilon > 0$. Das Master-Theorem ist deshalb nicht anwendbar.

c) $a = 4, b = 2, \log_b a = 2, f(n) = 3n^2 + \log n$
$f(n) = \Theta(n^2)$, also ist nach Master-Theorem $T(n) = \Theta(n^2 \log n)$.

d) $a = 4, b = 16, \log_b a = \frac{1}{2}, f(n) = 100 \log n + (2n)^{\frac{1}{2}} + \frac{1}{n^2}$
Da asymptotisch nur die größte Funktion in den drei Summanden ausschlaggebend ist (siehe Aufgabe H-1 a) auf dem 1. Übungsblatt), gilt $f(n) = \Theta(n^{\frac{1}{2}})$, also ist nach Master-Theorem $T(n) = \Theta(\sqrt{n} \log n)$.

Aufgabe 2:

Vorüberlegung:

$$f_2 = \Theta(\sqrt{n}) \ , \qquad f_1 = \Theta(n \log n) \ , \qquad f_3 = \Theta(n \log n)$$
$$f_4 = \Theta(n^2 \log n) \ , \qquad f_6 = \Theta(n^{\frac{31}{15}}) \ , \qquad f_5 = \Theta(n^{\log n})$$

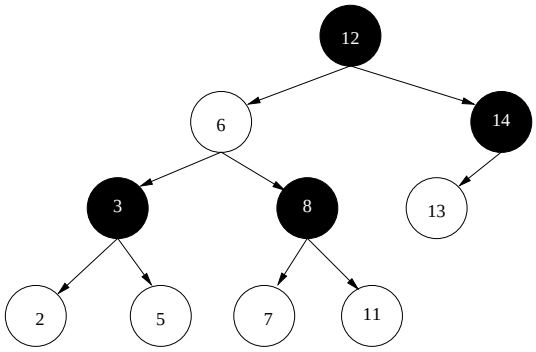
Daraus läßt sich folgende Lösung leicht ableiten:

	ja	nein
$f_5(n) = \Theta(f_3(n))$		X
$f_6(n) = O(f_4(n))$		X
$f_1(n) = O(f_3(n))$	X	
$f_2(n) = \Omega(f_3(n))$		X
$f_4(n) = O(f_1(n))$		X
$f_5(n) = \Omega(f_6(n))$	X	
$f_1(n) = \Omega(f_4(n))$		X
$f_2(n) = O(f_6(n))$	X	

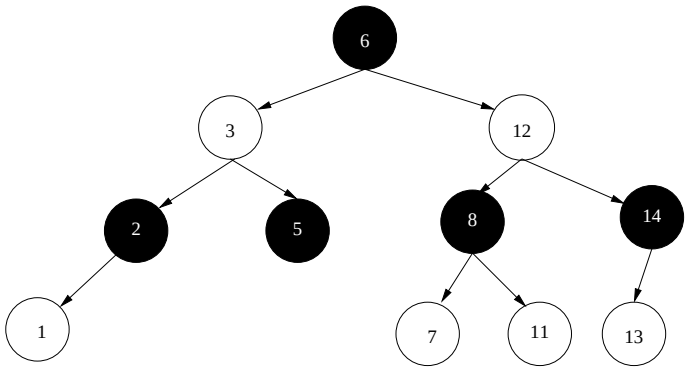
Aufgabe 3:

a) Makroknoten sind schwarze Knoten samt aller direkten roten Nachfolger, d.h. (12)-6, (5)-3, (8)-7-11 und (14)-13.

b) Operationen: neuer roter Knoten "2" als linker Nachfolger von "3", Rechtsrotation um die Position an der sich aktuell die "5" befindet und Umfärben der "3" sowie der "5".

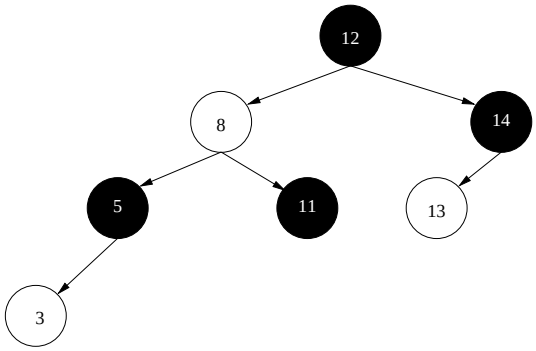


c) Operationen: neuer Knoten "1" als linker Nachfolger von "2", Umfärben der Knoten "3", "2" und "5", Rechtsrotation um die Wurzelposition, Umfärben der "6" sowie der "12".



d) Operationen: nach dem Löschen der “7” ist nichts mehr zu tun

e) Operationen: Vertauschen der wurzelnahen “8” mit der zu löschenden “6”



Aufgabe 4:

a) MERGE-SORT-O sortiert $a[p..r]$: Denn wenn $p \geq r$ ist das Array höchstens einelementig, also schon sortiert. Wenn $p < r$, dann ist $p \leq q < r$. Nach Induktionshypothese sind $a[p..q]$ und $a[q+1..r]$ sortiert. Da $q+1 \leq r$, ist der Zugriff auf $a[q+1]$ gültig. Wenn $a[q] \leq a[q+1]$ ist wegen aufsteigender Sortierung jedes Element aus $a[p..q]$ kleiner gleich jedem Element aus $a[q+1..r]$; Mischen ist also nicht notwendig. Wenn $a[q] > a[q+1]$, wird gemischt, und nach dem Mischen ist das ganze Array $a[p..r]$ sortiert.

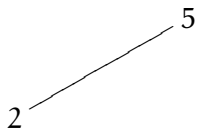
b) Nehme für A_n ein aufsteigend sortiertes Array mit n Elementen. Dann schlägt der Test in Zeile 5 immer fehl, es wird also nie gemischt, MERGE-SORT-O liefert die Eingabe unverändert zurück.

c) $T(n) = 2T(n/2) + c_0$. Nach der Master-Methode mit $c = \log_2 2 = 1$ und $f(n) = \Theta(1)$ und $f(n) = O(n^{1-\epsilon})$ für z.B. $\epsilon = 1/2$ ist $T(n) = \Theta(n)$. [Dies ist sinnvoll, da ja ein korrektes Sortierverfahren, auch wenn es nichts zu sortieren gibt, sicherstellen muss, dass das Ausgabe-Array sortiert ist, und das kostet $\Theta(n)$ Zeit.] Die best-case Laufzeit von Merge-Sort ist $\Theta(n \log n)$, also läuft MERGE-SORT-O im besten Fall schneller als Merge-Sort.

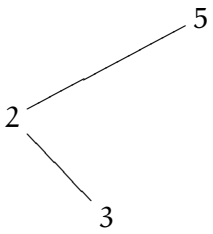
d) Damit ganz am Ende die Optimierung greift, müssen zu Beginn die n kleinsten Elemente auf $a[1..n]$ verteilt sein und die n größten Elemente (also die restlichen), auf $a[n+1..2n]$. Es gibt $(2n)!$ mögliche Verteilungen der Elemente auf $a[1..2n]$, davon erfüllen nur $n!n!$ die geforderte Eigenschaft. [Die n kleinsten können in $a[1..n]$ beliebig verteilt werden, das sind $n!$ Permutationen, analog für die n größten.] Also $P(n) = \frac{n!n!}{(2n)!}$.

$$P(5) = \frac{5}{10} \cdot \frac{4}{9} \cdot \frac{3}{8} \cdot \frac{2}{7} \cdot \frac{1}{6} < \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{3} \cdot \frac{1}{6} = \frac{1}{24 \cdot 6} < \frac{1}{20 \cdot 5} = 1\%.$$

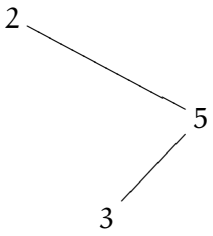
Aufgabe 5: Betrachte den quasibalancierten Baum:



Einfügen der 3 erzeugt den unbalancierten Baum:



Durch eine Rechtsrotation entsteht der weiterhin unbalancierte Baum:



Professor Geröllheimer sollte nochmal in Ruhe über sein Konzept nachdenken.

Aufgabe 6:

a) Der Weinkühlschrank fasst $8/(\log 8) = 8/3 = 2.66...$, geteilte Flaschen haben aber nur noch geringen Wert, also begnügen wir uns mit 2 Flaschen in Kategorie A. Beispielsweise habe der Kellner 8 Flaschen im Gesamtwert von 100, mit den einzelnen Werten $40, 25, 10, 5, 5, 5, 5, 5$. Die beiden teuersten Flaschen haben zusammen einen Wert von 65.

b) Zuerst SELECTieren wir das $n/\log n$ größte Element, in worst-case $\Theta(n)$ Zeit. Danach ist das Flaschen-Array so partitioniert, dass sich ab Position $n/\log n$ die teuersten Flaschen befinden (jedoch unsortiert). Diese sortieren wir nun mit MERGE-SORT, in $\Theta((n/\log n)\log(n/\log n)) = \Theta((n/\log n)(\log n - \log\log n)) = O(n)$ Zeit und nehmen von oben solange Flaschen, bis 60% des Wertes erreicht sind oder eben $n/\log n$ Flaschen gewählt wurden, und lagern sie in den Weinkühlschrank. Dies dauert weniger als $O(n)$ Zeit.

Nun SELECTieren wir das $\lfloor 0.6n \rfloor$ kleinste Element, wieder in $O(n)$ Zeit. Dann ist das Array so partitioniert, dass die 60% billigsten Flaschen unten sind. Diese lagern wir im Vorratsschrank, es bleiben die Kategorie B Flaschen über, die wir im Weinregal lagern.

Jede der Operationen läuft in $O(n)$, also insgesamt $O(n)$ Zeit, was besser ist als $O(n \log n)$.