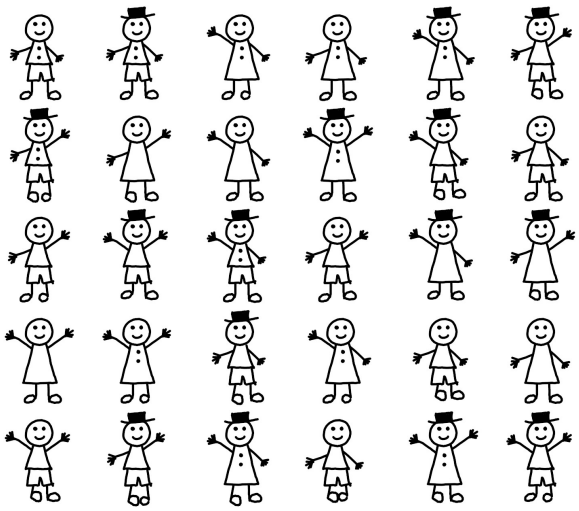
Universität Stuttgart

Beispielaufgabe zur Studienwahl

Das doppelte Lottchen: Suchen und Sortieren

© Michael Eisermann, Stefan Kohl, Friederike Stoll

Sie machen ein Praktikum bei einer Versicherung. Jeder der 10 Millionen Kunden hat eine 15-stellige Kundennummer und weitere Versicherungsdaten, die in einer Datenbank abgespeichert sind. Diese Datenbank hat vorerst noch keine besondere Struktur, insbesondere ist sie (noch) nicht nach Kundennummern sortiert. Sie sollen überprüfen, ob alle Kundennummern wirklich verschieden sind. Dazu verwenden Sie einen Computer mit Ihrer Lieblingsprogrammiersprache. Ein Vergleich $(<, =, >)$ kostet etwa 1 Mikrosekunde, sonstige Operationen sind vernachlässigbar (Indexrechnung, nachschlagen, umordnen, etc). Beispielsweise benötigt das Auffinden der kleinsten Kundennummer $9\,999\,999$ Vergleiche und dauert etwa 10 Sekunden.



Wo ist das Doppelgängerpaar?

Wenn Sie jedes Paar auf Dopplung prüfen, wie lange benötigt der Computer?

Wie lange dauert die Prüfung auf Dopplungen, wenn die Datenbank sortiert vorliegt?

Leider ist die Datenbank völlig unsortiert. Ihre Chefin möchte dies ändern und betraut Sie mit einer möglichst effizienten Sortierung. Diese soll ebenfalls nur paarweise Vergleiche nutzen und selbst im ungünstigsten Fall möglichst schnell sein. Sie gibt Ihnen noch den Tipp, dass Sie in der Literatur oder im Internet nach bekannten vergleichsbasierten Verfahren schauen sollten.

Wie lange benötigt der Computer mit einem solchen Verfahren zum Sortieren?

Beantworten Sie jede der drei Fragen mit einem der folgenden Zeitintervalle:

unter 1 Sek.	1 bis 5 Sek.	5 bis 20 Sek.	20 bis 60 Sek.	1 bis 3 Min.	3 bis 5 Min.	5 bis 60 Min.
1 bis 24 Stunden	1 bis 7 Tage	1 bis 4 Wochen	1 bis 12 Monate	1 bis 3 Jahre	3 bis 10 Jahre	länger als 10 Jahre

Weitere Aufgaben und Informationen unter:
Studienwahl-Kompass Mathematik
Universität Stuttgart

mathematik
fakultät8

Stufe 0 / Kurzantwort: (1) Beim Durchlaufen aller Paare führt der Computer 50 Billionen Vergleiche durch, dafür benötigt er unerträglich lange **1½ Jahre**. (2) Liegt die Datenbank bereits sortiert vor, so genügen 10 Millionen Vergleiche, also blitzschnelle **10 Sekunden**. (3) Um die Datenbank effizient zu sortieren, können wir verschiedene Algorithmen verwenden. Wir entscheiden uns hier für Mergesort (siehe `de.wikipedia.org/wiki/Mergesort` oder animiert `y2u.be/es2T6KY45cA`). Dieser Sortieralgorithmus löst unser Problem mit höchstens 240 Millionen Vergleichen. Nach spätestens 240 Sekunden, also nur **4 Minuten**, ist unsere Datenbank sortiert! Andere Verfahren liefern ähnliche Werte; im Allgemeinen geht es nachweislich nicht schneller als 218 Sekunden.

Stufe 1 / Ausführung: *Wenn Sie es genauer wissen wollen, führen wir die Rechnung für Sie aus.*

(1) Wie lange benötigt der Computer, um alle Paare zu vergleichen? Dazu zählen wir zunächst alle Paare. Wir haben $n = 10^7$ Einträge in unserer Datenbank. Eintrag 1 vergleichen wir mit den Einträgen $2, 3, \ldots, n$, das sind insgesamt $V_1 = 10^7 - 1$ Vergleiche. Eintrag 2 vergleichen wir mit den Einträgen $3, 4, \ldots, n$, das sind insgesamt $V_2 = 10^7 - 2$ Vergleiche. So können wir fortfahren: Eintrag k vergleichen wir mit allen folgenden Einträgen $(k+1), (k+2), \ldots, n$, dies sind insgesamt $V_k = 10^7 - k$ Vergleiche. Die Gesamtzahl aller paarweisen Vergleiche ist somit

$$V = \sum_{k=1}^{10^7-1} V_k = 9\,999\,999 + 9\,999\,998 + \cdots + 2 + 1.$$

Zur Berechnung dieser Summe gibt es einen genial-einfachen Trick, den Carl Friedrich Gauß schon als neunjähriger Schüler erkannte. Die folgende Summenformel heißt daher auch **der kleine Gauß**:

$$\begin{array}{rcrcrcrcrcrcrcr}
2V & = & 10^7{-}1 & + & 10^7{-}2 & + & 10^7{-}3 & + & \ldots & + & 3 & + & 2 & + & 1 \\
 & + & 1 & + & 2 & + & 3 & + & \ldots & + & 10^7{-}3 & + & 10^7{-}2 & + & 10^7{-}1 \\
\hline
 & = & 10^7 & + & 10^7 & + & 10^7 & + & \ldots & + & 10^7 & + & 10^7 & + & 10^7
\end{array}$$

Somit gilt $2V = 10^7(10^7 - 1)$, also aufgelöst

$$V = \frac{1}{2} \cdot 10^7 \cdot (10^7 - 1) = 49\,999\,995\,000\,000.$$

Alternative Betrachtung: Wie viele Vergleiche benötigen wir, wenn wir jeden Eintrag i mit *allen* anderen Einträgen $j \neq i$ vergleichen? Dies sind genau $10^7 \cdot (10^7 - 1)$ Vergleiche. Dabei behandeln wir jedoch jedes Paar $\{i, j\}$ doppelt, da wir einmal i mit j und einmal j mit i vergleichen. Somit ist die korrekte Anzahl der paarweisen Vergleiche $V = \frac{1}{2} \cdot 10^7 \cdot (10^7 - 1)$, wie oben angegeben.

Unser Computer erledigt einen Vergleich pro Mikrosekunde (10^{-6} Sekunden), also

$$\begin{aligned}
49\,999\,995\,000\,000 \cdot 10^{-6} \text{ Sek.} &= 49\,999\,995 \text{ Sek.} \\
&= 833\,333 \text{ Min. und } 15 \text{ Sek.} \\
&= 13\,888 \text{ Stunden, } 53 \text{ Min. und } 15 \text{ Sek.} \\
&= 578 \text{ Tage, } 16 \text{ Stunden, } 53 \text{ Min. und } 15 \text{ Sek.} \\
&= 1 \text{ Jahr, } 213 \text{ Tage, } 16 \text{ Stunden, } 53 \text{ Min. und } 15 \text{ Sek.}
\end{aligned}$$

Die Rechenzeit mit diesem (allzu naiven) Verfahren ist also **1 bis 3 Jahre**. Ihr Praktikum dauert sicher nicht lange genug, um das Ergebnis abzuwarten. Für Sie (und Ihre Chefin) lohnt es sich, dieses Problem rechtzeitig zu erkennen! Sie sollten Ihre Zeit besser investieren und sich ein effizienteres Verfahren überlegen. Denken hilft: Die Geschwindigkeit des Computers hängt nicht nur an der Hardware, sondern auch und vor allem an der Intelligenz / Ausbildung / Erfahrung des Benutzers!

(2) Wie lange dauert die Suche nach Dopplungen, wenn die Datenbank sortiert vorliegt?

Angenommen, die Datenbank ist bereits nach Kundennummern aufsteigend sortiert. Dies erleichtert uns die Suche nach Dopplungen enorm: Wir müssen lediglich jeden Eintrag $k = 1, \ldots, 9\,999\,999$ mit seinem Nachfolger $(k+1)$ vergleichen. Für diese $V = 9\,999\,999$ Vergleiche benötigen wir

$$9\,999\,999 \cdot 10^{-6} \text{ Sek.} \approx 10 \text{ Sek.}$$

Als Zeitintervall aus der Aufgabenstellung sind dies **5 bis 20 Sekunden**.

(3) Wie lange benötigt der Computer zum Sortieren? Zunächst müssen Sie entscheiden, mit welchem Algorithmus die Datenbank sortiert werden soll. Hierzu gibt es vergleichsbasierte Verfahren, die selbst in den ungünstigsten Fällen recht schnell sind und keinen zusätzlichen Speicherbedarf erzeugen. Egal, welchen dieser schnellen Algorithmen Sie verwenden, das Sortieren dauert in etwa gleich lang und benötigt etwa $n\lceil\log_2(n)\rceil$ Vergleiche; dies beweisen wir anschließend in Stufe 3. Notation: Für jede reelle Zahl $x \in \mathbb{R}$ bedeutet $\lfloor x \rfloor$ Abrunden und $\lceil x \rceil$ Aufrunden zur nächsten ganzen Zahl. Konkrete Anwendung: In unserem Fall ist $n = 10^7$, also $\ell = \lceil \log_2(10^7) \rceil = \lceil 23,25 \rceil = 24$. Somit genügen 240 Millionen Vergleiche, also maximal 240 Sekunden oder 4 Minuten. Genauer benötigt Mergesort höchstens $c(n) = n \cdot \ell - 2^\ell + 1$ Vergleiche, also 3 Minuten 44 Sekunden. Andere Sortieralgorithmen liefern ähnliche Werte. Die untere Schranke sind $\log_2(n!)$ Vergleiche, also 3 Minuten 38 Sekunden. In jedem Falle ist die Antwort **3 bis 5 Minuten** richtig.

Stufe 2 / Was will und soll diese Aufgabe?

Nach der Lösung dieser Aufgabe erläutern wir als Rück- und Ausblick, warum wir diese Problemstellung mathematisch interessant finden und inwiefern sie repräsentativ ist für das Mathematikstudium.

In der Mathematik geht es oft darum, eine Aussage zu beweisen oder eine Methode zu finden, die das vorgelegte Problem löst. Im Mathematikstudium erlernen Sie dazu viele nützliche Techniken. Leider bringt dieser Erfahrungsschatz wenig, wenn Sie sich naiv für die erstbeste Methode entscheiden und diese benötigt mehr Zeit, als Sie zur Verfügung haben, etwa weil Ihr Praktikum vor Ablauf eines Jahres endet. Genauso wäre es, wenn Sie eine Liste mit 10 Millionen Datensätzen ungeschickt sortieren, etwa mit quadratischem Aufwand. Nicht nur von Hand dauert das ewig, selbst ein Computer braucht dafür mehr als ein Jahr, wie oben ausgerechnet. Hätten Sie das gedacht?

> *In fact, there were many installations in which the task of sorting was responsible for more than half of the computing time. From these statistics we may conclude that either (i) there are many important applications of sorting, or (ii) many people sort when they shouldn't, or (iii) inefficient sorting algorithms have been in common use. The real truth probably involves all three of these possibilities, but in any event we can see that sorting is worthy of serious study, as a practical matter.* (Donald E. Knuth, *The Art of Computer Programming*)

Dies illustriert eine wichtige Grundregel: Zur praktischen Umsetzung Ihrer Kenntnisse müssen Sie die möglichen Methoden gründlich verstehen und für die vorliegende Anwendung eine geeignete auswählen. Manche Probleme lösen Sie mit Stift und Papier, meist ist zudem der Computer ein willkommenes Hilfsmittel, doch Sie müssen die Möglichkeiten erkennen und nutzen. Für jede ernsthafte Anwendung ist es unerlässlich, systematisch nach effizienten Lösungen zu suchen. Hierzu bietet diese Aufgabe ein repräsentatives Beispiel, mit realistischen Zahlen und anschaulicher Interpretation.

Ausblick: Algorithmen und Datenstrukturen. Im Mathematikstudium können Sie unter vielen interessanten und nützlichen Nebenfächern auswählen. Unser Beispiel Suchen und Sortieren ist ein grundlegendes Thema der Informatik, allgemeiner der Algorithmen und Datenstrukturen. Es gibt zahlreiche Sortieralgorithmen; sie unterscheiden sich nicht nur im benötigten Zeitaufwand, schlimmstenfalls oder durchschnittlich, sondern zum Beispiel auch darin, ob sie neben paarweisen Vergleichen

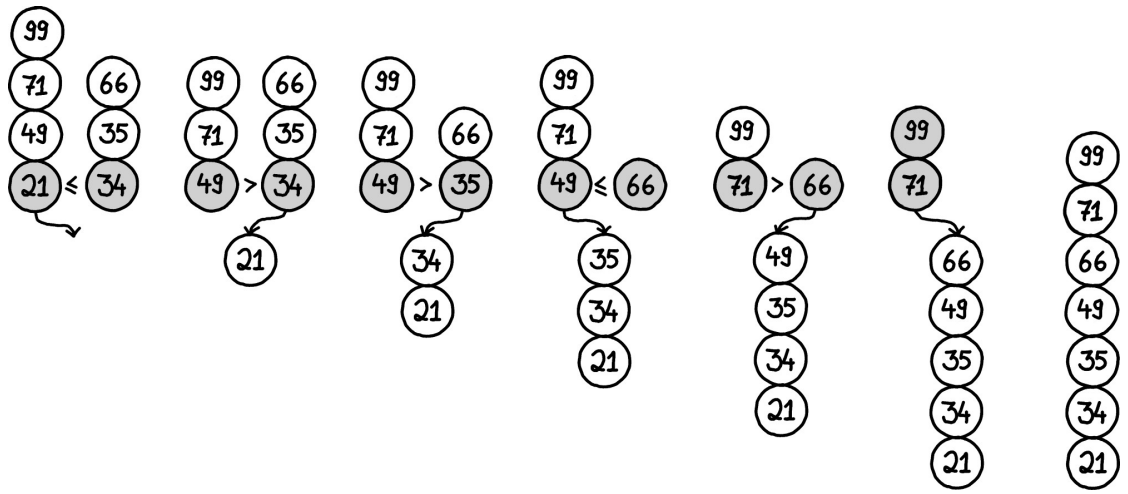
weitere Informationen verwenden, wie viel Speicherplatz sie benötigen, ob sie mit vorsortierten Listen gut umgehen oder ob sie stabil sind: Ist die Liste schon nach einem ersten Kriterium sortiert, etwa der Postleitzahl, so soll dies bei Sortierung nach weiteren Kriterien erhalten bleiben. Das hat wichtige Anwendungen! Donald E. Knuth widmet den gesamten dritten Band seines Lehrbuchs *The Art of Computer Programming* allein dem Suchen und Sortieren.

Anwendung im Beruf. Kann man mit Mathematik Geld verdienen oder ist sie eine brotlose Kunst? Tatsächlich sind die Berufsaussichten für MathematikerInnen sehr gut: Überall, wo logisches Denken und systematisches Problemlösen gefragt sind, werden MathematikerInnen gesucht.

Ein großes Arbeitsgebiet für MathematikerInnen ist neben Forschung und Entwicklung, Softwareunternehmen und Unternehmensberatungen auch die Versicherungsbranche. Die Produkte einer Versicherung müssen so konzipiert sein, dass die Versicherung Gewinn macht, aber auch nicht allzu teuer ist. Vor der Einführung wird daher jedes Versicherungsprodukt genauestens überprüft, und zwar großteils von MathematikerInnen. Es ist also nicht unwahrscheinlich, dass Sie ein Praktikum bei einer Versicherung machen. (Wenn Sie sich für ein solches Praktikum entscheiden, kann Ihnen dies im Masterstudium angerechnet werden.) Sicher werden Sie dabei auch programmieren, das gehört zum alltäglichen Handwerk und wird daher im Mathematikstudium in den ersten Semestern im Modul *Mathematische Programmierung* und später im *Computerpraktikum* erlernt und geübt. Eine Datenbank zu sortieren ist im Beruf eher eine Fingerübung zum Einstieg; bei solchen Fragen nützen Ihnen sichere theoretische Grundlagen und praktische Übung. Das zahlt sich insbesondere aus, wenn die Fragestellung kniffliger wird, und Sie nicht nach Schema F vorgehen können, sondern eigenständig die bekannten Verfahren auf neue Situationen anpassen müssen. Für Ihre Problemlösungen mit Sorgfalt und Kreativität werden Sie als MathematikerIn gut bezahlt. Solche Praktika und scheinbar simple Probleme sind durchaus real, eine solche Geschichte wird schön erzählt in `y2u.be/RGuJga2Gl_k`.

Stufe 3 / Komplexität, obere und untere Schranken. *Wenn Sie bis hierher gelesen haben, dann möchten Sie vielleicht einen Vorgeschmack bekommen, was Sie im Studium erwartet. Wir erklären zunächst einen schnellen Sortieralgorithmus und zeigen dann, dass es nicht wesentlich schneller gehen kann. Wenn Sie die Argumente mit Geduld und Neugier sowie Stift und Papier durchgehen, können Sie ungefähr einschätzen, ob diese Art mathematischer Arbeit Ihnen Freude bereitet.*

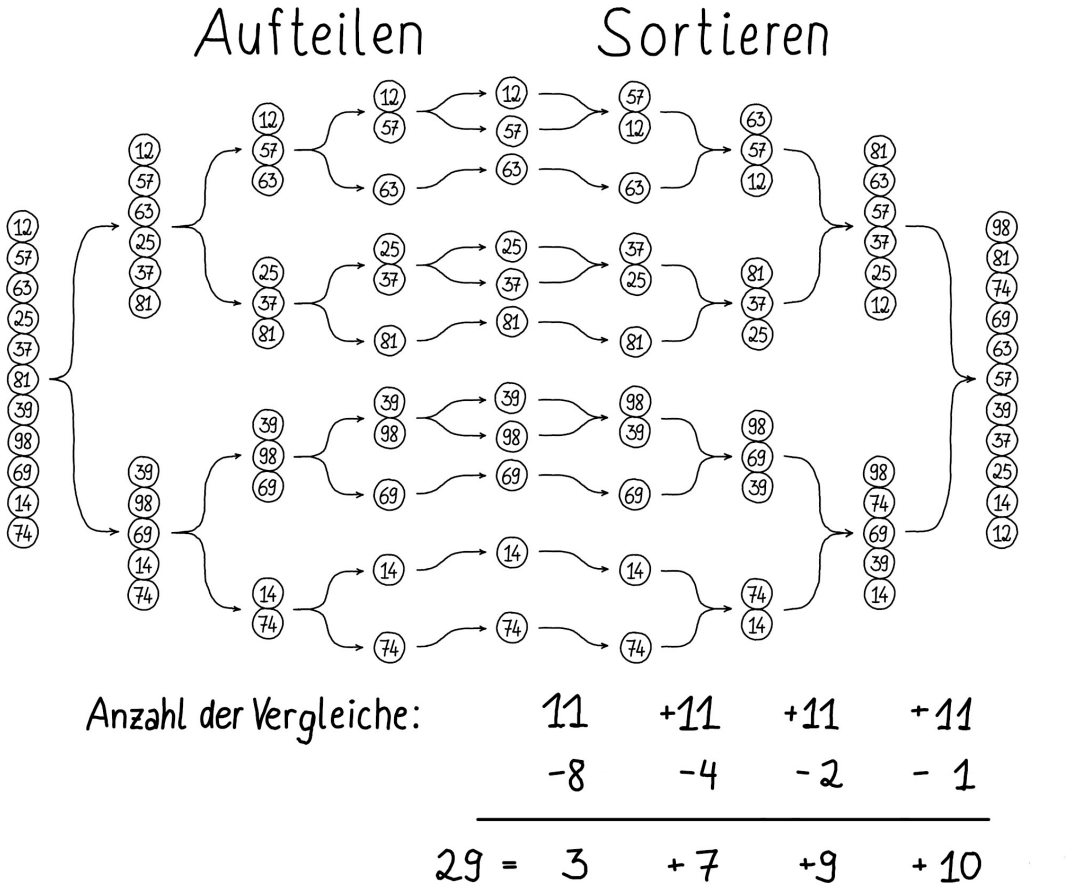
Da **Mergesort** relativ einfach und intuitiv ist, verwenden wir diesen Algorithmus: Zu sortieren ist die Liste $(A_1, A_2, \ldots, A_n)$. Im Falle $n \leq 1$ ist nichts zu tun. Im Falle $n \geq 2$ teilen wir unsere Liste mittig in zwei Listen $L = (A_1, \ldots, A_p)$ und $R = (A_{p+1}, \ldots, A_n)$ der Länge $p = \lfloor n/2 \rfloor$ und $q = \lceil n/2 \rceil$. Beide Hälften werden getrennt sortiert, und zwar rekursiv erneut mit Mergesort, und dann im Reißverschlussverfahren zusammengefügt (animiert in `y2u.be/es2T6KY45cA`).



Das Reißverschlussverfahren benötigt höchstens $p + q - 1$ Vergleiche.

Sei $c(n)$ die Anzahl der mit diesem Verfahren schlimmstenfalls ausgeführten Vergleiche. Hierzu gilt die obere Schranke $c(n) \leq n\lceil \log_2(n) \rceil$; dies wollen wir nun beweisen.

Erster Beweis: Turnier. Wie viele Vergleiche benötigen wir, um n Listeneinträge zu sortieren? Wir setzen $\ell := \lceil \log_2(n) \rceil$, sodass $2^{\ell-1} < n \leq 2^\ell$ gilt. Die Listen werden ähnlich wie bei einem Turnier in ℓ Runden vereinigt. Insgesamt finden $2^\ell - 1$ solche Vereinigungen statt, in der ersten Runde eventuell mit Listen der Länge 0. Um zwei Listen der Länge p und q zusammenzusortieren, benötigt man schlimmstenfalls $p + q - 1$ Vergleiche, also immer einen weniger als die Gesamtzahl der Listenelemente. In jeder Runde werden insgesamt n Elemente zusammensortiert. Das ergibt insgesamt $c(n) = n \cdot \ell - 2^\ell + 1$. Damit verstehen Sie insbesondere, wie diese Formel entsteht!



Sortieren als Turnier

Zweiter Beweis: Induktion. Wir wollen $c(n)$ berechnen. Wir wissen $c(0) = c(1) = 0$ und für $n \geq 2$ rekursiv $c(n) = c(\lfloor n/2 \rfloor) + c(\lceil n/2 \rceil) + n - 1$. Wir beweisen nun $c(n) = n \cdot \ell - 2^\ell + 1$ per Induktion über $n = 0, 1, 2, 3, \ldots$: Die Gleichung gilt für $n = 0$ und $n = 1$. Im Folgenden sei $n \geq 2$. Angenommen die Gleichung gilt für $c(\lfloor n/2 \rfloor)$ und $c(\lceil n/2 \rceil)$. Dann gilt sie auch für $c(n)$, denn:

$$\begin{aligned} c(n) &= c(\lfloor n/2 \rfloor) && + c(\lceil n/2 \rceil) && + n - 1 \\ &= \lfloor n/2 \rfloor(\ell - 1) - 2^{\ell-1} + 1 && + \lceil n/2 \rceil(\ell - 1) - 2^{\ell-1} + 1 && + n - 1 \\ &= n(\ell - 1) - 2^\ell + n + 1 \\ &= n\ell - 2^\ell + 1 \end{aligned}$$

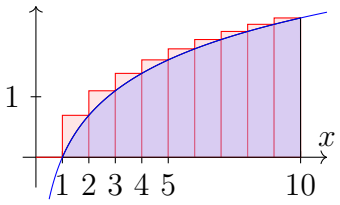
Bemerkung: Der erste Beweis durch Abzählen in einem Turnier ist kreativ und raffiniert. Der zweite Beweis nutzt die vollständige Induktion als routiniertes Handwerk. Induktion ist eine vielseitige Beweismethode und oft als konkrete Rechenmethode nutzbar, so wie hier. Wir bieten Ihnen beide Sichtweisen an: Welche finden Sie einfacher? lehrreicher? eleganter?

Untere Schranke. Wer sagt uns eigentlich, dass es nicht schneller geht? Mergesort benötigt höchstens $n\lceil\log_2(n)\rceil$ Vergleiche. Die besten bekannten vergleichsbasierten Sortieralgorithmen erfüllen alle dieselbe obere Schranke. Geht es noch besser? Können wir oder irgendjemand sonst einen noch besseren Sortieralgorithmus finden? Wir können dies mit einer unteren Schranke beantworten!

Schlimmstenfalls sind alle n Kundennummern verschieden, dann gibt es vor dem Sortieren $n!$ mögliche Anordnungen: Die erste Kundennummer kann an n verschiedenen Positionen sein, die zweite an den verbleibenden $n-1$ Positionen, usw. Das macht insgesamt $n! = n(n-1)(n-2)\cdots 2\cdot 1$ verschiedene Anfangszustände, die zu unterscheiden sind. Mit einem Vergleich kann man höchstens 2 Zustände unterscheiden, mit zwei Vergleichen höchstens 4 Zustände, mit v Vergleichen höchstens 2^v Zustände. Um also $n!$ Zustände zu unterscheiden, benötigen wir $v \geq \lceil\log_2(n!)\rceil$ Vergleiche.

Diese Zahl scheint recht groß. Wie groß genau? Rechnen wir es aus!

$$\begin{aligned}\ln(n!) &= \ln(1\cdot 2\cdot 3\cdots(n-1)\cdot n)\\ &= \ln(1)+\ln(2)+\ln(3)+\cdots+\ln(n-1)+\ln(n)\\ &= \sum_{k=1}^{n}\ln(k) \geq \int_{x=1}^{n}\ln(x)\,\mathrm{d}x = \Big[x\ln(x)-x\Big]_{x=1}^{n} \geq n\big[\ln(n)-1\big]\end{aligned}$$

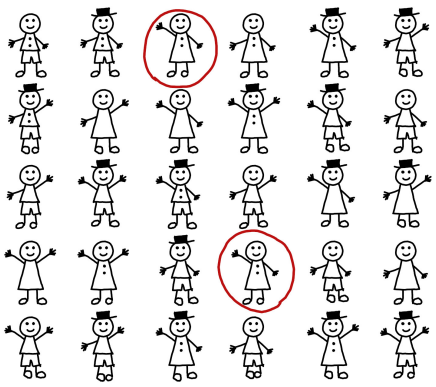


Bemerkung: (a) Die Ungleichung zwischen Summe und Integral sehen Sie, indem Sie über die Fläche des Funktionsgraphen (blau) die Summe als Balkendiagramm (rot) einzeichnen. (b) Die Stammfunktion von $\ln(x)$ können Sie durch partielle Integration finden und durch Ableiten nachprüfen.

Für $n = 10^7$ Einträge benötigen wir also im Allgemeinen mindestens

$$\log_2(n!) = \frac{\ln(n!)}{\ln(2)} \geq \frac{n\big[\ln(n)-1\big]}{\ln(2)} \approx 218\cdot 10^6$$

Vergleiche. Selbst der schnellste Algorithmus, egal ob bereits bekannt oder aktuell noch unbekannt, wird im Allgemeinen nicht schneller als 3 Minuten 38 Sekunden sein können. Wenn sich Ihre Chefin eine noch schnellere Lösung wünscht, dann können Sie ihr sachlich erklären: Die algorithmischen Möglichkeiten sind sorgfältig ausgeschöpft, ab jetzt hilft tatsächlich nur noch schnellere Hardware.



Wo ist das Doppelgängerpaar?

Und wie findet man im Suchbild die Doppelgänger?

Jede Figur mit jeder zu vergleichen, ist hier noch machbar, aber lästig. Wenn es mehr Figuren werden, wird es sehr langwierig.

Schneller geht es mit dem folgenden Verfahren: Man teilt die Figuren nach einem Merkmal auf, z.B. alle ohne Hut und alle mit Hut. In jeder der beiden Gruppen sucht man den Doppelgänger, indem man mit einem weiteren Merkmal ebenso verfährt, z.B. indem man in beiden Gruppen die Figuren mit Knöpfen und die ohne Knöpfe betrachtet. Teilt man die Gruppen immer weiter auf, so bleiben am Ende Gruppen mit einer Figur oder Gruppen aus Doppelgängern.

Auf dem Computer können wir dies implementieren durch einen „Fingerabdruck“ für jedes Bild: In unserem Beispiel gibt es sieben Merkmale: Kleid/Hose, mit/ohne Hut, mit/ohne Knöpfe, linke Hand, rechte Hand, linker Fuß, rechter Fuß. Dies liefert einen siebenstelligen Code, hier eine Binärzahl. Nach diesem können wir nun sortieren und so Doppelgänger finden. Dieses Verfahren, noch weiter verfeinert, wird von Google image search verwendet, um Fotos im Internet zu finden.