NitscheSahoo - AD Praktikum für 5.11.13

Aufgabe 3 – Laufzeitanalyse

Aufgabe 3.1:

In den Klammern hinter den Bezeichnungen in den Zeilen geben dar wie wir den Zeitaufwand dieser Operation im Code interpretieren. Eine Zuweisung braucht z.B. konstante Zeit. Eine Zuweisung für die Laufariable in einer For-schleife braucht so-viel Zeitaufwand, wie oft die Iteration emacht wird.

Algorithmus 1: Quersumme

1:Zuweisung(1)
2: Zuweisung für For-schleife (n)
3: Zuweisung(1) und Addition(1) und Dereferenzierung(1)
5: return(1)

Summe: $1 + n + n * (1 + 1 + 1) + 1$
$= 2 + 4 * n$

Algorithmus 2: (namenslos)

1: Zuweisung für For-schleife (n)
2: Zuweisung(1) und Dereferenzierung(1)
4: Zuweisung für For-schleife (n)
5: Zuweisung(1) und Dereferenzierung(1)
6: Zuweisung für For-schleife(n)
7: If-Abfrage(1) und Dereferenzierungen(2)
8: Zuweisung(1) und Dereferenzierungen(2)
12:return(1)

Summe: $(n * (1 + 1)) + (n * (1 + 1 + n * ((1 + 2) + (1 + 2)))) + 1$
$= 2*n + n*(2+n*6) + 1$
$= 2*n + 6*n^2 + 2*n + 1$
$= 6*n^2 + 4*n + 1$

Algorithmus 3: Matrixmultiplikation

1: Zuweisung für For-schleife (n)
2: Zuweisung für For-schleife (n)
3: Zuweisung(1) und Dereferenzierungen(1)*
4: Zuweisung für For-schleife (n)
5: Zuweisung(1), Multiplikation(1) und Dereferenzierungen(3)
9: return(1)

*Interpretation des Zugriffs auf einen Element eines 2dim-Arrays als einzelne Dereferenzierung trotz zwei Indizes

Summe: $n * (n * ((1 + 1) + n * (1 + 1 + 3))) + 1$
$= n * (n * (2 + n * 5)) + 1$
$= n * (2*n + 5*n^2) + 1$
$= 5*n^3 + 2*n^2 + 1$

Algorithmus 4: Allgemeines Beispiel

1: Zuweisung für For-schleife (n)
2: Zuweisung für For-schleife (i)*
3: Zuweisung(1), Addition(1) und Dereferenzierungen(1)
6: return(1)

* Wir nehmen an, dort sollte „to“ stehen statt einem „downto“, da sonst der Algorithmus sinnlos wäre.

Summe: $(1 + 2 + 3 + ... + n) * 3$
$= 3 * (n * (n + 1)) / 2$ (Gaussche Summenformel)
$= 1.5*n^2 + 1.5*n$

Die Drei in der ersten Zeile dieser Formeln kommt von den drei Operationen in der dritten Codezeile. Die aufsteigenden Zahlen der Summe sind eine Konsequenz der inneren For-Schleife die i-viele Iterationen macht.

Aufgabe 3.2.3:

Erwartung: Wir erwarten, dass bei wachsender Potenz k, die Laufzeit(bzw. die Anzahl der Rekursiven Aufrufe) der herkömmlichen iterativen Variante linear steigt. Bei der alternativen/rekursiven Variante erwarten wir eine logarithmische Komplexität da bei jedem Rekursionsaufruf das Problem in zwei geteilt wird, wobei beide Teilprobleme identisch sind und daher nur einmal brechnet werden müssen.

Ergebnisse: Siehe Excel Tabelle Sheet „Aufgb3“ -> Aufgabe 3

Interpretation: Unsere Erwartung hat sich bestätigt. Die Laufzeit des iterativen Variant ist liner, und die der alternativen ist logarithmisch. Z.B. ist bei k = 32 die iterative Variante auch bei 32 Aufrufen, während die alternative Variante bei 7 liegt.

(Hinweis: log(32) = 8) Feststellbar ist auch die Tatsache, dass die Anzahl der Aufrufe der alternativen Variante nur bei allen Zweierpotenzen steigt während die iterative Variante bei allen k steigt.

Aufgabe 3.2.4:

Erwartung: Wir erwarten, analog zu Aufgabe 3, dass der AccessCount bei der herkömmlichen Variante linear und bei der neuen Variante logarithmisch verläuft.

Ergebnisse: Siehe Excel Tabelle Sheet „Aufgb3“ -> Aufgabe 4

Interpretation: Unsere Erwartung hat sich im groben Verlauf der Kurven bestätigt. Aus dem gleichen Grund wie in Aufgabe 3, läuft die alterantive mit logarithmischen Aufwand und die herkömmliche in linearem Aufwand. Die kleinen Schwankungen in den AccessCounts bei der alternativen Implementation sind vermutlich eine Folge der Art und Weise, welche der beiden Fälle (ob ungerader oder gerader Fall) in den Rekursionsaufrufen verarbeitet werden. Da es möglich ist, dass zwei benachbarte Zahlen vollkommen unterschiedliche Anzahl an ungeraden und geraden Fällen haben, werden beide benachbarten Fälle stark unterschiedlichen AccessCount haben. Denn beim ungeraden Fall, wird in unserer funktionalen Implementation, eine komplette Matrix mehr erzeugt. Dessen AccessCount und die Schwankungen der Anzahl an geraden/ungeraden Fällen spiegeln sich in den Schwankungen des AccessCounts wieder.

Aufgabe 3.3: O-Notation

Aufgabenteil 1:
Zeige $15n^2 \in O(n^3)$:
Der Grenzwert ist $\lim_{n\to\infty} 15n^2/n^3 = \lim_{n\to\infty} 15/n = 0$.
Da der Grenzwert Null ist, folgt aus der Definition der O-Klasse, dass $15n^2 \in O(n^3)$.

Aufgabenteil 2:
Zeige $\mathrm{True} \neq (1/2n^3 \in O(n^2))$:
Der Grenzwert ist $\lim_{n\to\infty} 1/2n^3/n^2 = \lim_{n\to\infty} n/2 = \infty$
Da Der Grenzwert Unendlich ist, folgt aus der Definition der O-Klasse, dass $1/2n^3$ kein Element von $O(n^2)$

Aufgabenteil 3:
Sei $g(n) = 2n^2 + 3$:
a) $f_1(n) = 2n^2 + 2$ ist (für beliebiges n_0) stets kleiner als g(n) aber dennoch in O(g).
b)$f_2(n) = 2n^2 + 5$ ist (für beliebiges n_0) stets größer als g(n) aber dennoch in O(g).

Aufgabenteil 4:
Zeige, dass für zwei Polynome f und g gleichen Grades k gilt, dass $f \in \theta(g)$:
Dass f in der Theta-Klasse von g ist, wird durch den Grenzwert gezeigt.

Sei $f(n) = an^k + bn^{k-1} + \cdots + d$ und $g(n) = xn^k + yn^{k-1} + \cdots + z$, mit $a \neq 0$ und $x \neq 0$, dann ist der Grenzwert
$\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty}(an^k + bn^{k-1} + \cdots + d)/(xn^k + yn^{k-1} + \cdots + z)$.

Wir erweiteren Zähler und Nenner mit 1/(n^k):
$\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty}(a + bn^{-1} + \cdots + dn^{-n})/(x + yn^{-1} + \cdots + zn^{-n})$.

Alle Summanden in dem Bruch bis auf die vordersten Summanden werden bei größerne n immer kleiner.
$\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty}(a + \varepsilon)/(x + \varepsilon')$.

Zeigen dass (a+eps)/(x+eps‘) die eps irrelevant sind.
$\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty}(a + \varepsilon)/(x + \varepsilon')$.

Da im Zähler und Nenner die Epsilon immer weniger Einfluss haben, ergitb sich als Grenzwert
$\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} \frac{a+\varepsilon}{x+\varepsilon'} = \lim_{n\to\infty} \frac{a}{x} = \frac{a}{x}$.

Das Ergebnis ist weder Null noch Unendlich, da $a \neq 0$ und $x \neq 0$. Da der Grenzwert positive endliche Zahl ist, passt sie der Definition der Theta-Klasse: Es gilt $f \in \theta(g)$.

Aufgabenteil 5:
Zeige $\sum_{i=0}^{n} 2^i = f(n) \in O(2^n)$:
Zunächst wandeln wir f(n) in eine Form ohne Summenoperator um:

$$f(n) = 2^n + 2^{n-1} + \cdots + 2 + 1$$
$$2 * f(n) - f(n) = 2 * (2^n + 2^{n-1} + \cdots + 2 + 1) - (2^n + 2^{n-1} + \cdots + 2 + 1)$$
$$2 * f(n) - f(n) = (2^{n+1} + 2^n + \cdots + 2) - (2^n + 2^{n-1} + \cdots + 2 + 1)$$
$$2 * f(n) - f(n) = 2^{n+1} + (2^{n-1} - 2^{n-1}) + \cdots + (2 - 2) - 1$$
$$f(n) = 2^{n+1} - 1$$

Mit $\lim_{n\to\infty} f(n)/(2^n) = \lim_{n\to\infty}(2^{n+1} - 1)/(2^n) = \lim_{n\to\infty}(2 * 2^n - 1)/(2^n)$

$$= \lim_{n\to\infty} \frac{2 * 2^n - 2 * 0.5}{2^n}$$

$$= \lim_{n\to\infty} \frac{2(2^n - 0.5)}{2^n}$$

$$= \lim_{n\to\infty} 2$$

$$= 2$$

Damit ist gezeigt, dass $f(n)$ nicht nur in $O(2^n)$, sondern auch in $\theta(2^n)$ ist.