

AlgoDat — Ausarbeitung zum Übungsblatt 2

YRUCREM

Freitag, 12. April 2002

VERSION 1.0

Danke an GELI für einen Hinweis auf einen Rechenfehler.
Ebenfalls an ROBBY für den Hinweis, dass die Aufgabe 9 gefehlt hat.

LORDOFTHEBITE hat mich auf einen Tippfehler bei Aufgabe 2 hingewiesen.

Von TOCVOLXA erhielt ich einen Hinweis auf einen groben Denkfehler bei Aufgabe 7.

Der L^AT_EX-Source, sowie die neueste Version dieses Dokuments sind unter yrucrem.dr.ag/studium/algodat_1/uebungsblatt_2.html erhältlich.

Aufgabe 1

Welche Laufzeit in Θ -Notation für n haben folgende Programme?

Das Erste kommt $(n + 1)$ -mal zur Zeile 1 (beim $(n + 1)$ ten mal bemerkt das Programm, dass es die Schleife nicht noch einmal ausführen muss) und n -mal zur Zeile 2. Also kommen wir auf:

$$T(n) = (n + 1)c_1 + nc_2 = \Theta(n)$$

Das Zweite führt einmal Zeile 1, $(k + 1)$ -mal Zeile 2 und k -mal Zeile 3 aus:

$$T(n) = c_1 + (k + 1)c_2 + kc_3 = \Theta(1)$$

Warum $\Theta(1)$? Weil die Θ -Notation für n gefragt ist und das kommt in der Laufzeitformel gar nicht vor. Für die Θ -Notation nimmt man ja sozusagen die höchste Potenz von n in der Formel und die höchste Potenz von n in dieser Formel ist n^0 und das ist 1.

Aufgabe 2

Zuerst möchte ich den Algorithmus wie er im Skriptum dargestellt ist korrigieren:

In Zeile 5 sollte stehen: `bis A[i].key \geq x`

In Zeile 8 sollte stehen: `bis A[j].key \leq x`

In Zeile 9 sollte stehen: `falls i < j dann`

In Zeile 12 sollte stehen: `bis i > j`

In Zeile 5 brauchen wir deswegen ein ' \geq ', weil sonst die **wiederhole**-Schleife aus Zeile 3 nicht abgebrochen wird. Im Skriptum steht unter dem Pseudocode, dass die erste innere Schleife abgebrochen wird, weil ja das Pivotelement ganz rechts gewählt wurde. Wenn aber statt einem ' \geq ' ein '>' steht (sowie beim Errata auf der AlgoDat-homepage vorgeschlagen) und links vom Pivotelement nur kleinere oder gleiche Elemente sind, dann läuft i über den Rand des Feldes hinaus, weil die Abbruchbedingung nie gegeben ist.

Wenn in Zeile 8 nur ein '<' steht, muss das Stop-Element, das ganz am Anfang eingefügt wurde, *kleiner* und nicht nur *kleinergleich* sein, wie das kleinste Element im Feld. Ist das Stop-Element kleinergleich, dann muss in Zeile 8 ein ' \leq ' stehen.

Was macht Quicksort? Es wählt ein "*zerteilendes*" Element, das sogenannte *Pivotelement* und bringt alle Elemente die kleiner sind auf dessen linke Seite und alle Elemente die größer sind auf dessen rechte Seite.

Dafür benutzt der Algorithmus zwei Zeigervariablen. i durchläuft das Feld von links nach rechts und bleibt bei jedem Element stehen, das grösser oder gleich dem Pivotelement ist. j geht von rechts nach links und stoppt bei jedem kleineren Element. Wenn beide Zeiger stehen geblieben sind (und i auch wirklich noch links von j ist) werden diese beiden Elemente ausgetauscht. Wenn i und j aneinander vorbeilaufen, heißt das, dass links von der Position an dem das i steht nur Elemente sind, die kleiner sind als das Pivotelement und rechts davon nur Elemente die größer oder gleich groß sind. Also tauschen wir das Pivotelement mit dem Element an der Stelle i .

Nach jedem Durchgang steht ein Element genau an der Stelle an die es hingehört. Nun ruft sich Quicksort rekursiv auf um alle Elemente links beziehungsweise rechts des endgültig platzierten Elements zu sortieren.

	Feld										
1	ⁱ 9	0	5	8	2	6	3	8	4	² j	<u>6</u>
2	2	0	5	ⁱ 8	2	6	3	8	⁴ j	9	<u>6</u>
3	2	0	5	4	2	ⁱ 6	³ j	8	8	9	<u>6</u>
4	2	0	5	4	2	³ j	ⁱ 6	8	8	9	<u>6</u>
5	2	0	5	4	2	3	6	8	8	9	6
6	2	0	ⁱ 5	4	² j	<u>3</u>					
7	2	0	² j	ⁱ 4	5	<u>3</u>					
8	2	0	2	3	5	4					
9	ⁱ 2	⁰ j	<u>2</u>								
10	⁰ j	ⁱ 2	<u>2</u>								
11	0	2	2								
12				^j	ⁱ 5	<u>4</u>					
13					4	5					
14							^j	ⁱ 8	8	9	<u>6</u>
15								6	8	9	8
16									ⁱ 8 ^j	9	<u>8</u>
17							^j	8	ⁱ 9	<u>8</u>	
18								8	8	9	
19	0	2	2	3	4	5	6	6	8	8	9

Aufgabe 3

Quicksort so abändern, dass absteigend sortiert wird.

In Zeile 5: statt ' $\text{bis } A[i].\text{key} \geq x$ ' gehört ' $\text{bis } A[i].\text{key} \leq x$ '.

In Zeile 8: statt ' $\text{bis } A[j].\text{key} \leq x$ ' gehört ' $\text{bis } A[j].\text{key} > x$ '.

Außerdem muss das Stopp-Element am Anfang des Feldes, größer sein als alle Elemente des Feldes. Sonst bleibt alles gleich. Das müsste funktionieren, probiert es aus.

Aufgabe 4

Zeigen oder widerlegen Sie: $f = O(x^2)$.

$$x > \log x \Rightarrow x^2 > x \log x \Rightarrow x^2 + 3x^2 - 7x > \lfloor x \log x + 3x^2 - 7x \rfloor$$

Wenn wir also zeigen können, dass

$$4x^2 - 7x = O(x^2)$$

dann muss auch

$$x \log x + 3x^2 - 7x = O(x^2)$$

sein. Damit haben wir das Problem auf ein bereits geöstes zurückgeführt.

Die andere Möglichkeit wäre wieder durch die höchste Potenz zu dividieren, $x \rightarrow \infty$ gehen zu lassen dann würde man sehen, dass man eine Konstante finden kann und die Behauptung somit richtig ist.

Aufgabe 5

Zeigen oder widerlegen Sie folgende Aussage:

$$3^{5^{n+1}} = \Theta(3^{5^n})$$

$$0 \leq c_1 3^{5^n} \leq 3^{5^{n+1}} \leq c_2 3^{5^n} \quad | : 3^{5^n}$$

$$0 \leq c_1 \leq \frac{3^{5^{n+1}}}{3^{5^n}} \leq c_2$$

$$0 \leq c_1 \leq 3^{5 \cdot 5^n - 5^n} \leq c_2$$

$$0 \leq c_1 \leq 3^{4 \cdot 5^n} \leq c_2$$

$$\lim_{n \rightarrow \infty} 3^{4 \cdot 5^n} = \infty$$

Es läßt sich kein c_2 finden, das die Bedingungen erfüllt, also muss die Behauptung falsch sein.

Aufgabe 6

Gegeben ist die Funktion f wie folgt:

$$f(n) = \begin{cases} n^2 & \text{falls } n \text{ keine Primzahl} \\ n^3 & \text{falls } n \text{ Primzahl} \end{cases}$$

Welche der folgenden Aussagen sind richtig und welche sind falsch?

$f = O(n^2)$ ist *falsch*, weil sich kein n_0 finden lässt, ab dem es keine Primzahlen mehr gibt. Darum gibt es immer wieder Stellen an denen $f(n) = n^3$ ist.

$f = O(n^3)$ ist *richtig*, weil $f(n)$ sowieso höchstens n^3 sein kann

$f = \Theta(n^3)$ ist *falsch*, weil man kein n_0 finden kann ab dem es nur noch Primzahlen gibt. Darum gibt es immer wieder Stellen an denen $f(n) = n^2$ ist.

$f = \Omega(n^3)$ ist aus dem gleichen Grund *falsch* wie die obige Behauptung.

Aufgabe 7

Der Algorithmus *ist* abhängig vom Inhalt der Eingabefolge, weil keine unnötigen Datenbewegungen vorkommen (ie. es werden keine Elemente auf sich selbst verschoben). Im Best-Case haben wir 0 Datenbewegungen und im Worst-Case $\Theta(n^2)$ Bewegungen.

Die Schlüsselvergleiche sind *nicht* vom Inhalt der Folge abhängig, sondern nur von der Anzahl der Elemente.

Hier die Laufzeitfunktion:

$$nc_1 + n(n-1)c_2 + (n-1)^2c_3 + \sum_{i=1}^{\frac{n^2-n}{2}} (t_i) c_4$$

Das ergibt:

$$nc_1 + (n^2 - n)c_2 + (n^2 - 2n + 1)c_3 + \sum_{i=1}^{\frac{n^2-n}{2}} (t_i) c_4 = \Theta(n^2)$$

Eigentlich dürfte alles bis auf die Summe klar sein und die will ich jetzt kurz erklären. Nach jedem Durchlauf der Äusseren Schleife befindet sich das größte Element, der unsortierten Teilfolge, ganz an deren Ende (also an der richtigen Position). Das kommt davon, dass Bubblesort ein Element so lange

“vor sich her schiebt” bis ein noch größeres Element gefunden wird (und dann schiebt er das vor sich her, ...). Das bedeutet, dass am Ende jedes Durchlaufes der äußeren Schleife (mit dem Schleifenzähler i) die letzten i Elemente bereits sortiert sind. Das wiederum bedeutet, dass in diesem Bereich *sicher* keine Verschiebungen mehr stattfinden. Sobald also die innere Schleife den Index $(n - i)$ überschritten hat, kommen keine Verschiebungen mehr vor. Ich versuche, das kurz zu verdeutlichen.

Ist die äußere Schleife in ihrem ersten Durchlauf ist $i = 1$, das würde bedeuten sobald die innere Schleife den Index $(n - 1)$ passiert hat, kommen keine Verschiebungen mehr hinzu. Aber $(n - 1)$ ist sowieso das Maximum für die innere Schleife.

Am Ende des Ersten Durchgangs ist das größte Element ganz am Ende des Feldes. $i = 2$, das heißt ab $j = n - 2$ gibt es für die innere Schleife eigentlich nichts mehr zu tun, das letzte Element ist ja schon an der richtigen Stelle.

Wir summieren jetzt also auf wie oft die Bedingung in Zeile 3 erfüllt sein *könnte* und lassen aber die Durchgänge der inneren Schleife weg, die sowieso nichts mehr bringen.

Die Bedingung könnte bei jedem Durchlauf der äußeren Schleife $(n - i)$ -mal erfüllt sein. Also beim ersten mal $(n - 1)$ -mal, dann $(n - 2)$ -mal, ..., dann $(n - (n - 2))$ -mal, dann $(n - (n - 1))$ -mal. Es könnte also $\sum_{i=1}^{n-1} i = \frac{n^2-n}{2}$ -mal sein, dass wir eine Vertauschung haben. Ob wir vertauschen müssen, hängt natürlich vom Element ab, das an der Stelle i steht. Wir müssen also an $\frac{n^2-n}{2}$ Stellen schauen ob hier eine Vertauschung nötig ist. Das machen wir mit $\sum_{i=1}^{\frac{n^2-n}{2}} t_i$, wobei $t_i = 1$ wenn an dieser Stelle getauscht werden muss und $t_i = 0$, wenn nicht getauscht werden muss.

Aufgabe 8

Wer das Beispiel 7 vollständig verstanden hat, sollte hiermit keine Probleme haben. Wie oben gesagt, ist es deswegen erlaubt die innere Schleife nur bis $n - i$ laufen zu lassen, weil nach jedem Durchlauf der äußeren Schleife ein Element an die richtige Position gestellt wurde und da Bubble-Sort die Elemente immer vor sich herschiebt, bis es etwas größeres findet sind immer die letzten i Elemente bereits sortiert. Deswegen macht es keinen Sinn mit der inneren Schleife über den Index $n - i$ zu gehen, weil dort nichts mehr zu tun ist.

Wie wirkt sich das auf die Laufzeitfunktion aus?

$$T(n) = nc_1 + \sum_{i=1}^{n-1} (n-i+1)c_2 + \sum_{i=1}^{n-1} (n-i)c_3 + \sum_{i=1}^{\frac{n^2-n}{2}} (t_i) c_4$$

Die Summen erklären sich ähnlich wie oben. Z.B.: $\sum_{i=1}^{n-1} n-i+1$. Wenn das i der äußeren Schleife gleich 1 ist, gelangen wir $(n-1+1)$ -mal zur Zeile 2. Ist das i gleich 2 gelangen wir nochmal $(n-1+1)$ -mal zur Zeile 2, ...

Die Summe für die Zeile 4 bleibt genauso wie bei Aufgabe 7, weil wir da ja schon berücksichtigt haben, dass es maximal so viele Vertauschungen geben kann.

Damit wir die Laufzeitfunktion besser mit der vorherigen vergleichen können lösen wir die Summen noch auf und erhalten:

$$nc_1 + \frac{n^2+n-2}{2}c_2 + \frac{n^2-n}{2}c_3 + \sum_{i=1}^{\frac{n^2-n}{2}} (t_i) c_4 = \Theta(n^2)$$

Aufgabe 9

Der Algorithmus ist stabil, weil gleichgroße Elemente nie miteinander vertauscht, werden. Außerdem werden immer nur direkt benachbarte Elemente vertauscht (es kann zum Beispiel nicht sein, dass ein 2er der ganz hinten steht mit nur einer vertauschung nach vorne kommt und einen anderen 2er der in der Mitte stand "überholt" (was bei Quicksort zum Beispiel schon möglich ist)).

Aufgabe 10

Die Behauptung stimmt nur für bestimmte n (obwohl diese beliebig groß sein können). Die Definition von $O(g(n))$ ist aber, dass man eine Konstante c finden muss die grösser oder gleich ist als der Funktionswert an der Stelle n ABER diese Konstante muss für *alle* $n \geq n_0$ größer sein als der Funktionswert an der Stelle n . Beim Beweis durch Induktion werden immer nur bestimmte Funktionswerte betrachtet und für die lassen sich natürlich immer Konstanten finden die größer sind, aber wie gesagt, die Konstante müsste für alle $n \geq n_0$ größer sein als der Funktionswert.

Man müsste die Induktion mit der Ungleichung von der Definition von $O(g(n))$ durchführen und das n gegen unendlich gehen lassen und dann funktioniert es nicht mehr.