

Lösungsblatt Nr. 1

Aufgabe 1

Das Konzept der *random access machine* (RAM) ist ein idealisierter Rechner, der beliebig genau mit reellen Zahlen operiert. Zu allem Überfluß kann diese Maschine die vier Grundrechenarten auch noch ausführen, ohne Zeit zu verbrauchen. Lediglich Entscheidungen, die im laufenden Programm Verzweigungen auslösen, benötigen je einen Zeitschritt.

a) Rechenzeit

Sowohl auf realen Prozessoren als auch auf einer Turingmaschine verursachen Rechenoperationen Zeitaufwand. (Da dieser bei einer gebastelten oder simulierten Turingmaschine sehr bald immens wird, soll diese nicht weiter betrachtet werden.)

Aufgrund massiver Parallelverarbeitung mittels hochgezüchteter Spezial-Hardware (früher numerische Kopprozessoren, heute integriert auf dem Hauptprozessor) kann die zu veranschlagende Zeit für arithmetische Grundoperationen mit $O(1)$ angegeben werden.

b) Verzweigungen

Der “algorithmische Gehalt” eines Programms liegt wesentlich in den Kontrollstrukturen. Diese Kontrollstrukturen stellen sich aber im Ablaufbaum (siehe Vorlesung) wie folgt dar:

- Fallunterscheidungen lösen wie auch im Programmcode Verzweigungen aus, nur führen diese nach einem **end if** (ggf. iteriert als **end case**) nicht wieder zusammen – die Baumstruktur wäre sonst nicht mehr zyklensfrei –, sondern der darauffolgende Ablauf findet sich als Unterbaum unter jeder Verzweigungsalternative wieder. Denn in den alternativen Zweigen werden in aller Regel unterschiedliche Berechnungen angestellt, und so wird auch ein anderes Ergebnis A gewonnen, mithin eine andere Ziel-funktion f berechnet. (Sollte dies nicht der Fall sein, so verzweigt sich der Ablaufbaum gar nicht.)
- Schleifen (egal, ob es sich um **for**, **while** oder **repeat** handelt) werden komplett “ausgerollt”, d. h. jedes (zur Laufzeit: potentielle) Schleifenende ergibt eine eigene Abzweigung. Die Abbruchbedingung der Schleife ist dabei der Verzweigungstest “Ist $B(\dots) > 0$?” im Ablaufbaum.

Anhand dieser Betrachtungen wird deutlich, daß Ablaufbaum und *random access machine* reale Maschinen nicht in bezug auf Genauigkeit oder Rechenzeit simulieren, sondern eher hinsichtlich des Programmablaufs und der Beschaffenheit der Grundoperationen; ein Vergleich $B(\dots) \stackrel{?}{>} 0$ benötigt tatsächlich eine gewisse Zeit.

Daß Rechenoperationen auf der RAM keine Zeit verbrauchen, Vergleichsoperationen aber sehr wohl, kann so interpretiert werden, daß jeder *Knoten* des Ablaufbaumes $O(1)$ Schritte benötigt. Nun sind es der Berechnungen in jedem Knoten ja – durch die Länge des Programmkodes begrenzt – endlich viele, sagen wir $O(1)$. Wenn wir diese Zeit auf die Vergleichsoperation umlegen, wird das RAM-Modell mitsamt dem Ablaufbaum also sehr realistisch. (Berechnungen mit impliziten Kontrollstrukturen wie $\sum_{i=1}^n \dots$ oder $\prod_{i=1}^n \dots$ müßten für beliebig große n allerdings redlicherweise unterbleiben bzw. in Kontrollstrukturen mit Verzweigungen transformiert werden.)

c) **Genauigkeit**

Aufgrund der praktisch begrenzten Genauigkeit kann heute keine numerische Recheneinheit mit irrationalen, also reellen Zahlen operieren. Die sog. Maschinenzahlen, in aller Regel rationale Zahlen, haben eine endliche Genauigkeit, die durch die Anzahl der Fließkommastellen bestimmt ist.

In einem Punkt kommt die RAM realen Computern bzw. Prozessoren allerdings sehr nahe: Potenzreihen wie Logarithmus, Sinus, Exponentialfunktion u. ä. lassen sich nur bis zu einer endlichen Iterationstiefe berechnen, wenn die Berechnung terminieren soll. (Die Genauigkeit ist ja ohnehin durch die Anzahl der Stellen begrenzt.) Alle derartigen Näherungen für solche mathematischen Funktionen sind somit *rationale Funktionen* im Sinne der Vorlesung, also endliche Terme mit Addition, Subtraktion, Multiplikation und Division.

d) **Elementaroperationen**

Noch einmal zur Übersicht:

Modell	Elementaroperationen
Random Access Machine	arithmetische Grundoperationen; Zuweisung an Variable; Größenvergleich mit Null
realer Prozessor	arithmetische Grundoperationen; Zuweisung an Variable; Größenvergleich; Sprung- und andere Programmsteuerungs-Operationen (bedingte und unbedingte)
Turingmaschine	ein Eingabezeichen von (linearem) Band lesen, dies mit aktuellem Maschinenzustand verknüpfen, daraus einen neuen Zustand erhalten und Schreib-/Lesekopf auf dem Band um max. 1 Position verschieben, dorthin ein Ausgabezeichen schreiben

Aufgabe 2

a) Bedingungen für die Baum-Eigenschaft eines gerichteten Graphen:

- Die wichtigste (und intuitiv unmittelbar einleuchtende) Eigenschaft ist die **Zyklenfreiheit**. Dies impliziert insbesondere die Schleifenfreiheit, d. h., daß kein Knoten auf sich selbst verweist.
- Weiterhin darf es nur **eine Wurzel** geben.¹ Sonst hätte man es mit einem Wald, also mehreren Bäumen, zu tun.
- Eine weitere Forderung ist, daß der Baum, auch wenn man die Kanten als ungerichtete interpretieren würde, zyklensfrei bleiben muß. Die Ver-

¹Daß es *mindestens* eine Wurzel gibt, geht bereits aus der zyklensfreien Eigenschaft hervor.

zeigerung muß daher **injektiv** sein, d. h., keine zwei Zeiger dürfen auf denselben Knoten verweisen.²

Fallen Ihnen noch weitere Bedingungen ein, die *nicht* von den hier aufgezählten erschlagen werden? Mir nicht.

- b) Nach ein wenig Ausprobieren kommt man schnell darauf, daß ein Binärbaum mit n Knoten stets $m = n + 1$ leere Zeiger enthält. Dies ist also sowohl der Minimal- als auch der Maximalwert.

Beweis durch *bottom-up*-Induktion über die Struktur des Baumes:

- **Induktionsbeginn:** Der betrachtete Teilbaum besteht aus nur einem Blatt; in diesem stehen also zwei leere Zeiger.

Es ist offenbar $n = 1$ und $m = 2$, die Behauptung $m = n + 1$ ist also erfüllt.

- **Induktionsschritt:** Die Wurzel V des betrachteten Teilbaums ist kein Blatt.

Fall A: der Knoten V hat genau ein Kind, für dessen Unterbaum gemäß Induktionsvoraussetzung bereits $m_0 = n_0 + 1$ gilt. Da V selbst ein zusätzlicher Knoten ist, ergibt sich

$$\begin{aligned} m &= m_0 + \underbrace{1}_{\text{Zeiger}} \\ &= (n_0 + 1) + 1 \\ &= (n_0 + \underbrace{1}_{\text{Wurzel}}) + 1 \\ &= n + 1 \end{aligned}$$

Fall B: der Knoten V hat zwei Kinder, für die gemäß Induktionsvoraussetzung bereits jeweils $m_L = n_L + 1$ und $m_R = n_R + 1$ gilt. Da V selbst ein zusätzlicher Knoten ist, ergibt sich

$$\begin{aligned} m &= m_L + m_R \\ &= (n_L + 1) + (n_R + 1) \\ &= (n_L + n_R + \underbrace{1}_{\text{Wurzel}}) + 1 \\ &= n + 1 \end{aligned}$$

q. e. d.

- c) Die Gesetzmäßigkeit ist bekannt: ein binärer Baum der Höhe h hat mindestens $m = h + 1$ Knoten (nämlich dann, wenn er zur linearen Liste degeneriert ist). Ein vollbesetzter Baum der Höhe h erreicht den Maximalwert von $n = 2^{h+1} - 1$ Knoten.

Formal ganz streng und pedantisch, wollen wir beides abermals über eine strukturelle Induktion beweisen.³

Behauptung Nr. 1:

$$m = h + 1$$

Beweis:

²Diese Forderung impliziert insbesondere, daß keine Mehrfach-, also in unserem Falle: Doppelkanten existieren.

³Obwohl die Aussage über m unmittelbar einleuchtet, und die Aussage über n auch mittels der geometrischen Reihe $n = \sum_{i=0}^h 2^i = 2^{h+1} - 1$ zeigen ist.

- **Induktionsbeginn:** Der betrachtete Teilbaum besteht aus nur einem Blatt; es ist offenbar $m = 1$ und $h = 0$, die Behauptung $m = h + 1$ ist also erfüllt.
- **Induktionsschritt:** Die Wurzel V des betrachteten Teilbaums ist kein Blatt, hat also mindestens ein Kind. Weil wir den minimalen Fall betrachten, heißt dies: genau ein Kind. Für den bei diesem Kind beginnenden Unterbaum gilt gemäß Induktionsvoraussetzung $m_0 = h_0 + 1$. Wir erhalten:

$$\begin{aligned}
 m &= m_0 + \underbrace{1}_{\text{Wurzel}} \\
 &= (h_0 + 1) + 1 \\
 &= (h_0 + \underbrace{1}_{\text{Niveau}}) + 1 \\
 &= h + 1
 \end{aligned}$$

Behauptung Nr. 2:

$$n = 2^{h+1} - 1$$

Beweis:

- **Induktionsbeginn:** Der betrachtete Teilbaum besteht aus nur einem Blatt; es ist offenbar $n = 1$ und $h = 0$, also ist auch die Behauptung $n = 2 - 1 = 2^{h+1} - 1$ erfüllt.
- **Induktionsschritt:** Die Wurzel V des betrachteten Teilbaums ist kein Blatt. Weil wir den Maximalfall betrachten, hat V genau zwei Kinder, für deren Unterbäume gemäß Induktionsvoraussetzung jeweils $n_0 = 2^{h_0+1} - 1$ gilt. Da außerdem $h = h_0 + 1$ ist, ergibt sich:

$$\begin{aligned}
 n &= 2n_0 + \underbrace{1}_{\text{Wurzel}} \\
 &= 2 \cdot (2^{h_0+1} - 1) + 1 \\
 &= 2 \cdot 2^{h_0+1} - 2 + 1 \\
 &= 2^{h_0+2} - 2 + 1 \\
 &= 2^{h+1} - 1
 \end{aligned}$$

q. e. d.

- d) Nach ein paar Skizzen schält sich die Vermutung heraus, daß ein Graph mit $V = 2n$ Knoten bis zu $E = n^2$ Kanten enthalten kann, ohne daß *Untergraphen von drei je paarweise zusammenhängenden Knoten* (die wir im folgenden der Einfachheit halber "Dreiecke" nennen wollen) entstehen.

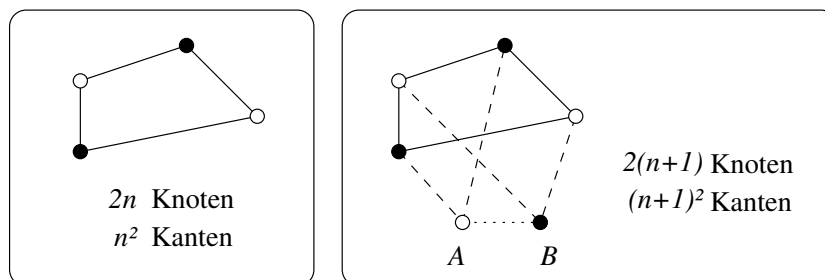
Beweisidee: Wir versuchen uns abermals an einer vollständigen Induktion. Diese läuft allerdings nicht über die Struktur des Graphen, da dieser per se keine 'Ordnung' hat (nicht einmal die Kanten sind ja gerichtet), sondern wir über ihn recht wenig wissen. Die Induktion läuft also über die Anzahl der Knoten:

- **Induktionsbeginn $n = 1$:** Wir haben $V = 2n = 2$ Knoten und $E = n^2 = 1$ Kante. Offenbar läßt sich hier kein Dreieck finden.⁴

⁴Wem dieser Fall zu scheel ist, da hier ja ohnehin nicht weitere Kanten hinzugenommen werden könnten, der/die beginne mit $n = 2$: ein Viereck mit $V = 2n = 4$ Knoten und $E = n^2 = 4$ Kanten ist sicherlich der maximale Graph ohne Dreiecke.

- **Induktionsschritt** $(n) \Rightarrow (n+1)$: Wir betrachten also einen Graphen G mit $2n$ Knoten, für den wir es schon als erwiesen ansehen, daß er mit n^2 Kanten maximal vernetzt ist, sodaß keine Dreiecke entstehen (**Induktionsvoraussetzung**). Es ist zwar nicht vollkommen evident, aber doch naheliegend,⁵ daß ein solch maximal vernetzter Graph *bipartit* ist. Das bedeutet, daß sich die Knoten von G in zwei Klassen (disjunkte Mengen S und T) einteilen lassen; zwischen zwei Knoten unterschiedlicher Klassen darf⁶ dann immer eine Kante gezogen werden, zwischen Knoten gleicher Klasse nicht.⁷ Es ist offensichtlich, daß so kein Dreieck entstehen kann, denn mindestens zwei seiner Knoten würden in dieselbe Klasse fallen – und wären damit nicht verbunden.

Um den Induktionsschritt nun zu vollziehen, nehmen wir zwei weitere Knoten, A und B , hinzu. Ohne Beschränkung der Allgemeinheit schlagen wir A der Klasse S zu und B der Klasse T .⁸ Dann dürfen wir A mit jedem Knoten aus T , und B mit jedem Knoten aus S verbinden. Dies sind genau $2n = |V|$ neue Kanten, da so jeder Knoten aus G eine neue Kante bekommt. Außerdem verbinden wir A und B , wodurch eine weitere Kante entsteht.



Wir haben also $2n + 1$ neue Kanten für 2 neue Knoten.

Laut Induktionsvoraussetzung hat der ursprüngliche Graph (mit $2n$ Knoten) n^2 Kanten. Wir müssen nun nur noch zeigen, daß der erweiterte Graph (mit $2n + 2$ Knoten und den alten + neuen $= n^2 + (2n + 1)$ Kanten) nun also $(n + 1)^2$ Kanten hat. Dies sagt aber bereits die binomische Formel.

q. e. d.

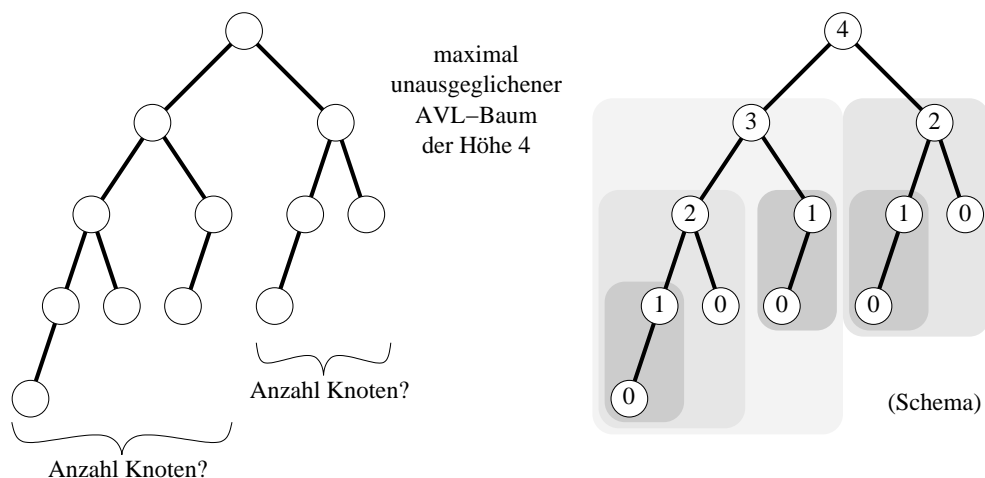
⁵ein Beweis dieser Behauptung wird hier nicht streng geführt; oder doch? Eigentlich wäre noch zu zeigen, daß mit $n^2 + 1$ Kanten notwendig Dreiecke entstehen.

⁶und muß (da wir ja den maximal vernetzten Fall betrachten wollen)

⁷Weiterhin kann festgestellt werden, daß die Eigenschaft der “maximalen Vernetzung ohne Dreiecke” dann eintritt, wenn diese beiden Klassen gleich groß sind. Jeder Knoten aus S darf mit jedem Knoten aus T verbunden werden; dann gibt es genau $|S| \cdot |T|$ Kanten. Dieses Produkt $|S| \cdot |T| = |S| \cdot (V - |S|)$ wird für festes $|V|$ aber dann maximal, wenn $|S| = |T| = V/2$ ist. Und weil $V/2 = n$ ist, ist $|E| = |S| \cdot |T| = n^2$.

⁸Wir könnten auch A und B derselben Klasse zuteilen; dann dürften wir sie allerdings nicht verbinden. Der einzige Unterschied, der beim Durchspielen dieses Falles gegenüber dem im fortlaufenden Text ausgeführten entsteht, ist diese fehlende Verbindung (vorausgesetzt, daß vorher $|S| = |T|$ war). Sie verhindert, daß dieser Fall maximal viele Kanten hervorbringt. (Vgl. vorangehende Fußnote)

Aufgabe 3



- a) Für die Anzahl der Knoten n_h eines höchst unausgewogenen Baums der Höhe h , der gerade noch die 1962 von Adel'son-Vel'skii und Landis ("AVL") in der UdSSR formulierte Bedingung erfüllt, läßt sich die Rekurrenzrelation

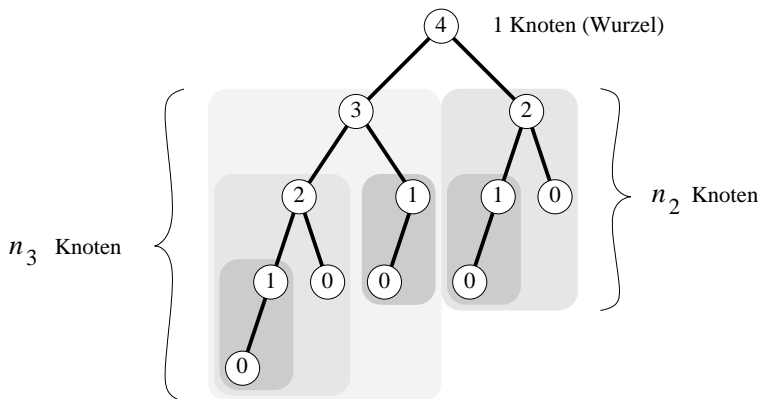
$$n_h = 1 + n_{h-1} + n_{h-2}$$

mit den Basisfällen

$$n_0 = 1 \quad n_1 = 2$$

aufstellen.

Die Berechnung von n_4 ergibt sich demnach aus:



Da sich nun bei einem AVL-Baum die Höhen der beiden Teilbäume nur um höchstens 1 unterscheiden dürfen – und dies in unserer *worst-case*-Betrachtung auch tun –, bestimmt sich das Verhältnis von linkem und rechtem Teilbaum also auf

$$\begin{aligned} \frac{n_L}{n_R} &= \frac{n_{h-1}}{n_{h-2}} = \frac{1 + n_{h-2} + n_{h-3}}{n_{h-2}} \\ &= \frac{1}{n_{h-2}} + 1 + \frac{n_{h-3}}{n_{h-2}} \end{aligned}$$

- b) Wir nehmen realistischerweise an, daß der gesuchte Wert $\gamma := \lim_{h \rightarrow \infty} n_L/n_R$ existiert und $\neq 0$ ist. Dann gilt wegen (a)

$$\gamma = \lim_{h \rightarrow \infty} \frac{n_L}{n_R} = \lim_{h \rightarrow \infty} \frac{n_{h-1}}{n_{h-2}}$$

$$\begin{aligned}
&= \lim_{h \rightarrow \infty} \left(\frac{1}{n_{h-2}} + 1 + \frac{n_{h-3}}{n_{h-2}} \right) \\
&= 0 + 1 + \lim_{h \rightarrow \infty} \frac{n_{h-2}}{n_{h-1}} \quad (\text{Indexverschiebung}) \\
&= 1 + \frac{1}{\gamma}
\end{aligned}$$

Multiplikation mit γ liefert die quadratische Gleichung $\gamma^2 - \gamma - 1 = 0$. Es folgt, daß γ dem sogenannten Goldenen Schnitt genügt:

$$\gamma = \frac{1 + \sqrt{5}}{2} \approx 1,6180$$

- c) In einem optimalen binären Baum mit n Knoten sind alle Höhenniveaus bis auf das höchste, $h = \lfloor \log_2 n \rfloor + 1$, voll besetzt. Im schlechtesten Fall werden in einem solchen Baum zur Elementsuche h Schritte benötigt, da in jedem Schritt ein Teilbaum mit $\approx 1/2 = 50\%$ der zuvor betrachteten Knoten ausgeschlossen wird.

Im Falle eines schlechtestmöglich balancierten AVL-Baumes, wie er in dieser Aufgabe betrachtet worden ist, werden in jedem Suchschritt nur mindestens $\approx 1/(1+\gamma) \approx 38,2\%$ der zuvor betrachteten Knoten ausgeschlossen, wenn dies nämlich der jeweils kleinere Teilbaum ist (vgl. (b)). Die Suche wird dann im jeweils größeren Teilbaum fortgesetzt, der noch $\approx \gamma/(1+\gamma) \approx 61,8\%$ der zuvor betrachteten Knoten enthält. Daher sind im Falle dieses schlechtestmöglich balancierten AVL-Baumes mit n Knoten bis zu $h \approx \log_{(1+\gamma)/\gamma} n = \log_\gamma n$ Vergleiche vonnöten.

(Das Schöne am Goldenen Schnitt γ ist, daß $1/(\gamma+1) = 2-\gamma$ und $\gamma/(\gamma+1) = \gamma-1$ und $(1+\gamma)/\gamma = \gamma$ ist, denn es ist ja $\gamma^{-1} = \gamma-1$.)

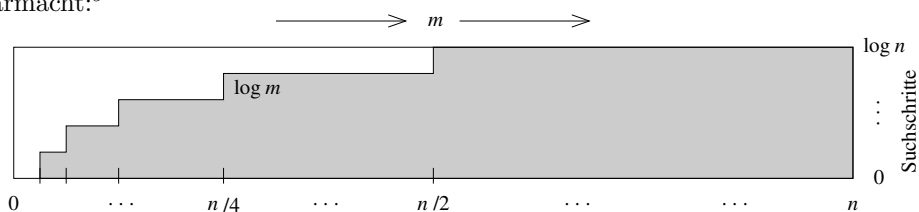
Asymptotisch verbleibt die Effizienz der binären Suche im schlechtesten gegen den optimalen Fall aber bis auf den konstanten Faktor $c = \log 2 / \log \gamma = \log_\gamma 2 \approx 1,440$ in derselben Funktionenklasse $O(\log n)$.

Ein höchst unausgewogener AVL-Baum der Höhe h ist übrigens gleichzeitig ein AVL-Baum der Höhe h mit minimaler Knotenanzahl.

Aufgabe 4

Vorgehen 1: Alle Platten werden nacheinander aus ihren sortierten Hüllen gezogen, und schrittweise eine sortierte Folge der Platten aufgebaut. Diese sortierte Folge der Platten wird schließlich iterativ in die immer noch sortierte Folge der leeren Hüllen gesteckt.

Der Aufwand für dieses einfache Verfahren beläuft sich auf genau n Schritte für das Herausnehmen einer jenen Platte aus ihrer Hülle; das Aufbauen einer sortierten Folge dauert elementweise $(1 + \log_2 m) + 1$ Schritte (binär suchen + einfügen), wenn die Folge aktuell m Elemente enthält ($m = 0, \dots, n-1$), also insgesamt weniger als $n \cdot (1 + \log_2 n) + n$ Schritte, nämlich genau $n \cdot (1 + \log_2 n)$, wie man sich leicht klarmacht:⁹



⁹Den nichtschraffierten Teil des länglichen Rechtecks sparen wir; er läßt sich durch die geometrische Reihe $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots = n$ abschätzen.

Schließlich sind noch die sortierten Platten in die sortiert vorliegenden Hüllen zu stecken, was genau n Schritte erfordert. Alles in allem ergibt sich ein Aufwand von

$$n + n \cdot (1 + \log_2 n) + n = 3n + n \cdot \log_2 n$$

Schritten, im O-Kalkül ausgedrückt liegt dies in der Komplexitätsklasse $\Theta(n \cdot \log n)$.

Vorgehen 2: Alle Platten werden nacheinander aus ihren sortierten Hüllen gezogen, und (nicht notwendig sortiert) aufgereiht. Dann werden diese unsortierten Platten iterativ abgearbeitet und jeweils in die passende, nun leere Hülle gesteckt.

Wir haben nun n Schritte für das Herausnehmen der Platten und n Schritte für das Hineinstecken der Platten. Da zwischendurch aber noch die richtige Hülle gefunden werden muß (bis auf diesen Vorgang ist das Verfahren ja identisch mit *Bucket Sort*), sind für diesen Vorgang mit binärer Suche jeweils $1 + \log_2 n$ Schritte zu veranschlagen. Es resultieren wieder insgesamt

$$n + n \cdot (1 + \log_2 n) + n = 3n + n \cdot \log_2 n$$

Schritte; der asymptotische Aufwand liegt immer noch bei $\Theta(n \log n)$.

Vorgehen 3: Wir nehmen an, daß unser Freund sicher nicht alle Platten der umfangreichen Sammlung gehört und in die falschen Hüllen permutiert hat. Daher bietet sich das folgende Verfahren an:

- (1) Suche aufsteigend in den geordneten Hüllen die erste Platte, die in einer falschen Hülle steckt; merke die Nummer m dieser Hülle.
- (2) Suche zur gefundenen Platte die passende Hülle und stecke sie dort hinein.
Mit der dort vorgefundenen (notwendig falschen) Platte verfare wie in Schritt (2), und zwar, bis der Permutationszyklus bei Hülle Nr. m wieder endet.
- (3) Suche aufsteigend von m weiter in der sortierten Folge der Plattenhüllen irgendwelche falsch einsortierten Platten (Schritt (1)).

Der Algorithmus endet, wenn $m = n$ wird.

Der Aufwand für Schritt (1) bzw. (3) ist: n mal eine Platte aus ihrer Hülle herausziehen und wieder hineinstecken, selbst wenn sie richtig eingetütet war; also $3n$ Zeittakte inklusive Vergleich auf Gleichheit der Schlüssel.

Jede falsch eingeordnete Platte wird irgendwann in Schritt (2) genau einmal von der falschen in die richtige Hülle gesteckt, die ansonsten am korrekten Ort verbleibt. Da der gute Freund genau k Platten falsch einsortiert hat, sind dies genau $k \cdot (1 + (1 + \log_2 n) + 1)$ Zeittakte (Platte herausziehen + Hülle suchen + dort hinein).

Insgesamt beläuft sich der Aufwand für dieses optimierte Verfahren auf

$$3n + k \cdot (1 + (1 + \log_2 n) + 1) = 3n + 3k + k \log_2 n$$

Schritte. Dieses Ergebnis ließe sich noch verbessern, da die binäre Suche ja nur im Bereich von $m + 1, \dots, n$ operieren muß und nicht global auf $1, \dots, n$.¹⁰

Asymptotisch macht sich der reduzierte Aufwand als $O(n + k \log n)$ spürbar.

¹⁰Mit einer Argumentation, die ähnlich läuft wie in der Abb. zu Verfahren 1, ließe sich – unter Annahme einer Gleichverteilung der k falschen Platten auf die n Hüllen – vielleicht zeigen, daß das Gesamtergebnis $3n + k \cdot (1 + (1 + \log_2 n) + 1) - k = 2n + 2k + k \log_2 n$ Schritte beträgt.

Bemerkung

Man kann schneller als binär suchen, wenn man gemäß einer Metrik (Abstandsmaß) auf den Schlüsseln ungefähr den Ort in der sortierten Folge vorausahnt, an dem ein bestimmter Schlüssel – bei Gleichverteilung – liegen müßte. Wie bei der binären Suche iteriert man dieses Vorgehen, bis der richtige Schlüssel gefunden ist.

Das Verfahren ist als *interpolierende Suche* bekannt und wurde 1957 zuerst formal definiert – obwohl es wohl seit Menschengedenken sowieso jeder benutzt, der in einem Telefonbuch oder Lexikon einen Eintrag sucht. In den 70er Jahren haben verschiedene Informatiker und Mathematiker (z. B. Yao & Yao oder auch Perl, Itai & Avni) unabhängig voneinander bewiesen, daß der Aufwand den extrem günstigen Wert von $O(\log \log n)$ hat.

Die Gleichverteilungsannahme kann übrigens entfallen, wenn die Wahrscheinlichkeitsverteilung (ungefähr) bekannt ist. Dann kann die Metrik nämlich dieser Verteilung angepaßt und so die Interpolation “entzerrt” werden.