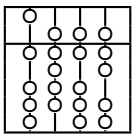
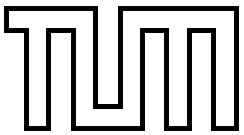


TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK



Lehrstuhl für Effiziente Algorithmen
Prof. Dr. Christian Scheideler, Dr. Stefan Schmid

SS 2008
Midterm-Klausur
31.05.08

Midterm-Klausur zu Grundlagen: Algorithmen und Datenstrukturen

Name	Vorname	Studiengang	Matrikelnummer
Hörsaal	Reihe	Sitzplatz	Unterschrift

Allgemeine Hinweise:

- **Bitte füllen Sie die oben angegeben Felder vollständig aus und unterschreiben Sie!**
- **Schreiben Sie nicht mit Bleistift oder in roter/grüner Farbe!**
- **Lösen Sie die Aufgaben auf dem freien Platz unter der Angabe (bzw. auf der Rückseite). Fordern Sie weiteres Papier von der Klausuraufsicht, falls Ihnen der Platz nicht ausreicht.**
- **Als Hilfsmittel ist ausschließlich ein handbeschriebenes DinA4-Blatt zugelassen. Bei Missachtung wird die gesamte Klausur mit null Punkten bewertet.**
- **Die Arbeitszeit beträgt 120 Minuten.**
- **Prüfen Sie, ob Sie alle 7 Seiten erhalten haben.**

Aufgabe 1 [5 Punkte] **Multiple Choice Aufgabe**

Kreuzen Sie pro Teilaufgabe höchstens ein Kästchen an. Für ein falsches Kreuz gibt es einen halben Minuspunkt, für ein richtiges einen halben Pluspunkt. Wenn Sie kein Kreuz setzen, bekommen Sie auch keine Punkte. Eine negative Gesamtpunktzahl dieser Aufgabe wird zu 0 aufgerundet. Maximieren Sie Ihre Punktzahl!

a) $O(n^2) \subseteq \Theta(n^2)$ ☐ Richtig ☒ Falsch

b) $\Theta(n^2) \cap \Theta(n) = \Theta(n)$ ☐ Richtig ☒ Falsch

c) $(\log n)^{10} \in O(\sqrt{n})$ ☒ Richtig ☐ Falsch

d) $2^{2n} \in O(2^n)$ ☐ Richtig ☒ Falsch

e) $f(n) \in O(g(n)) \Rightarrow f'(n) \in O(g'(n))$ ☐ Richtig ☒ Falsch

f) $f'(n) = \omega(g'(n)) \Rightarrow f(n) = \omega(g(n))$ ☒ Richtig ☐ Falsch

g) $O(f(n)) = \{g(n) | \forall c > 0\ \exists n_0 > 0\ \forall n \geq n_0 : g(n) \leq c \cdot f(n)\}$ ☐ Richtig ☒ Falsch

h) Betrachte die folgende Rekursionsgleichung für $n = 3^k$ für ein $k \in \mathbb{N}$: $T(1) = 3$ und $T(n) = 3n + 2 \cdot T(n/3)$ für alle $n > 1$. Ist $T(n) \in O(n)$? ☒ Richtig ☐ Falsch

i) Ein unbeschränkter Stack kann in der Listendarstellung mit $O(1)$ Zeitaufwand pro *pushBack* und *popBack* Operationen realisiert werden. ☒ Richtig ☐ Falsch

j) Kuckuckshashing benötigt im worst case $O(1)$ Zeitaufwand pro *insert*, *remove* und *find* Operation. ☐ Richtig ☒ Falsch

Aufgabe 2 [8 Punkte] **Oh-Notation**

a) [2 Punkte] Geben Sie den Wachstumsvergleich ($o(\cdot)$, $\omega(\cdot)$ und $\Theta(\cdot)$) für die Funktionen $f(n) = n$ und $g(n) = (\log n)^2$ an und begründen Sie Ihre Aussage!

b) [2 Punkte] Seien f und g positive und monoton wachsende Funktionen, und sei

$$O_v(f(n)) = \{g(n) | \exists c_1, c_2 > 0 \; \forall n \in \mathbb{N} : g(n) \leq c_1 \cdot f(n) + c_2\}$$

Zeigen Sie, dass $O(f(n)) = O_v(f(n))$, das heisst: $O(f(n)) \subseteq O_v(f(n))$ und $O_v(f(n)) \subseteq O(f(n))$.

c) [2 Punkte] Zeigen Sie: Für zwei positive Funktionen $f : \mathbb{N} \to \mathbb{N}$ und $g : \mathbb{N} \to \mathbb{N}$ gilt:

$$f(n) \in o(g(n)) \Leftrightarrow g(n) \in \omega(f(n))$$

d) [2 Punkte] Betrachten Sie die Rekursionsgleichung $T(n)$ für $n = 2^k$ für ein $k \in \mathbb{N}$ mit

$$T(1) = 1$$

$$T(n) = 2T(n/2) + n^2.$$

Leiten Sie (ohne Mastertheorem!) eine geschlossene Form $f(n)$ her, so dass $T(n) = \Theta(f(n))$.

Lösung 2

a) $f(n) \in \omega(g(n))$ und $g(n) \in O(f(n))$

b) $O(f(n)) \subseteq O_v(f(n))$: $c_2 = f(n_0)$ und $c_1 = c$; $O_v(f(n)) \subseteq O(f(n))$: $c = c_1 + c_2$ und $n_0 = 1$

c) $f(n) \in o(g(n)) \Leftrightarrow \forall c > 0 \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq cg(n) \Leftrightarrow \forall 1/c > 0 \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : 1/c f(n) \leq g(n) \Leftrightarrow g(n) \in \omega(f(n))$

d)

$$\begin{aligned}
T(n) &= 2T(n/2) + n^2 \\
&= 2 \cdot (2 \cdot T(n/4) + n^2/4) + n^2 \\
&= \ldots \\
&= 2^k \cdot T(n/2^k) + \sum_{i=0}^{k-1} n^2/2^i \\
&= \ldots \\
&= n \cdot T(1) + n^2 \sum_{i=0}^{k-1} 1/2^i \\
&= n + n^2 \cdot (2 - 1/2^{\log n - 1})
\end{aligned}$$

Also $T(n) \in \Theta(n^2)$.

Aufgabe 3 [4 Punkte] **Rekursion**

In dieser Aufgabe werden Sie Rekursionsgleichungen *ohne Verwendung des Mastertheorems* lösen.

a) [2 Punkte] Gegeben sei der folgende rekursive Algorithmus für die Berechnung von $b = a^n$ für ein $a \in \mathbb{N}$ und $n \in \mathbb{N}$:

```
1: Potenz(a,n):
2: if n = 0 then b := 1
3: else
4:     b:=Potenz(a,⌊n/2⌋)
5:     if (n ungerade) then b := a · b^2  else b := b^2
6: return b
```

Stellen Sie die Rekursionsgleichung $T(n)$ zu $Potenz(a, n)$ auf und leiten Sie eine geschlossene Form her! Sie können annehmen, dass alle Operationen außer dem rekursiven Aufruf konstante Zeit benötigen. Für die Analyse dürfen Sie annehmen, dass n eine Zweierpotenz ist.

b) [2 Punkte] Gegeben sei eine Menge von n Objekten mit Volumen $a_1, ..., a_n \in \mathbb{N}$ und ein Rucksack mit Volumen C. Wir möchten die Menge der Objekte mit maximalem Gesamtvolumen A finden, für die $A \leq C$ ist. Dazu verwenden wir den folgenden Algorithmus, der mit *Rucksack(n,A)* mit $A = \sum_{i=1}^{n} a_i$ aufgerufen wird.

```
1: Rucksack(n,A):
2: if n = 0 then
3:     if (A ≤ C) then return A else return 0
4: else return max{Rucksack(n − 1, A), Rucksack(n − 1, A − a_n)}
```

Stellen Sie die Rekursionsgleichung $T(n)$ zu *Rucksack(n,A)* auf und leiten Sie eine geschlossene Form her (alle Elementaroperationen kosten nur konstante Zeit).

Lösung 3

a)

$$\begin{aligned} T(n) &= T(n/2) + c \\ &= T(n/4) + 2c \\ &= T(n/8) + 3c \\ &= ... \\ &= T(n/2^k) + k \cdot c \\ &= T(1) + \log n \cdot c \\ &= (\log n + 1) \cdot c \end{aligned}$$

Also $T(n) \in \Theta(\log n)$.

b)

$$\begin{aligned} T(n) &= 2T(n-1)+c \\ T(n) &= 2(2T(n-2)+c)+c \\ &= ... \\ &= 2^k T(n-k) + c\sum_{i=1}^{k} 2^i \\ &= ... \\ &= 2^n c + c\sum_{i=1}^{n} 2^i \\ &= 2^n c + c(2^{n+1}-1) \end{aligned}$$

Also $T(n) \in \Theta(2^n)$.

Aufgabe 4 [7 Punkte] **While Schleifen**

Für gegebene Werte $a, b \in \mathbb{N}$ wollen wir a *modulo* b bestimmen. Wenn die ganzzahlige Division nicht als Elementaroperation zur Verfügung steht, braucht dieses Problem einen geeigneten Algorithmus. Wir betrachten zwei Varianten.

a) [2 Punkte] Betrachten Sie foldenden Algorithmus:

```
1: while a ≥ b do
2:     a := a − b
3: return a
```

Bestimmen Sie die Laufzeit $T(a, b)$ in Abhängigkeit von a und b. Benutzen Sie dafür die Potentialfunktion $\Phi(a, b) = a/b$ für die Analyse der while-Schleife, und argumentieren Sie möglichst formal!

b) [5 Punkte] Betrachten Sie folgenden Algorithmus:

```
1: if a < b then return a
2: k := 1
3: while (k · b < a) do k := 2k
4: l := 0; r := k
5: while r − l ≥ 2 do
6:     m := (r + l)/2
7:     if (m · b ≤ a) then l := m else r := m
8: return a − l · b
```

Bestimmen Sie die Laufzeit $T(a, b)$ in Abhängigkeit von a und b. Benutzen Sie dafür $\Phi_1(a, b, k) = a/(k \cdot b)$ in der ersten while-Schleife und $\Phi_2(a, b, k) = r - l$ in der zweiten while-Schleife, wobei Sie die Tatsache ausnutzen, dass am Ende der ersten while-Schleife $k \cdot b \in [a, 2a]$. Argumentieren Sie möglichst formal!

Lösung 4

a) Sei Φ_i der Potenzialwert vor der i-ten Iteration. $\Phi_1 = a/b$ und für alle i: $\Phi_i \geq 1$, sonst terminiert's. Weil $\Phi_i = \Phi_{i-1} - 1$ folgt, dass die Laufzeit gerade a/b ist.

b) Zu Beginn der ersten Whileschleife ist $\Phi_{1,1} = a/b$, danach gilt bei der i-ten Iteration $\Phi_{1,i} = \Phi_{1,i-1}/2$. Das geht solange bis $\Phi > 1$. Die Laufzeit ist also $\log(a/b)$. Analog halbiert sich das Potenzial auch in der zweiten Whileschleife immer, und es gilt $\Phi_{2,i} = \Phi_{2,i-1}/2$, und auch hier ist die Anzahl Iterationen in $O(\log(a/b))$.

Aufgabe 5 [5 (Bonus-) Punkte] **Datenstrukturen und amortisierte Analyse**

Achtung: Diese Aufgabe ist schwierig! Versuchen Sie sie erst dann zu lösen, wenn Sie alle anderen Aufgaben bearbeitet haben! Es ist eine Bonusaufgabe, und die Maximalnote kann auch ohne diese Aufgabe erreicht werden.

Wir wollen eine Datenstruktur DS implementieren, die pro *insert*, *remove* und *find* Operation im worst case nur $O(1)$ Zeit braucht und für n Elemente $O(n)$ Speicherplatz benötigt. Dazu wollen wir die folgenden Strukturen verwenden:

DS1: DS1 braucht für n Elemente $O(n)$ Speicherplatz. Jede *remove* und *find* Operation kostet $O(1)$ Zeit im worst case, aber die i-te *insert* Operation kostet Zeit $O(t_i)$, wobei $t_i \in \mathbb{N}$ der größte Wert ist, für den 2^{t_i-1} die Zahl i teilt. Die ersten insert Operationen brauchen also Laufzeit: 1,2,1,3,1,2,1,4,1,2,1,3,1,2,1,5,1,2,...

DS2: DS2 braucht für n Elemente $O(n^2)$ Speicherplatz. Jede *insert*, *remove* und *find* Operation braucht aber nur $O(1)$ Zeit im worst case.

a) [2 Punkte] Bestimmen Sie die Gesamtlaufzeit von DS1 für eine beliebige Folge von n Operationen. Schließen Sie daraus auf den amortisierten Zeitaufwand einer *insert* Operation.

b) [3 Punkte] Bauen Sie aus DS1 und DS2 eine Datenstruktur DS, die unsere Anforderungen oben erfüllt. Skizzieren Sie dazu eine geeignete *insert*, *remove* und *find* Operation.

Lösung 5

a) Im schlimmsten Fall sind alle Operationen *insert()*. Amortisiert haben wir für n Operation deshalb maximal Kosten $\sum_{i=1}^{n} i/2^i \in O(1)$.

b) siehe Vorlesung