# Übungsblatt05 - Groß-O-Notation und Laufzeitanalysen - Musterlösung

TH Mittelhessen, FB MNI, Berechenbarkeit und Komplexität, Prof. Dr. B. Just

## Aufgabe 1

a.) Bitte schreiben Sie die folgenden Funktionen $f(n)$ in der Groß-O-Notation:

i.) $f(n) = n^2 + 2n + 4$ ii.) $f(n) = n^3 + \log_2 n$ iii.) $f(n) = n^2 + \log n + \sqrt{n}$
iv.) $f(n) = 2^n + 2n + 4$ v.) $f(n) = 3^n + \log_2 n$ vi.) $f(n) = n + \log n + \log\log n$

b.) Welche der Funktionen aus a.) erfüllen $f(n) = o(n^4)$?

## Lösung Aufgabe 1:

a.)

i.) $n^2 + 2n + 4 = O(n^2)$ ii.) $n^3 + \log_2 n = O(n^3)$ iii.) $n^2 + \log n + \sqrt{n} = O(n^2)$
iv.) $2^n + 2n + 4 = O(2^n)$ v.) $3^n + \log_2 n = O(3^n)$ vi.) $n + \log n + \log\log n = O(n)$.

b.) $f(n) = o(n^4)$?

i.) $n^2 + 2n + 4 = o(n^4)$ ii.) $n^3 + \log_2 n = o(n^4))$ iii.) $n^2 + \log n + \sqrt{n} = o(n^4)$
iv.) $2^n + 2n + 4 \neq o(n^4)$ v.) $3^n + \log_2 n \neq o(n^4)$ vi.) $n + \log n + \log\log n = o(n^4))$.

## Aufgabe 2

Betrachtet werden die folgenden Algorithmen:

i.) Addition zweier natürlicher Zahlen $n_1, n_2 \in \mathbb{N}$.

ii.) Addition zweier Brüche $z_1/n_1, z_2/n_2 \in \mathbb{Q}$, dargestellt durch ihre Zähler und Nenner.

iii.) Finden des Maximus von $n$ natürlichen Zahlen $s_1, s_2, ..., s_n \in \mathbb{N}$.

iv.) Sortieren der $n$ natürlichen Zahlen aus c.) mit Bubblesort.

v.) Multiplizieren zweier $n \times n$- Matrizen, deren Einträge alle natürliche Zahlen sind, die kleiner als eine obere Schranke $B \in \mathbb{N}$ sind.

vi.) Bestimme die Zahl der Zusammenhangskomponenten eines Graphen mit $n$ Knoten, der durch seine Adjazenzmatrix gegeben ist. Dabei wird angenommen, dass man in einem Schritt auf jeden Eintrag der Adjazenzmatrix zugreifen kann.

a.) Bitte geben Sie die Zahl der arithmetischen Operationen für alle Algorithmen in Groß-O-Notation an.

b.) Bitte geben Sie die Zahl der Bit-Operationen für alle Algorithmen in Groß-O-Notation an.

**Lösung Aufgabe 2:**

a.) / b.) Im folgenden werden jeweils Algorithmen angegeben, und ihre Laufzeit (großzügig) abgeschätzt.

i.) Addition zweier natürlicher Zahlen $n_1, n_2 \in \mathbb{N}$.

Algorithmus: Klassische Addition, stellenweise von rechts nach links, Überträge werden beachtet.

Laufzeit in arithmetischen Operationen: $O(1)$.

Laufzeit in Bitoperationen: $O(\log(n_1 + n_2))$, da $O(\log n)$ die Länge der Binärdarstellung einer Zahl $n$ ist.

ii.) Addition zweier Brüche $z_1/n_1, z_2/n_2 \in \mathbb{Q}$, dargestellt durch ihre Zähler und Nenner.

Algorithmus: Zähler und Nenner der Summe werden berechnet durch

$$\frac{z_1}{n_1} + \frac{z_2}{n_2} = \frac{z_1 \cdot n_2 + z_2 \cdot n_1}{n_1 \cdot n_2}$$

Laufzeit in arithmetischen Operationen: $O(1)$.

Laufzeit in Bitoperationen: Sei $B$ die größte der Input-Zahlen $z_1, z_2, n_1, n_2$. Dann ist die Laufzeit in Bitoperationen $O(\text{Inputlänge}^2) = 0((\log B)^2)$, da das die Zahl der Bitoperationen zur Multiplikation zweier ganzer Zahlen ist, deren Betrag höchstens $B$ ist.

iii.) Finden des Maximus von $n$ natürlichen Zahlen $s_1, s_2, ..., s_n \in \mathbb{N}$.

Algorithmus: Setze zunächst das Maximum auf die erste Zahl, vergleiche dann mit den restlichen Zahlen und ersetzte das Maximum, falls eine größere Zahl gefunden wird.

Laufzeit in arithmetischen Operationen: $O(n)$ Vergleiche.

Laufzeit in Bitoperationen: Sei $B$ die größte der Input-Zahlen $s_1, ..., s_n$. Dann ist die Laufzeit in Bitoperationen $O(n \cdot \log B)$, da $O(\log B)$ die Zahl der Bits jeder Input-Zahl ist, und jeder Vergleich höchstens so viele Bitoperationen benötigt, wie die größte Zahl Stellen hat.

iv.) Sortieren der $n$ natürlichen Zahlen aus c.) mit Bubblesort.

Algorithmus: Bubblesort durchläuft $n-1$ mal den Input, und vergleicht jeweils zwei aufeinanderfolgende Zahlen. Steht die größere links, werden beide vertauscht..

Laufzeit in arithmetischen Operationen: $O(n^2)$ Vergleiche, da $n$ mal durchlaufen wird, und jeweils $O(n)$ Vergleiche durchgeführt werden.

Laufzeit in Bitoperationen: Sei $B$ die größte der Input-Zahlen $s_1, ..., s_n$. Dann ist die Laufzeit in Bitoperationen $O(n^2 \cdot \log B)$, da $O(\log B)$ die Zahl der Bits jeder Input-Zahl ist, und jeder Vergleich höchstens so viele Bitoperationen benötigt, wie die größte Zahl Stellen hat.

v.) Multiplizieren zweier $n \times n$- Matrizen, deren Einträge alle natürliche Zahlen sind, die kleiner als eine obere Schranke $B \in \mathbb{N}$ sind.

Algorithmus: Jede Komponente wird ausgerechnet, indem man das Skalarprodukt der entsprechenden Zeile und Spalte der Input-Matrizen bildet.

Laufzeit in arithmetischen Operationen: $O(n^3)$ arithmetische Operationen, da $n^2$ Zahlen ausgerechnet werden, und das Ausrechnen jeder Zahl $n$ Multiplikationen und $n-1$ Additionen benötigt.

Laufzeit in Bitoperationen: Wir schätzen hier sehr großzügig ab, um nicht in die Datails gehen zu müssen. Sei $L$ die Länge des gesamten Inputs (die Zahlen sind also in der Größenordnung von $2^{O(L)}$). Dann braucht eine arithmetische Operation höchstens $O(L^2)$ Bitoperationen. Die Zahl der Bitoperationen ist also insgesamt höchstens $O(n^3 \cdot L^2)$.

vi.) Bestimme die Zahl der Zusammenhangskomponenten eines Graphen mit $n$ Knoten, der durch seine Adjazenzmatrix gegeben ist. Dabei wird angenommen, dass man in einem Schritt auf jeden Eintrag der Adjazenzmatrix zugreifen kann.

Die Knoten seien mit $1, 2, ..., n$ bezeichnet. Sie werden nach und nach markiert (das bedeutet, dass sie in einer bereits gezählten ZSK sind), und dann gestrichen (das bedeutet, dass ihre Nachbarn in der ZSK vollständig markiert sind).

Algorithmus:

1.) Setze die Anzahl der Zusammenhangskomponenten auf 0.

2.) Wenn es noch einen nicht gestrichenen Knoten gibt: Wähle den nicht gestrichenen Knoten aus, dessen Nummer am kleinsten ist. Erhöhe die Anzahl der ZSK um 1.

2.1.) Markiere alle Nachbarn des Knoten und streiche den Knoten selbst.

2.2.) Führe Schritt 2.1. solange aus, wie es noch markierte Knoten gibt.

Laufzeit in arithmetischen Operationen: $O(n^3)$ arithmetische Operationen. Denn in Schritt 2.1. wird jedesmal ein Knoten gestrichen, der Schritt kann also höchstens $n$ mal durchlaufen werden. Jeder Durchlauf führt höchsten $O(n)$ Markierungen durch. Die Gesamtlaufzeit sind also höchstens $O(n^2)$ arithmetische Operationen.

Laufzeit in Bitoperationen: Jede arithmetische Operation durchläuft höchstens konstant oft den gesamten Input, um die Knoten zu finden bzw. zu markieren bzw. zu streichen. Die Zahl der Bitoperationen ist also $O(n^3 \cdot L)$, wobei $L$ die Länge des Inputs in Bits ist.

## Aufgabe 3

Bitte erfinden Sie Algorithmen, die die folgenden Berechnungsprobleme durch Ausprobieren lösen, und geben Sie die Zahl der im worst case benötigten Rechenschritte in der Groß-O-Notation an:

a.) Faktorisierung einer Zahl $n = p \cdot q$.
Hier soll die Zahl der benötigten Rechenschritte in Abhängigkeit von der Eingabelänge angegeben werden.

b.) Finden einer erfüllenden Belegung einer Boole'schen Formel.
Hier soll die Zahl der benötigten Rechenschritte in Abhängigkeit von der Anzahl der Boole'schen Variablen angegeben werden.

c.) Finden eines Weges in einem Graphen von einem Knoten s zu einem Knoten t.
Hier soll die Zahl der benötigten Rechenschritte in Abhängigkeit von der Anzahl der Knoten des Graphen angegeben werden.

## Lösung Aufgabe 3:

a.) Faktorisierung einer Zahl $n = p \cdot q$.
Hier soll die Zahl der benötigten Rechenschritte in Abhängigkeit von der Eingabelänge angegeben werden.

Algorithmus: Probiere für $i = 2, 3, 4, 5, ...\lfloor\sqrt{n}\rfloor$ aus, ob $n$ ohne Rest durch $i$ teilbar ist. Auf diese Art wird die kleinere der Zahlen $p$ und $q$ gefunden, da diese kleiner als $\sqrt{n}$ ist.

Laufzeit: Es werden $O(\sqrt{n})$ arithmetische Operationen ausgeführt. Die Eingabelänge ist aber nur $L = O(\log n)$. Die Zahl der arithmetischen Operationen ist also $O(\sqrt{2^L}) = O(2^{L/2})$. Der Algorithmus ist nicht polynomiell.

b.) Finden einer erfüllenden Belegung einer Boole'schen Formel.
Hier soll die Zahl der benötigten Rechenschritte in Abhängigkeit von der Anzahl der Boole'schen Variablen angegeben werden.

Algorithmus: Probiere für jede der $2^n$ möglichen Belegungen der Variablen aus, ob diese Belegung die Formel erfüllt.

Laufzeit: Die Formel wird $2^n$ mal ausgewertet. Das Auswerten einer Formel dauert höchstens $O(L^2)$ Schritte, wenn $L$ die Länge des Inputs ist (diese Abschätzung ist wieder großzügig). Insgesamt werden also $O(2^n \cdot L^2)$ Schritte ausgeführt. Der Algorithmus ist nicht polynomiell.

Bemerkung: Die Laufzeit kann nicht durch einen Ausdruck nach oben abgeschätzt werden, der nur von $n$ abhängt. Denn die Formel kann beliebig lang sein.

c.) Finden eines Weges in einem Graphen von einem Knoten s zu einem Knoten t.
Hier soll die Zahl der benötigten Rechenschritte in Abhängigkeit von der Anzahl der Knoten des Graphen angegeben werden.

Algorithmus: Betrachte alle Permutationen der Knoten außer s und t. Prüfe für jede Permutation und jeden Knoten v innerhalb der Permutation, ob man bei s starten kann, dann den Weg bis v gehen kann und dann von v nach t gehen kann.

Der Algorithmus ist natürlich brute force, aber er findet offenbar einen Weg, wenn es einen gibt.

Laufzeit: Bei $n$ Knoten im Graphen werden $(n-2)!$ Permutationen betrachtet. Für jede Permutation werden $O(n)$ Schritte durchgeführt, um zu testen, ob die Kanten im Graphen vorhanden sind, und ob man von einem Knoten v aus zu t gehen kann. Insgesamt sind es $O(n!/(n-1)) = O((n-1)!)$ Schritte. Der Algorithmus ist nicht polynomiell.

Bemerkung: Der schnellste Algorithmus für das Problem ist der Dijkstra-Algorithmus. Er braucht $O(m+n\log n)$ Schritte, wobei $m$ die Anzahl der Kanten ist. Das sind höchstens $O(n^2)$ Schritte, also polynomiell.

## Aufgabe 4

Bitte arbeiten Sie im Sipser die Seiten 275-282 durch. Den Teil mit der Simulation einer zwei-Band TM durch eine ein-Band TM können Sie weglassen :).
Am Ende sollten Sie fit in der Groß-O-Notation sein, und die Laufzeitanalyse aus Algorithmen und Datenstrukturen wieder präsent haben :).

## Lösung Aufgabe 4:

Hier gibt es keine Musterlösung :)