Version vom
2. März 2020

12.

Rekursives Problemlösen

Buch Mark Weiss „Data Structures & Problem Solving Using Java“ siehe:
- 327-363 (rekursives Problemlösen)
- 393-396 (Mergesort)

Lernziele Kapitel 12 Rekursives Problemlösen

- Prinzip der Rekursion und rekursiver Methode verstehen
- Prinzip des „divide et impera“ anwenden können
- Mergesort in Bottom-up- und Top-down-Varianten beherrschen
- Das Prinzip von Quicksort verstehen

Thema / Inhalt

Hier geht es um **Rekursion**, im engeren Sinne um rekursives Problemlösen – also der Einsatz von Rekursion als Technik, um in algorithmischer Weise ein Berechnungsproblem zu lösen. Wir betrachten zunächst einige Kurven (u.a. die **Peano-Kurve** und die **Hilbert-Kurve)**, die entsprechend eines rekursiven Bildungsgesetzes aufgebaut sind und einen intuitiven Zugang zum Prinzip der Rekursion liefern (gleichzeitig aber auch in mathematischer Hinsicht interessant sind, weil sie als raumfüllende Kurve die Frage nach dem Wesen einer Kurve und dem Unterschied zwischen einer und zwei Dimensionen evozieren).

Anschliessend diskutieren wir das „**Divide-et-impera-Paradigma**“, bei dem ein „grosses“ oder „schwieriges“ Problem („top-down“) so lange rekursiv in kleinere und einfachere Teilprobleme zerlegt wird, bis diese gelöst („beherrschbar“) sind – meist, weil dann ein trivialer Fall vorliegt (wie z.B. das Minimum einer einelementigen Menge bestimmen). Anschliessend wird aus diesen Teillösungen schrittweise („bottom-up“) eine Lösung für das Gesamtproblem

Thema / Inhalt (2)

rekonstruiert. Die rekursive Struktur von Divide-et-impera-Verfahren induziert, dass deren Korrektheit typischerweise durch Induktion gezeigt werden kann.

Das Paradigma kommt bei vielen Such- und Sortierverfahren zur Anwendung, u.a. bei Mergesort, das ebenfalls in diesem Kapitel besprochen wird. Im Prinzip könnte man sogar die altägyptische Multiplikation, den euklidischen Algorithmus oder die Binärsuche als Verkörperungen dieses Paradigmas auffassen; weil aber das Problem dabei nur auf eine einzige (einfachere) Instanz des gleichen Problems zurückgeführt wird, und nicht tatsächlich eine Aufteilung erfolgt, sieht man meistens davon ab. Hingegen stellen die im vorherigen Kapitel besprochen Spielbaumanalyseverfahren (wie Minimax) oder die Auswertung von Operatorbäumen Anwendungen des Paradigmas dar, ebenso wie die für die Praxis wichtigen Prinzipien der schnellen Fourier-Trans-formation (FFT) oder der schnellen Multiplikation mit dem Schönhage-Strassen-Algorithmus – die letzteren beiden Themen sind allerdings nicht Gegenstand dieser Vorlesung.

Beim Lehrbuchklassiker des rekursiven Problemlösens, den **Türmen von Hanoi**, diskutieren wir nicht nur den Lösungsansatz, sondern nutzen das Beispiel auch dafür, die Zeitkomplexität der rekursiven Lösung sowie einige generelle Begriffe und Konzepte des rekursiven Problemlösens (wie Rekursionsbaum oder dynamische Aufrufkette) zu thematisieren.

Beim **Mergesort**-Verfahren besprechen wir zwei Hauptvarianten: Einerseits die rekursive Top-down-Variante, andererseits die Bottom-up-Variante, bei der sukzessive längere sortierte Teilfolgen miteinander verschmolzen werden. Beide Varianten haben eine Laufzeitkomplexität von O(n log n), unabhängig von der konkreten Verteilung der zu sortierenden Werte. Mergesort wurde 1945 von John von Neumann gewissermassen nebenbei erfunden – wobei das Prinzip wohl schon früher beim Sortieren von Gegenständen (Karteikarten etc.) der physischen Welt angewendet wurde. Die Notizen von John von Neumann lassen erkennen, wie mühsam das

Thema / Inhalt (3)

Programmieren auf Maschinensprache-Niveau seinerzeit war – aber es handelte sich hier ja auch um eines der ersten Computerprogramme überhaupt! Dass von Neumann als Mathematiker ein nicht-numerisches Problem für seine erste „Programmierübung" wählte, ist auch bemerkenswert.

Neben Mergesort gehen wir noch kurz auf Quicksort ein – erfunden Anfang der 1960er-Jahre von Tony Hoare, der uns auch an anderer Stelle der Vorlesung schon begegnet ist (Hoare-Kalkül zur Verifikation von Programmen). Quicksort lässt sich in eine schöne strukturelle Analogie zu Mergesort setzen: Beide verwenden das Divide-et-impera-Paradigma, statt einer nachgelagerten Merge-Phase nutzt Quicksort hingegen eine vorgelagerte Partition-Phase.

Dive and conquer

Divide and conquer

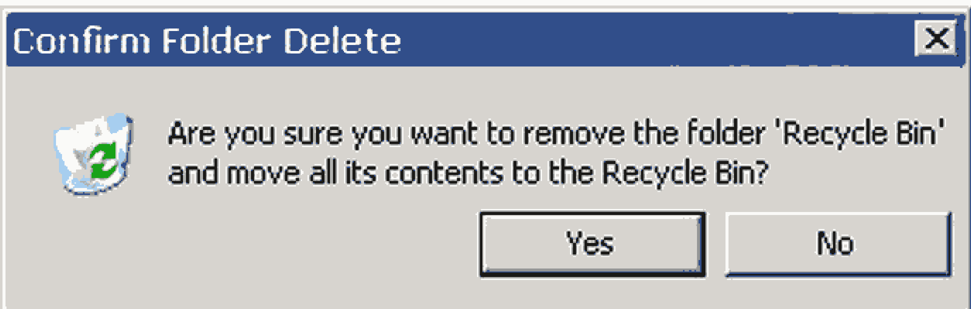
Rekursion

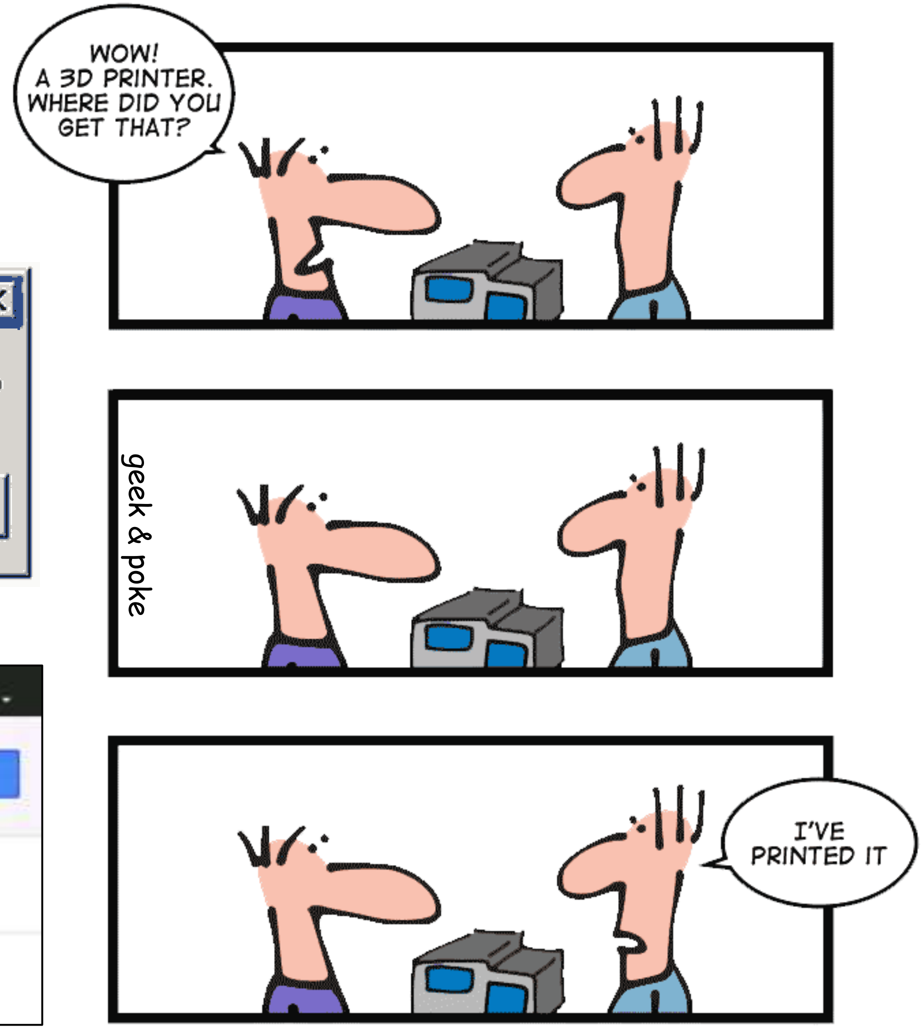
Rekursive Strukturen und Verfahren sind uns in der Vorlesung (auch Teil I) schon mehrfach begegnet

Die notorische Eigenheit der deutschen Sprache, das Verbum ans Ende des Satzes zu stellen, über welche lustige Geschichten von geistesabwesenden Professoren, die einen Satz beginnen, die ganze Vorlesung lang weiterreden, und damit aufhören, dass sie eine Kette von Verben herunterleiern, wobei die Zuhörer, für die die einzelnen Satzbruchteile schon längst jeglichen Zusammenhang verloren haben, völlig verwirrt werden, erzählt werden, ist ein sehr gutes Beispiel für linguistische Rekursion. -- *Douglas Hofstadter*

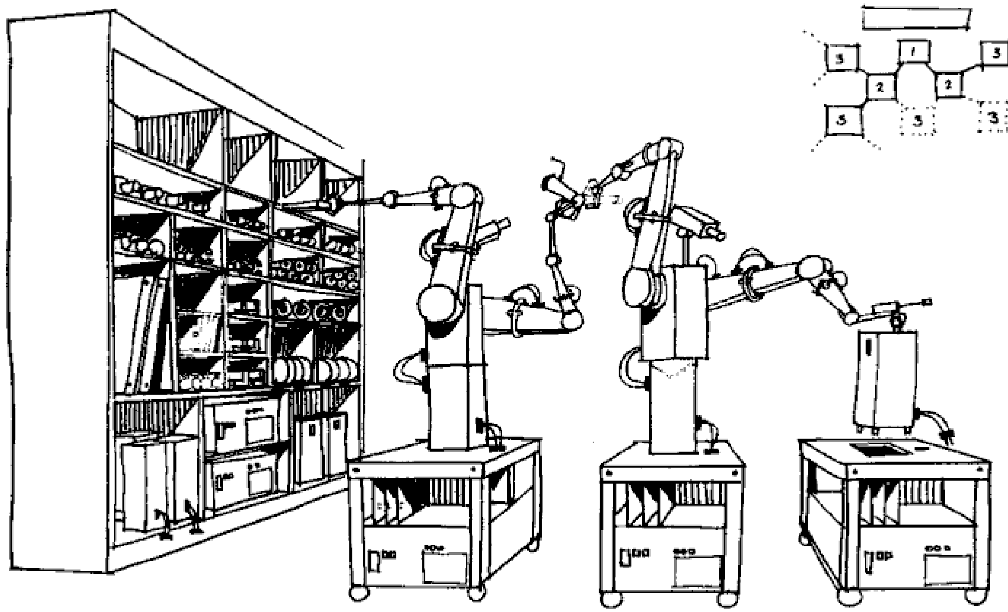
Rekursion

Ein Relativsatz, der einen Relativsatz, der einen Relativsatz enthält, enthält.





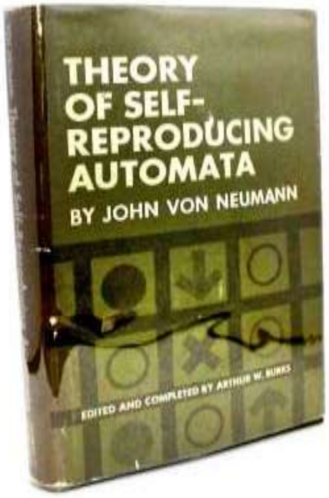
Rekursion → Selbstreplikation (John von Neumann)



https://en.wikipedia.org/wiki/File:Advanced_Automation_for_Space_Missions_figure_5-29.gif

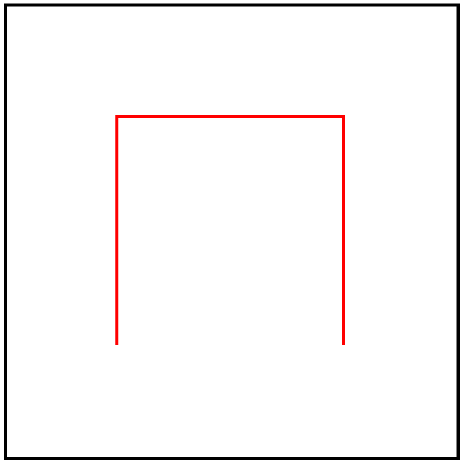
A NASA-funded artistic rendition of Neumann's automaton that produce automaton, the original caption of which is: "proposed demonstration of simple robot self-replication" (1980).

The so-called "Neumann automaton theory" originated in a 20 Sep. 1948 Hixton Symposium lecture, Pasadena, Ca., organized by American chemical engineer Linus Pauling, given by Hungarian-born American chemical engineer and mathematician John von Neumann, during the course of which Neumann invented a famous thought experiment which illustrates the role which free energy plays in creating statistically unlikely configurations of matter. Neumann imagined a robot or automaton, made of wires, electrical motors, batteries, etc., constructed in such a way that when floating on a lake stoked with component parts, it will reproduce itself. Neumann, in his lecture, first compares computers to biological information processing systems then suggests a program to deal with "automata that produce automata", the gist of which is captured in the following statement: "Can one build an aggregate out of such elements in such a manner that if it is put into a reservoir, in which there float all these elements in large numbers, it will then begin to construct other aggregates, each of which will then at the end turn out to be another automaton exactly like the original ones?"
[www.eoht.info/page/Neumann+automaton+theory]

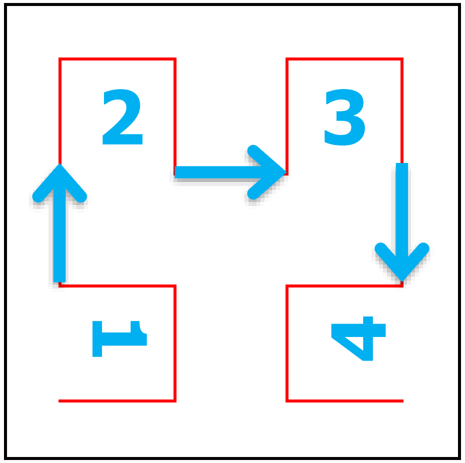


Bsp: Hilbert-Kurve als rekursives Muster

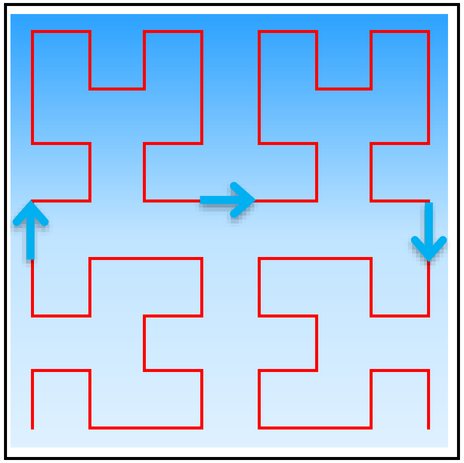
hilbert(1)

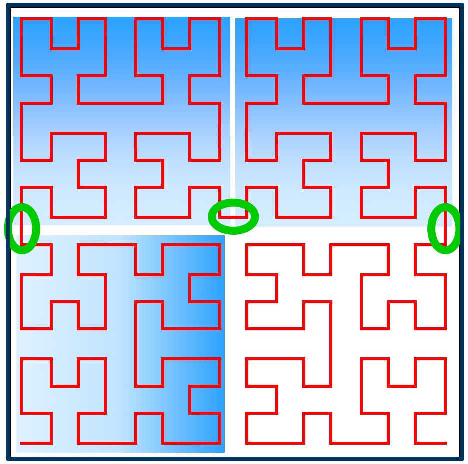


hilbert(2)

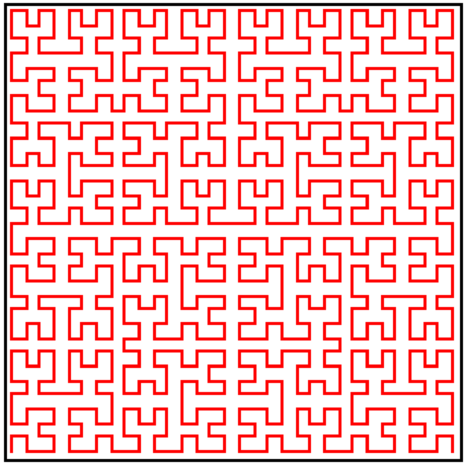


hilbert(3)

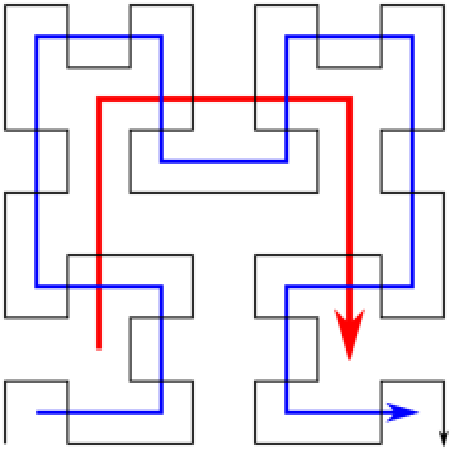


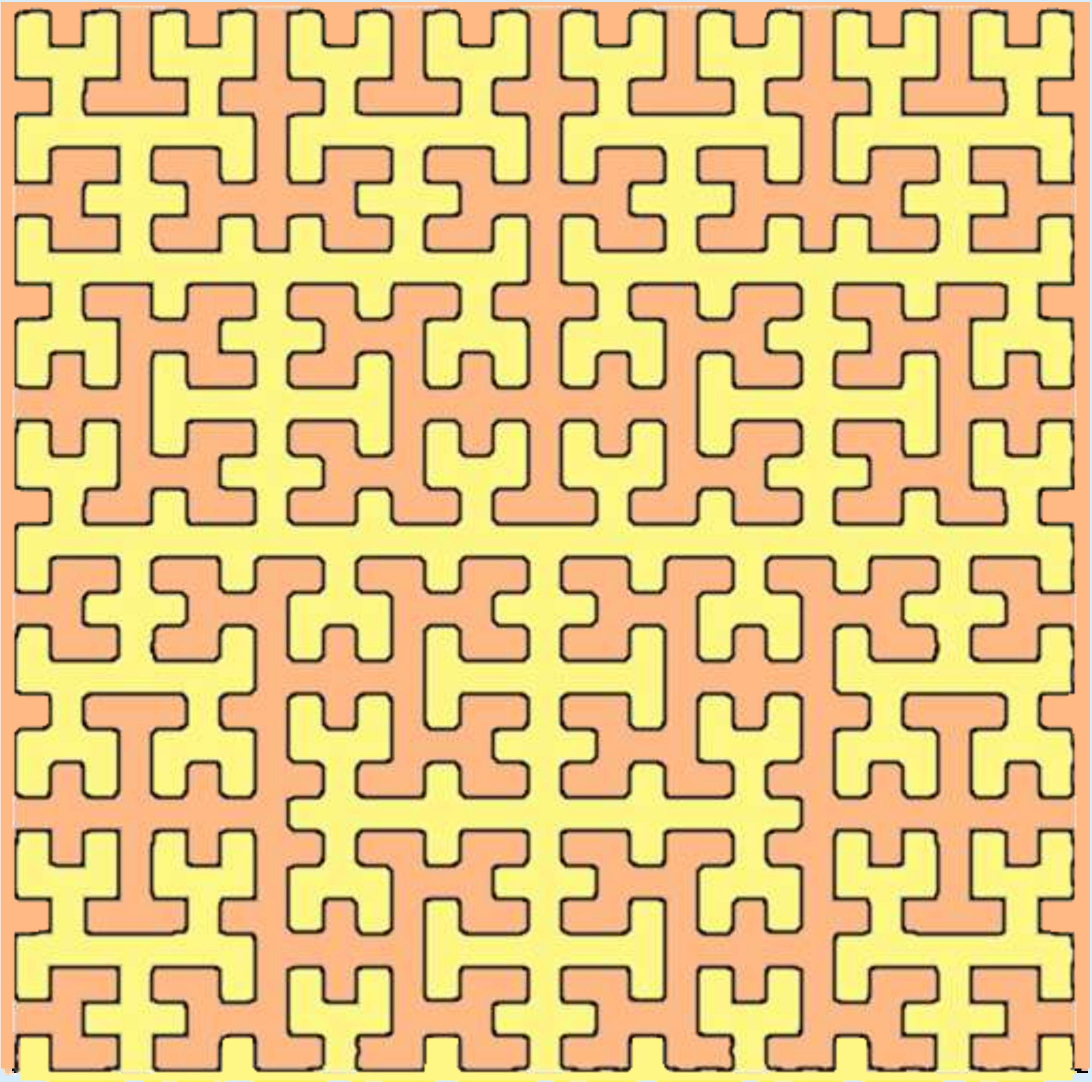


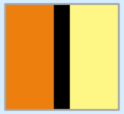
hilbert(4)



hilbert(5)



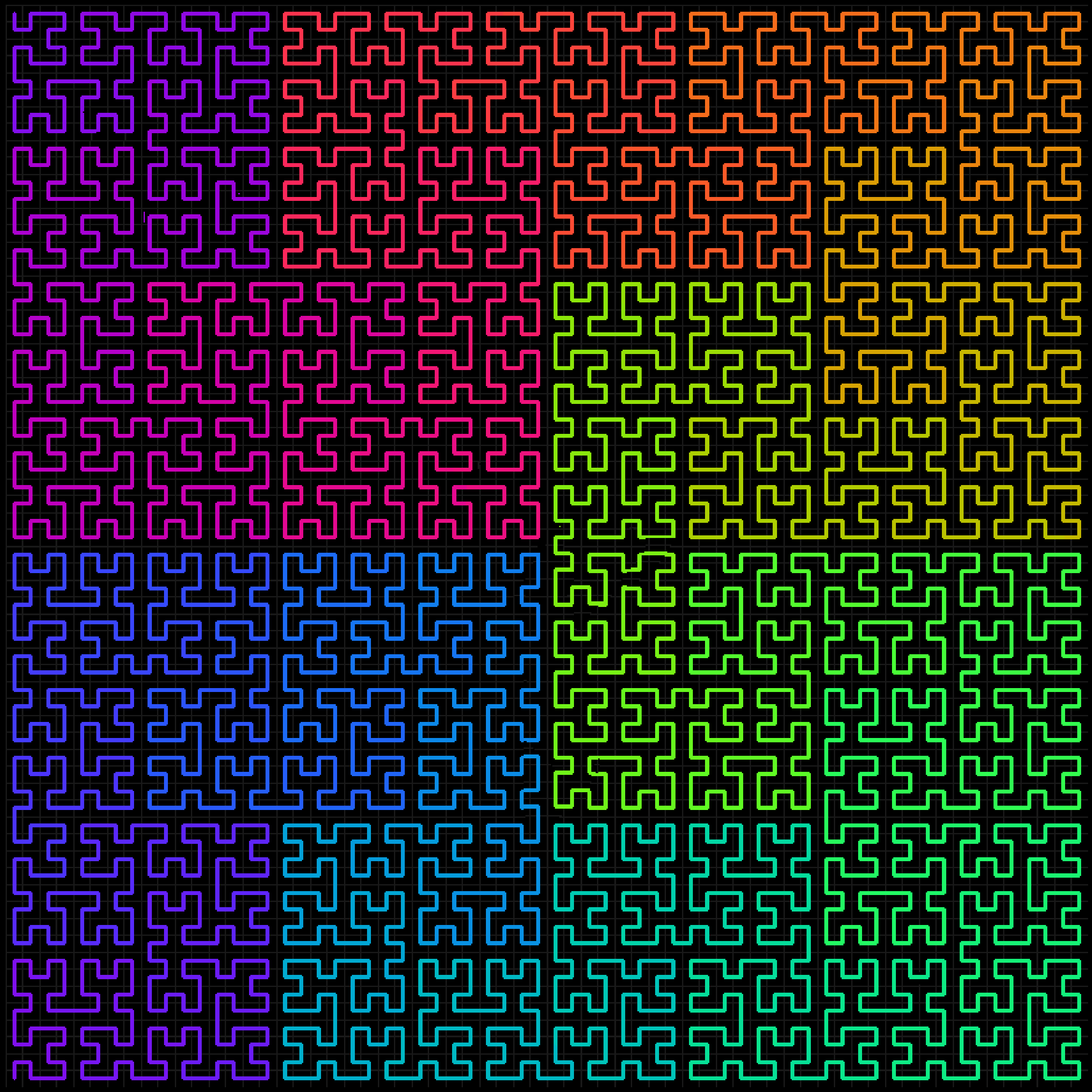




Kurve trennt die Fläche in zwei Teile → Beim Durchlaufen links orange, rechts gelb färben

So, wie die Seine Paris in 2 Teile teilt: RG / RD

Stetiger Farbverlauf der Kurve mit 1024 Ecken



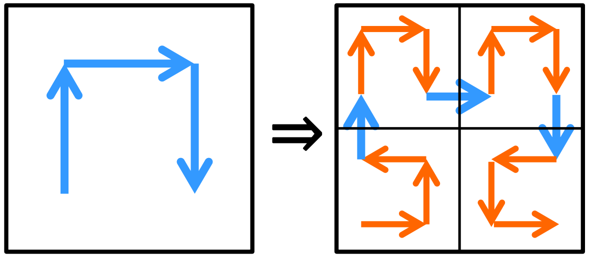
https://i.redd.it/9pslajy95tbz.gif

Lokal verschieden dichte Hilbert-Kurve

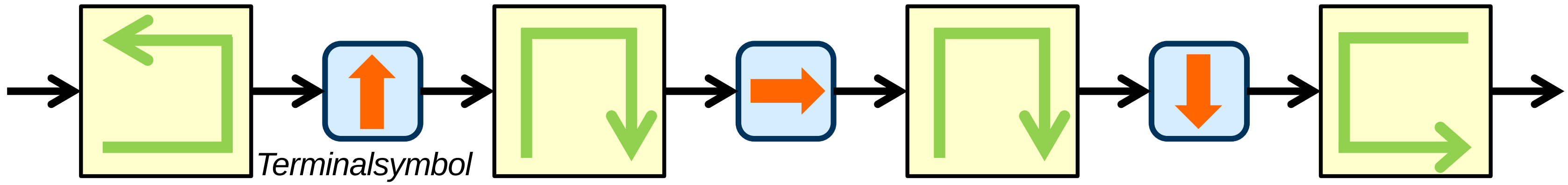
Anwendung z.B. bei der Bild-
codierung und -komprimierung

Syntaxdiagramme für Hilbert-Kurven

- Die Hilbert-Kurve hat eine rekursive Struktur
  - Das Grundmuster der Stufe i tritt 4 Mal auf der Stufe i+1 auf (z.T. gedreht / gespiegelt)
  - Die vier Teile sind durch Linien, die selbst das Grundmuster reflektieren, verbunden



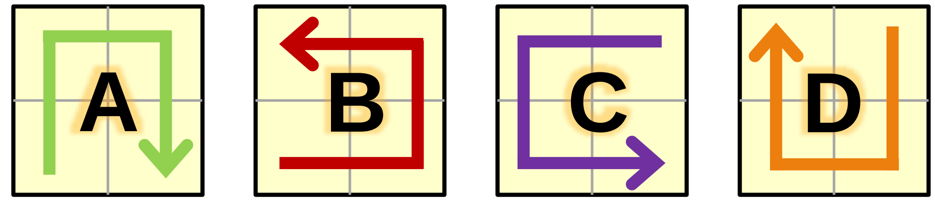
- Das rekursive Bildungsgesetz kann mit Syntaxdiagrammen (also einer „Grammatik") ausgedrückt werden; zum Nicht-Terminalsymbol  z.B. gehört folgendes Diagramm:



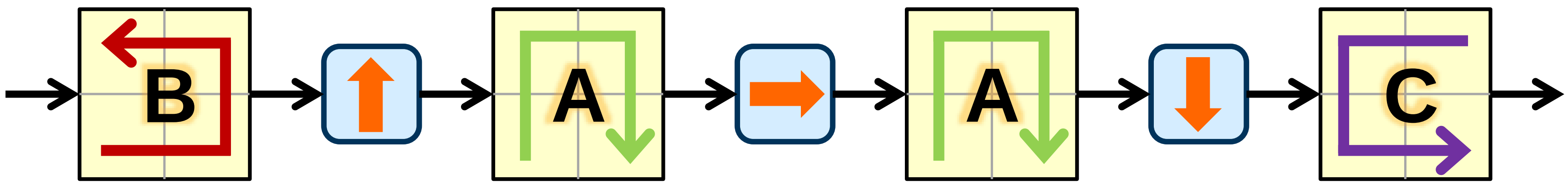
- Man benötigt noch Möglichkeiten, aus der Rekursion auszustei-gen und analoge Diagramme für die anderen 3 Nicht-Terminale

Syntaxdiagramme für Hilbert-Kurven (2)

- Genauer betrachten wir vier Grundformen, die als Nicht-Terminalsymbole fungieren:

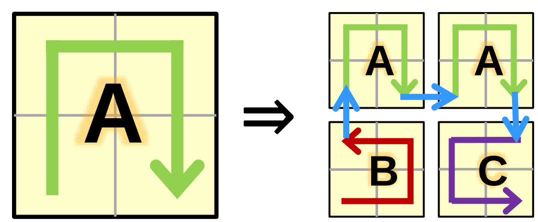


- Syntaxdiagramme zu jedem Nicht-Terminal reflektieren die rekursive Verfeinerung, hier für die Grundform A:



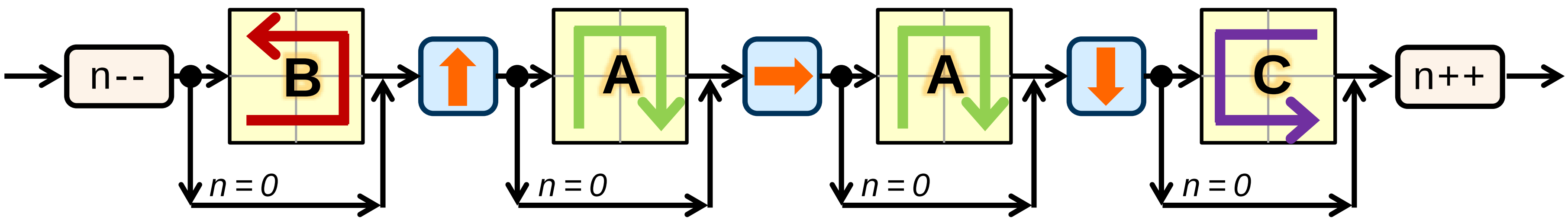
  - Für die Formen B, C, D stellt man leicht analoge Syntaxdiagramme auf

- Dies induziert z.B. die Transformation, die aus der Stufe *hilbert(1)* die Stufe *hilbert(2)* erzeugt:



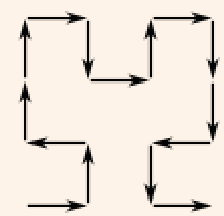
Syntaxdiagramme für Hilbert-Kurven (3)

- Man benötigt nun noch ein Rekursionsende, also einen Zweig im Syntaxdiagramm, wo nicht rekursiv verfeinert wird – wir bilden dazu einen Kurzschluss um die inneren Nicht-Terminale:



  - Die Rekursionstiefe halten wir auf einer Variablen „n" nach, die heruntergezählt wird; bei n=0 wird nicht weiter verfeinert

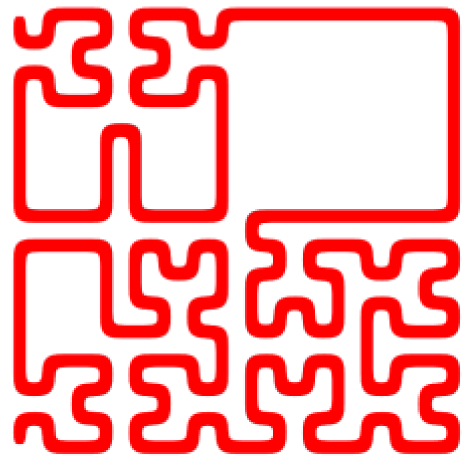
- Die Terminalsymbole , etc. interpretiert man sinnvollerweise als Plotterkommandos, welche einen entsprechenden Strich zeichnen
  - Dafür eignet sich die unten diskutierte Turtle-Grafik

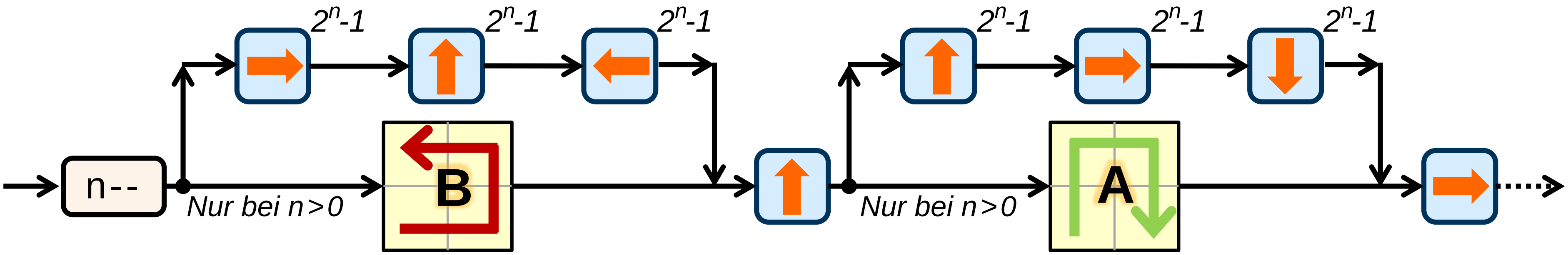


Das Ergebnis, wenn mit n=2 initial Figur A gezeichnet wird

Hilbert-Kurven unterschiedlicher Dichte

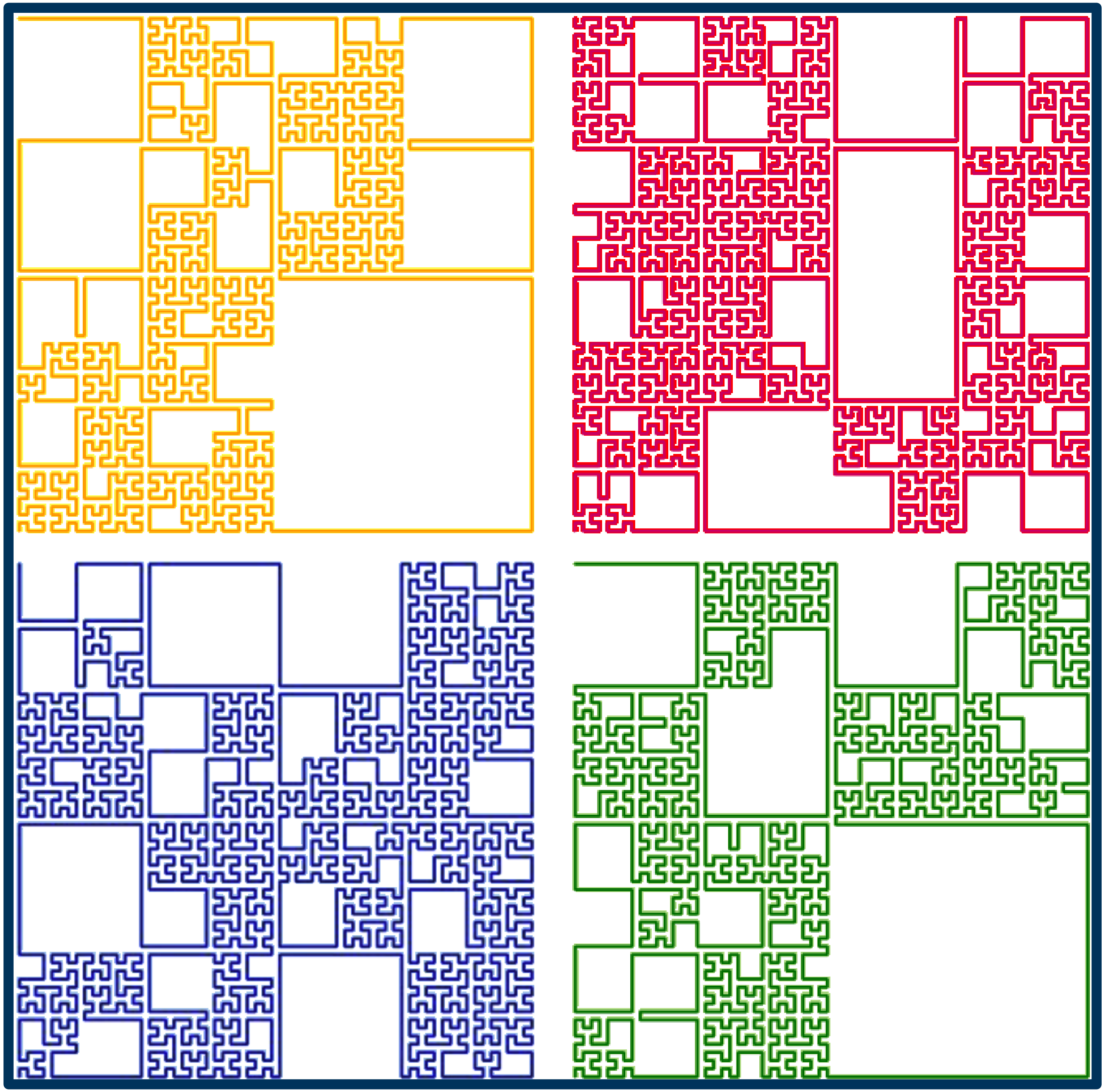
- Möchte man (auf irgendeiner Rekursionsstufe) die vier Quadranten unterschiedlich tief verfeinern, um Figuren wie im nebenstehenden Bild zu erhalten, dann müssen (als Ersatz für die inneren Figuren) längere Striche gezogen werden, deren Länge von der aktuellen Rekursionstiefe abhängt.





  - Der untere Zweig (mit indirekter Rekursion) darf jeweils nur bei n > 0 betreten werden; beim oberen Zweig sind Striche der Länge 2^n-1 zu zeichnen
  - Ob man sich ansonsten bei einer Verzweigung für oben oder unten entscheidet, mag man vom Zufall oder der zu erzeugenden Kurve abhängig machen

Hilbert-Kurven mit lokal unterschiedlicher Dichte



Die Erfindung der Hilbert-Kurve

Ueber die stetige Abbildung einer Linie auf ein Flächenstück.*)

Von

David Hilbert in Königsberg i. Pr.

Peano hat kürzlich in den Mathematischen Annalen**) durch eine arithmetische Betrachtung gezeigt, wie die Punkte einer Linie stetig auf die Punkte eines Flächenstückes abgebildet werden können. Die für eine solche Abbildung erforderlichen Functionen lassen sich in übersichtlicherer Weise herstellen, wenn man sich der folgenden geometrischen Anschauung bedient. Die abzubildende Linie — etwa eine Gerade von der Länge 1 — theilen wir zunächst in 4 gleiche Theile 1, 2, 3, 4 und das Flächenstück, welches wir in der Gestalt eines Quadrates von der Seitenlänge 1 annehmen, theilen wir durch zwei zu einander senkrechte Gerade in 4 gleiche Quadrate 1, 2, 3, 4 (Fig. 1). Zweitens theilen wir jede der Theilstrecken 1, 2, 3, 4 wiederum in 4 gleiche Theile, so dass wir auf der Geraden die 16 Theilstrecken 1, 2, 3, . . ., 16 erhalten; gleichzeitig werde jedes der 4 Quadrate 1, 2, 3, 4 in 4 gleiche Quadrate getheilt und den so entstehenden 16 Quadraten

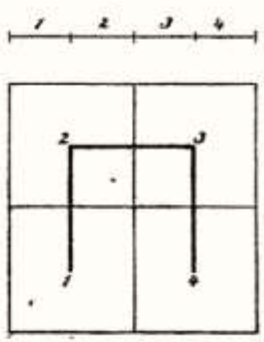


Fig. 1.

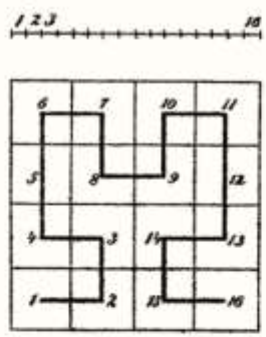


Fig. 2.

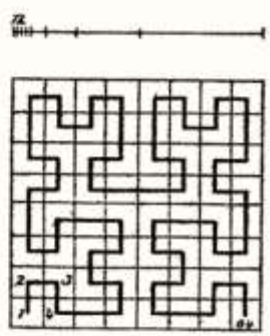


Fig. 3.

werden dann die Zahlen 1, 2 . . . 16 eingeschrieben, wobei jedoch die Reihenfolge der Quadrate so zu wählen ist, dass jedes folgende Quadrat sich mit einer Seite an das vorhergehende anlehnt (Fig. 2). Denken wir uns dieses Verfahren fortgesetzt — Fig. 3 veranschaulicht den

*) Vergl. eine Mittheilung über denselben Gegenstand in den Verhandlungen der Gesellschaft deutscher Naturforscher und Aerzte. Bremen 1890.

**) Bd. 36, S. 157.

nächsten Schritt —, so ist leicht ersichtlich, wie man einem jeden gegebenen Punkte der Geraden einen einzigen bestimmten Punkt des Quadrates zuordnen kann. Man hat nur nöthig, diejenigen Theilstrecken der Geraden zu bestimmen, auf welche der gegebene Punkt fällt. Die mit den nämlichen Zahlen bezeichneten Quadrate liegen nothwendig in einander und schliessen in der Grenze einen bestimmten Punkt des Flächenstückes ein. Dies sei der dem gegebenen Punkte zugeordnete Punkt. *Die so gefundene Abbildung ist eindeutig und stetig und umgekehrt einem jeden Punkte des Quadrates entsprechen ein, zwei oder vier Punkte der Linie.* Es erscheint überdies bemerkenswerth, dass durch geeignete Abänderung der Theillinien in dem Quadrate sich leicht *eine eindeutige und stetige Abbildung finden lässt, deren Umkehrung eine nirgends mehr als dreideutige ist.*

Die oben gefundenen abbildenden Functionen sind zugleich einfache Beispiele für überall stetige und nirgends differentiirbare Functionen.

Die mechanische Bedeutung der erörterten Abbildung ist folgende: *Es kann sich ein Punkt stetig derart bewegen, dass er während einer endlichen Zeit sämmtliche Punkte eines Flächenstückes trifft.* Auch kann man — ebenfalls durch geeignete Abänderung der Theillinien im Quadrate — zugleich bewirken, *dass in unendlich vielen überall dichtvertheilten Punkten des Quadrates eine bestimmte Bewegungsrichtung sowohl nach vorwärts wie nach rückwärts existirt.*

Was die analytische Darstellung der abbildenden Functionen anbetrifft, so folgt aus ihrer Stetigkeit nach einem allgemeinen von K. Weierstrass bewiesenen Satze*) sofort, dass diese Functionen sich in unendliche nach ganzen rationalen Functionen fortschreitende Reihen entwickeln lassen, welche im ganzen Intervall absolut und gleichmässig convergiren.

Königsberg i. Pr., 4. März 1891.

*) Vergl. Sitzungsberichte der Akademie der Wissenschaften zu Berlin, 9. Juli 1885.

David Hilbert (1891) Ueber die stetige Abbildung einer Line auf ein Flächenstück. Mathematische Annalen, 38(3), 459-460.

Sur une courbe, qui remplit toute une aire plane.

Par

G. Peano à Turin.

Dans cette Note on détermine deux fonctions x et y, uniformes et continues d'une variable (réelle) t, qui, lorsque t varie dans l'intervalle $(0, 1)$, prennent toutes les couples de valeurs telles que $0 \leq x \leq 1$, $0 \leq y \leq 1$. Si l'on appelle, suivant l'usage, *courbe continue* le lieu des points dont les coordonnées sont des fonctions continues d'une variable, on a ainsi un arc de courbe qui passe par tous les points d'un carré. Donc, étant donné un arc de courbe continue, sans faire d'autres hypothèses, il n'est pas toujours possible de le renfermer dans une aire arbitrairement petite.

Der Artikel von Peano, auf den sich Hilbert bezieht:

Giuseppe Peano: *Sur une courbe, qui remplit tout une aire plane.* Mathematische Annalen 36 (1890), S. 157–160.

MATHEMATISCHE ANNALEN.

IN VERBINDUNG MIT C. NEUMANN

BEGRÜNDET DURCH

RUDOLF FRIEDRICH ALFRED CLEBSCH.

Unter Mitwirkung der Herren

Prof. P. Gordan zu Erlangen, Prof. C. Neumann zu Leipzig,
Prof. K. VonderMühll zu Basel

gegenwärtig herausgegeben

von

Prof. Felix Klein
zu Göttingen.
Prof. Walther Dyck zu München. Prof. Adolph Mayer zu Leipzig.

XXXVI. Band.

LEIPZIG,

DRUCK UND VERLAG VON B. G. TEUBNER.

1890.

Cantor, Peano, Hilbert

Brian Hayes über „Crinkly Curves“ (Auszug), American Scientist, May-June 2013, pp. 178-183

In 1877 the German mathematician Georg Cantor made a shocking discovery. He found that a two-dimensional surface contains no more points than a one-dimensional line. Cantor compared the set of all points forming the area of a square with the set of points along one of the line segments on the perimeter of the square. He showed that the two sets are the same size. Intuition rebels against this notion. Inside a square you could draw infinitely many parallel line segments side by side. Surely an area with room for such an infinite array of lines must include more points than a single line—but it doesn’t. Cantor himself was incredulous: “I see it, but I don’t believe it,” he wrote.

Yet the fact was inescapable. Cantor defined a one-to-one correspondence between the points of the square and the points of the line segment. Every point in the square was associated with a single point in the segment; every point in the segment was matched with a unique point in the square. No points were left over or used twice. […]

Geometrically, Cantor’s one-to-one mapping is a scrambled affair. Neighboring points on the line scatter to widely separated destinations in the square. The question soon arose: Is there a *continuous* mapping between a line and a surface? In other words, can one trace a path through a square without ever lifting the pencil from the paper and touch every point at least once? It took a decade to find the first such curve.

The first successful recipe for a space-filling curve was formulated in 1890 by Giuseppe Peano […] Peano did not provide a diagram or even an explicit description of what his curve might look like; he merely defined a pair of mathematical functions that give x and y coordinates inside a square for each position t along a line segment. Soon David Hilbert […] devised a simplified version of Peano’s curve and discussed its geometry.

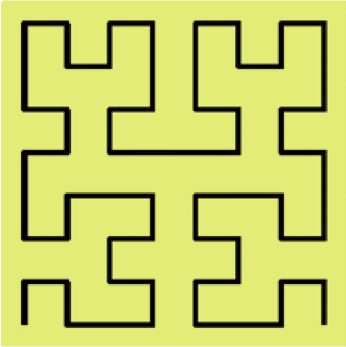
In everyday speech the word *curve* suggests something smooth and fluid, without sharp corners, such as a parabola or a circle. The Hilbert curve is anything but smooth. All finite versions of the curve consist of 90-degree bends connected by straight segments. In the infinite limit, the straight segments dwindle away to zero length, leaving nothing but sharp corners. The curve is all elbows.

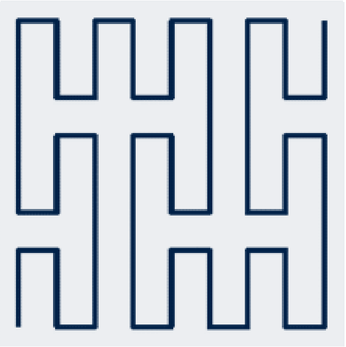
Kurven?

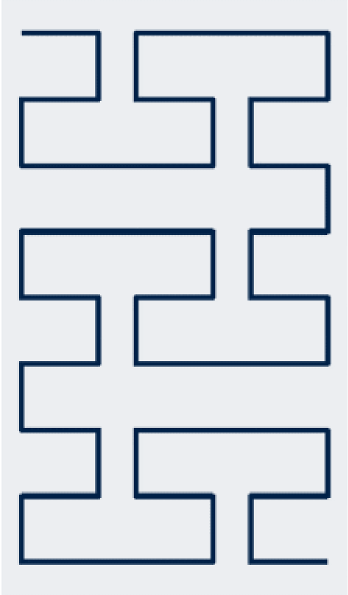
„Was ist eigentlich eine Kurve? Diese Frage gehört zu denen, die leichter aufzuwerfen als zu beantworten sind. [...] Um das Jahr 1880 hat C. Jordan eine Definition aufgestellt, die ungefähr das wiedergibt, was uns anschaulich vorschwebt. Diese Definition besagt im Grunde folgendes: Eine Kurve ist das, was ein Punkt bei stetiger Bewegung durchläuft. Die Bewegung eines Punktes beherrschen wir, wenn wir von jedem Zeitaugenblick angeben können, wo sich der Punkt befindet, wenn wir also den Ort als Funktion der Zeit kennen. Wenn sich ein Punkt in der Ebene bewegt, ist der Ort durch zwei Zahlen x, y bestimmt, die Zeit durch eine Zahl t. Die Bewegung wird also vollständig beschrieben sein, wenn ich angebe, wie x und y von t abhängen. [...]

Nehmen wir an, dass die Bewegung in der Zeiteinheit – sagen wir in einer Minute – vor sich gehe, dann können wir t auf das Intervall von 0 bis 1 beschränken. Die Definition Jordans läuft dann auf folgendes hinaus: Eine Kurve ist ein stetiges und eindeutiges Abbild der Einheitsstrecke. Um so merkwürdiger ist die Entdeckung Peanos aus dem Jahre 1890, dass es Gebilde gibt, die eindeutige und stetige Bilder der Einheitsstrecke sind, die also Kurven darstellen im Sinne Jordans, die aber im strengen Sinne des Wortes ein ganzes Quadrat füllen. Wollte man so eine Kurve zeichnen, so müsste man eine ganze Quadratfläche schwarz anfärben – das wäre das Bild der Kurve.

Das Problem, das sich Peano und Hilbert gesetzt hatten, war folgendes: Gegeben sind die Einheitsstrecke und das Einheitsquadrat. Wie kann man diese beiden Gebilde so aufeinander beziehen, dass jedem Punkt der Einheitsstrecke genau ein Punkt des Quadrates entspricht und dass die Abbildung stetig ist? Anschaulich gesprochen: Denken wir uns einen Reisenden, der in einer Minute alle Punkte eines Quadrates passieren soll. Wie wird seine Reiseroute aussehen?







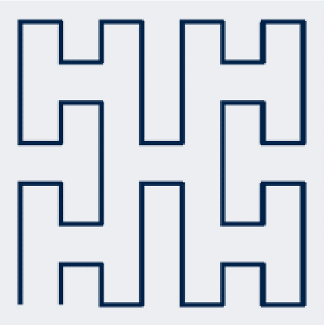
Raumfüllende Kurven?

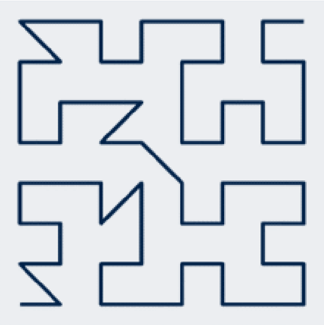
Wir teilen die Minute in vier gleiche Teile und ebenso das Quadrat und richten es nun so ein, dass der Reisende in einer Viertelminute immer gerade ein Viertel des Quadrates bereist. In der ersten Viertelminute soll er das erste Quadratviertel durchwandern, ohne es später wieder zu berühren, in der zweiten Viertelminute das zweite Quadratviertel usw. [...] Wir setzen den Teilungsprozess fort, wir teilen also jedes Viertel der Quadratfläche wieder in vier gleiche Teile, die ganze Quadratfläche also in 16 gleiche Intervalle, die ebenfalls die Nummern 1 bis 16 erhalten, und ordnen nun jeder Teilstrecke das Teilquadrat mit der gleichen Nummer zu. Dabei müssen wir nur darauf achten, dass zwei Quadrate mit aufeinanderfolgenden Nummern eine Seite gemein haben. Das ermöglicht es uns, Strecke und Quadrat in der gewünschten Weise aufeinander abzubilden. Zu dem Zweck haben wir dreierlei nachzuweisen:

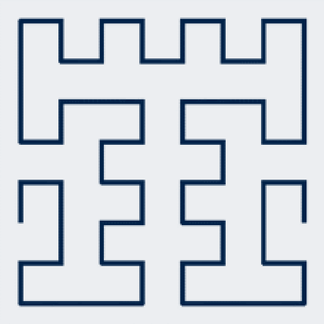
1. Jedem Punkt der Einheitsstrecke entspricht ein Punkt des Quadrats (d.h. in jedem Augenblick befindet sich der Reisende an genau einer Stelle der Fläche).
2. Kein Punkt des Quadrates geht leer aus (der Reisende kommt überall hin).
3. Die Abbildung ist stetig (er bewegt sich in einer zusammenhängenden Linie).

Durch die Entdeckung Peanos scheint sich einer der fundamentalen Unterschiede zu verwischen, der Unterschied zwischen den Dimensionen. Wenn eine Kurve eine Fläche erfüllen kann – wie soll man dann zwischen ein- und zweidimensional unterscheiden?“

Aus „Einführung in das mathematische Denken“ von Friedrich Waismann (1936). Waismann (1896 – 1959) war ein österreichischer Mathematiker und Philosoph, Mitglied des Wiener Kreises (dem u.a. Rudolf Carnap und Kurt Gödel angehörten, und mit dem auch Ludwig Wittgenstein, Alfred Tarski, Oskar Morgenstern und Karl Popper Kontakt pflegten). 1938 nach Grossbritannien emigriert, wirkte er in Cambridge und dann in Oxford.







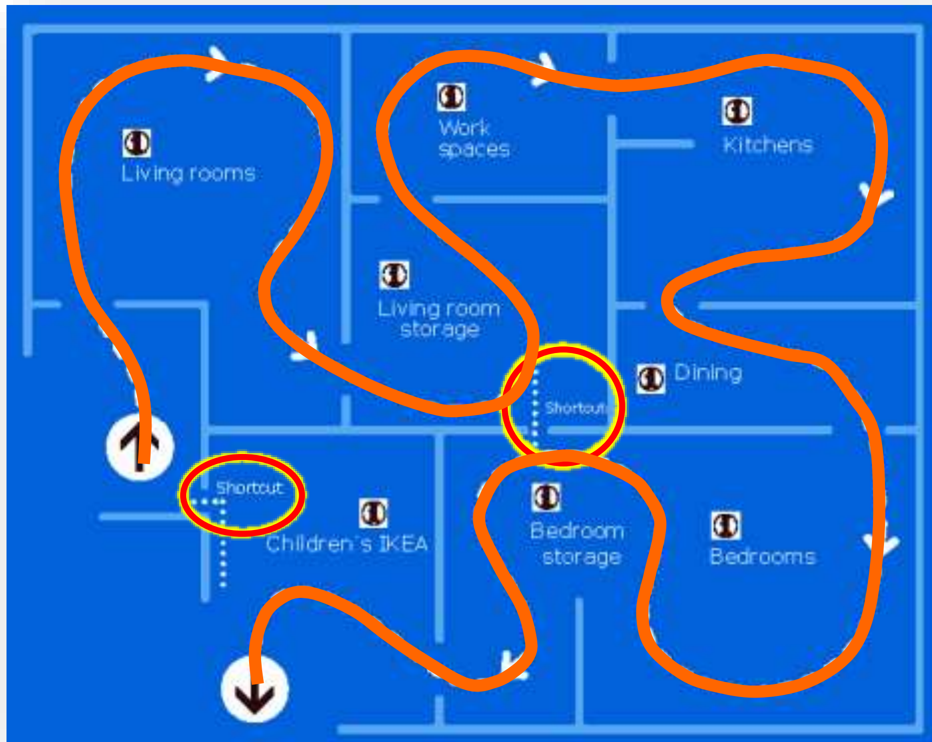
Hilbert-Kurve – raumfüllend bei IKEA

IKEA stores are notoriously hard to navigate in a fast way. For instance, every 15 meters there is a curve. The reason is that if the aisle is a long straight line you will subconsciously raise your eyes and look for the horizon, and miss all the shopping opportunities on your sides. The maze-like showrooms ensure that you spend more time in the store, thereby increasing the likelihood of making extra impulse purchases.

The standard route meanders through the store. What you should look for however is escape alleys and secret doors. They are not highlighted in stores, in fact IKEA is putting a lot of effort into making them less visible. But using them will get you through in only 10 minutes.

[nordic.businessinsider.com]

Look at the floorplan of an IKEA store, they very often use a Hilbert curve as the route you have to follow to get from entrance to exit. The logic behind this is obvious from the space-filling nature of the curve." – Steve Baker

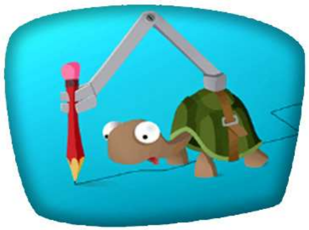


"People can have a hard time navigating the store. There have been stories of people saying that they feel like we are purposely keeping them in.

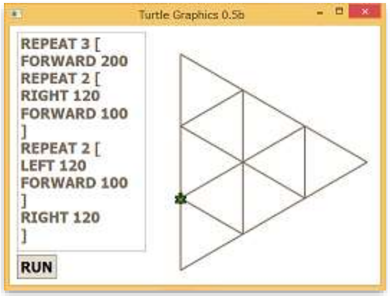
We want them to be able to find what they are looking for. We want them to discover all the features and services in the store that make it a pleasant shopping experience, rather than getting lost!"

-- Laurie Satran, IKEA

Rekursive Hilbert-Kurve mit Turtle-Grafik!



Mit **Turtle-Grafik** wird eine Bildbeschreibungssprache bezeichnet, bei der man sich vorstellt, dass ein stifttragender Roboter (die Schildkröte) sich auf der Zeichenebene bewegt und mit einfachen Kommandos, wie Stift heben, senken, vorwärts laufen und drehen, gesteuert werden kann. [Wikipedia]



Zwei Formen von Pseudocode:

Let y be the direction opposite to x.

- Turn in the direction y
- Draw a Hilbert curve of order n-1, direction y
- Move one step forward
- Turn in the direction x
- Draw a Hilbert curve of order n-1, direction x
- Move one step forward
- Draw a Hilbert curve of order n-1, direction x
- Turn in the direction x
- Move one step forward
- Draw a Hilbert curve of order n-1, direction y

```
def hilbert(level, angle):
    if level == 0: return
    right(angle)
    hilbert(level-1, -angle)
    forward(size)
    left(angle)
    hilbert(level-1, angle)
    forward(size)
    hilbert(level-1, angle)
    left(angle)
    forward(size)
    hilbert(level-1, -angle)
    right(angle)
```

Quellen: http://users.polytech.unice.fr/~helen/pythonCIP1/images/tortue.png http://stackoverflow.com/questions/7218269/hilbert-curve-analysis http://stackoverflow.com/questions/35813844/generating-a-3d-space-filling-hilbert-curve-using-turtle-graphics/35819302

Rekursives Hilbert-Turtle-Programm in Java

```
public class Hilbert
{   private Turtle turtle;
    public Hilbert(int n)
    {   turtle = new Turtle(); double max = Math.pow(2, n);
        turtle.setXscale(0, max); turtle.setYscale(0, max); hilbert(n);
    }
```

```
private void hilbert(int n)
{   if (n == 0) return;
    turtle.turnLeft(90);
    treblih(n-1);
    turtle.goForward(1.0);
    turtle.turnLeft(-90);
    hilbert(n-1);
    turtle.goForward(1.0);
    hilbert(n-1);
    turtle.turnLeft(-90);
    turtle.goForward(1.0);
    treblih(n-1);
    turtle.turnLeft(90);
}
```

```
public void treblih(int n)
{   if (n == 0) return;
    turtle.turnLeft(-90);
    hilbert(n-1);
    turtle.goForward(1.0);
    turtle.turnLeft(90);
    treblih(n-1);
    turtle.goForward(1.0);
    treblih(n-1);
    turtle.turnLeft(90);
    turtle.goForward(1.0);
    hilbert(n-1);
    turtle.turnLeft(-90);
}
```

```
  public static void main(String[] args)
    { int n = Integer.parseInt(args[0]);
      new Hilbert(n);
    }
}  // http://introcs.cs.princeton.edu/java/32class/Hilbert.java.html (Robert Sedgewick)
```

Die Klasse „Turtle" realisiert die offensichtlichen Kommandos (z.B. unter Rückgriff auf elementare Zeichenfunktionen einer Library), siehe z.B. http://introcs.cs.princeton.edu/java/32class/Turtle.java.html

Seymour Papert und die Turtle-Grafik

Seymour Papert (1928 – 2016) war Professor für Mathematik und Pädagogik am MIT; er gründete zusammen mit Marvin Minsky das MIT Artificial Intelligence Lab sowie zusammen mit Nicholas Negroponte das MIT Media Lab. 1967 entwickelte er (als Schüler von Jean Piaget und Anhänger der konstruktivistischen Lerntheorie) die Sprache Logo, mit der Kinder auf experimentell-intuitive Weise Programmieren lernen sollten. Hierzu dient die Turtle-Grafik, bei der sich Cursor als „Schildkröten“ über den Bildschirm bewegen lassen, die Linien malen. Bevor die Display-Technik soweit war, gab es schon erste Turtle-Roboter für Logo.

https://commons.wikimedia.org/wiki/File:Seymour_Papert.jpg

https://sudburybeach.files.wordpress.com/2017/03/mindstorms.jpg

Seymour Papert und die Turtle-Grafik (2)

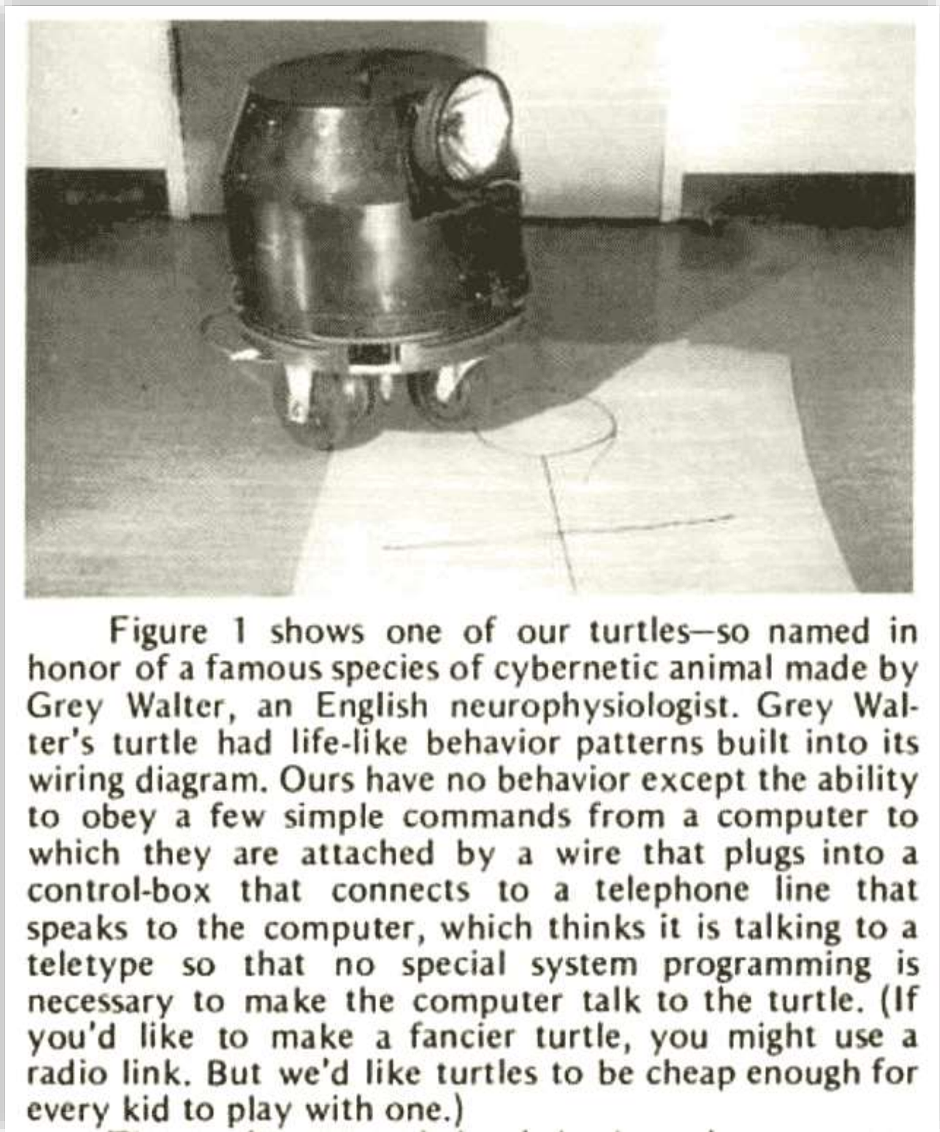


Figure 1 shows one of our turtles—so named in honor of a famous species of cybernetic animal made by Grey Walter, an English neurophysiologist. Grey Walter's turtle had life-like behavior patterns built into its wiring diagram. Ours have no behavior except the ability to obey a few simple commands from a computer to which they are attached by a wire that plugs into a control-box that connects to a telephone line that speaks to the computer, which thinks it is talking to a teletype so that no special system programming is necessary to make the computer talk to the turtle. (If you'd like to make a fancier turtle, you might use a radio link. But we'd like turtles to be cheap enough for every kid to play with one.)

http://cyberneticzoo.com/wp-content/uploads/Papert-Turtle-x640.jpg

Die erste Logo-Roboterschildkröte entstand 1969 / 70; es war ein drahtgebundenes gelbes Gefährt auf drei Rädern, das wie ein umgestülpter Papierkorb aussah. Erste funkferngesteuerte Roboterschildkröten für Logo entstanden 1972. Damals gab es noch keine PCs, und die Interaktion mit Computern erfolgte über Fernschreibgeräte (teletype, „tty“).

http://cyberneticzoo.com/wp-content/uploads/22-turtle-x640.jpg

Links: Aus “Twenty Things to Do with a Computer” von Seymour Papert und Cynthia Solomon, 1971

„Divide et impera" – ein wichtiges Paradigma

- Bezeichnung eines Problemlösungsprinzips in der Informatik
  - Bei dem typischerweise Rekursion verwendet wird
- Ursprung der Redewendung aber in der Machtpolitik

Teile und herrsche; divide and conquer

Bildquelle: Wikipedia (Till Niermann, CC-BY-SA-3.0)

„Divide et impera“ – ein wichtiges Paradigma

- Bezeichnung eines Problemlösungsprinzips in der Informatik
  - Bei dem typischerweise Rekursion verwendet wird
- Ursprung der Redewendung aber in der Machtpolitik
  - Gegner, Volksgruppen oder Untertanen hinsichtlich Merkmalen wie Religion, Ethnie etc. in Teilgruppen aufspalten
  - Einzelne Teilgruppen getrennt besiegen oder spezifisch unter Kontrolle halten
  - Uneinigkeit unter den Teilgruppen fördern, diese evtl. sogar gegeneinander ausspielen
- Im römischen Reich und in der Kolonialzeit (und auch darüber hinaus) angewendet
  - Oft mit lang anhaltenden Konsequenzen

„Divide et impera" – ein wichtiges Paradigma

Haben wir implizit schon mehrfach verwendet

- *Beispielproblem: Jüngste Person im Hörsaal bestimmen*

1) Teile das Problem in zwei „Hälften"

Oder evtl. in mehr als zwei Teile?

2) Löse die beiden kleineren / einfacheren Teilprobleme

Meist auf die gleiche Art: rekursiv!

Immer kleinere Teilprobleme

3) Setze Gesamtlösung aus den Lösungen der Teilprobleme zusammen

Das geht natürlich nur bei Problemen mit geeigneter modularer Struktur

- *Im Beispiel: jüngst (Hörsaal) =*

  jüngst { *jüngst{linke Hälfte}*, *jüngst{rechte Hälfte}* ***}***

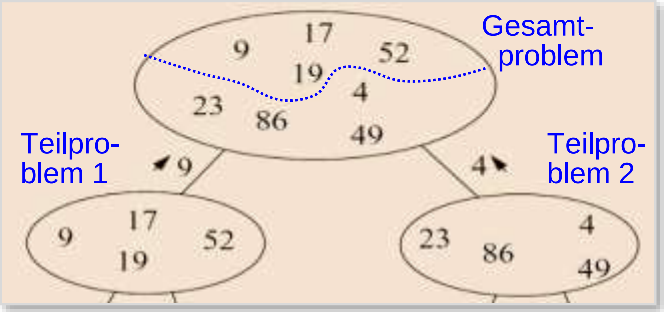
- Evtl. Teilprobleme gleichzeitig bearbeiten (→ Parallelisierung)?

„Divide et impera“: Voraussetzungen

- Das Problem muss beim Partitionieren „einfacher“ / kleiner werden
  - Sollte für jedes der (möglichst etwa gleich grossen!) Teilprobleme gelten
- Wenn ein Teilproblem „atomar“ geworden ist (z.B. eine ein-elementige Menge), dann muss dafür eine (hoffentlich „triviale“) Lösung auf andere Art existieren bzw. gefunden werden
- Gesamtlösung soll aus den Teillösungen einfach kombinierbar sein
  - Und dies sollte effizient durchführbar sein

Man „sieht“ direkt, dass dies zu einer Baumstruktur führt!

Backtracking- und Spielbäume repräsentieren oft Instanzen von divide et impera



- Nicht mit divide et impera bezeichnet wird im Allgemeinen die direkte Anwendung eines Verfahrens rekursiv auf eine <u>einzige</u> (einfachere) Instanz des gleichen Problems („decrease and conquer“) wie z.B. altägyptische Multiplikation, euklidischer Algorithmus, Binärsuche mit Intervallhalbierung etc.

Beispiel für divide et impera: Rekursive Bestimmung des Minimums einer geordneten Menge

(1) Teile die Menge P in 2 Teilmengen P1, P2 („Partition"), so dass:

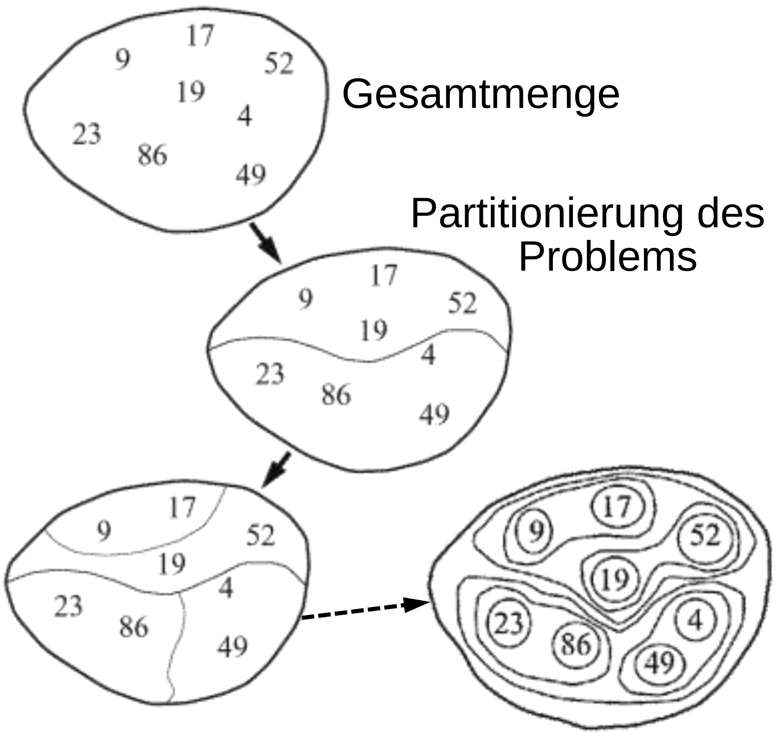
- $P1 \cup P2 = P$
- $P1 \cap P2 = \emptyset$

(2) Löse das Problem für die beiden jew. kleineren Mengen P1 und P2

- Sei p′ das Minimum von P1
- Sei p″ die entspr. Lösung für P2

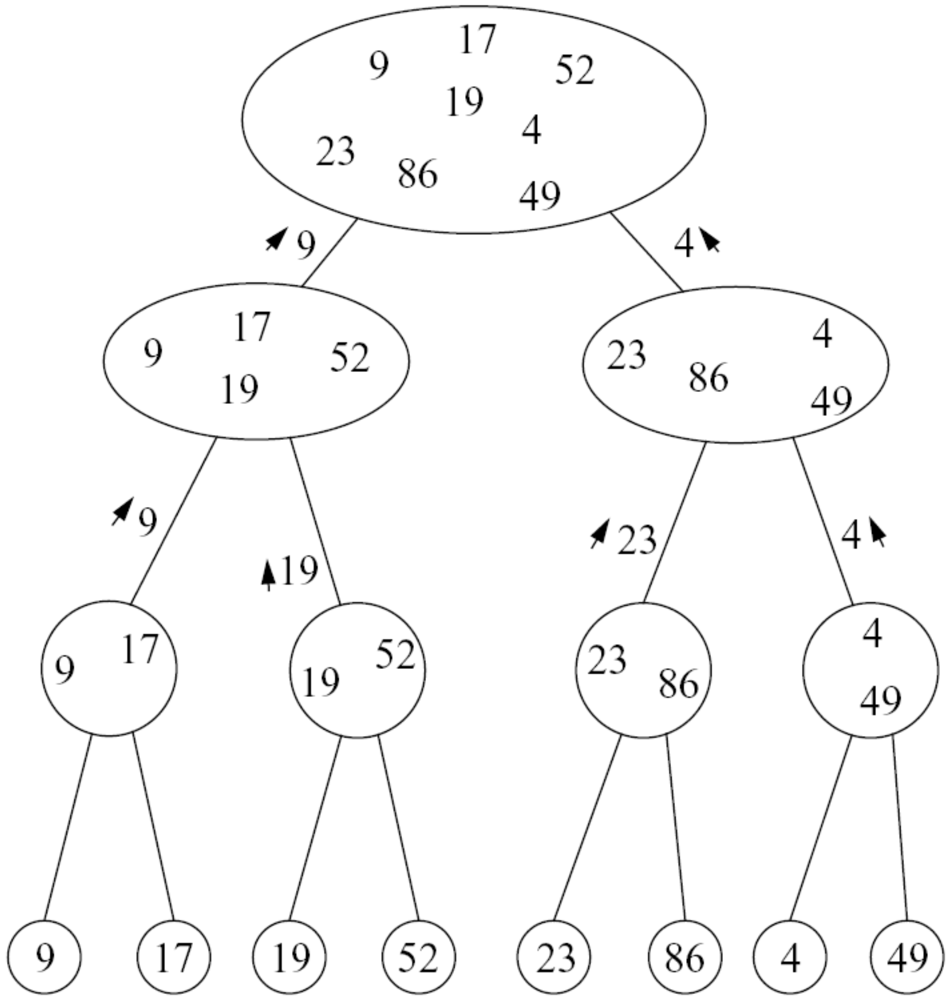
(3) Vergleiche p′ mit p″ und liefere den kleineren der beiden Werte als Ergebnis p zurück

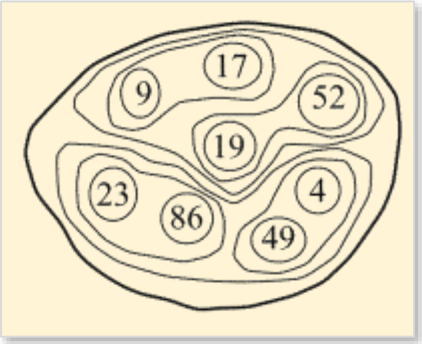
- Also: p = min(p′,p″)



Das ist ein Baum
(in Darstellung als Mengendiagramm)

Eine andere Darstellung des Rekursionsbaums





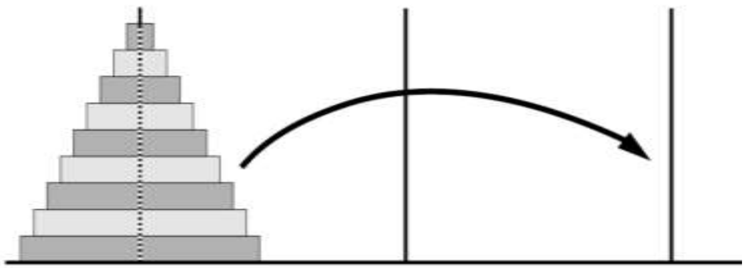
Erinnert an
Bottom-up-Minimax

Die Türme von Hanoi –
Ein Klassiker der rekursiven Problemlösung

http://blocs.xtec.cat/ceipsantmarticm/files/2009/07/torre-de-hanoi.jpg

Türme von Hanoi

- Der Legende nach versucht sich ein buddhistischer Mönchsorden seit vielen Jahren an folgendem Problem:
  - 64 goldene Scheiben verschiedener Grösse sind auf **Turm 1** aufeinander gestapelt
  - Es darf immer nur eine kleinere Scheibe auf einer grösseren liegen, nie umgekehrt
  - Scheiben dürfen nur einzeln von einem Turm zu einem anderen gebracht werden
  - Am Ende sollen alle Scheiben korrekt auf **Turm 3** liegen



- Gesucht ist ein Algorithmus, der angibt, wann welche Scheibe von wo nach wo zu bewegen ist
  - Verfügbar ist ein anfangs leerer **Turm 2**

Ein Problem, das sich relativ leicht rekursiv lösen lässt, aber nicht so einfach ohne Rekursion

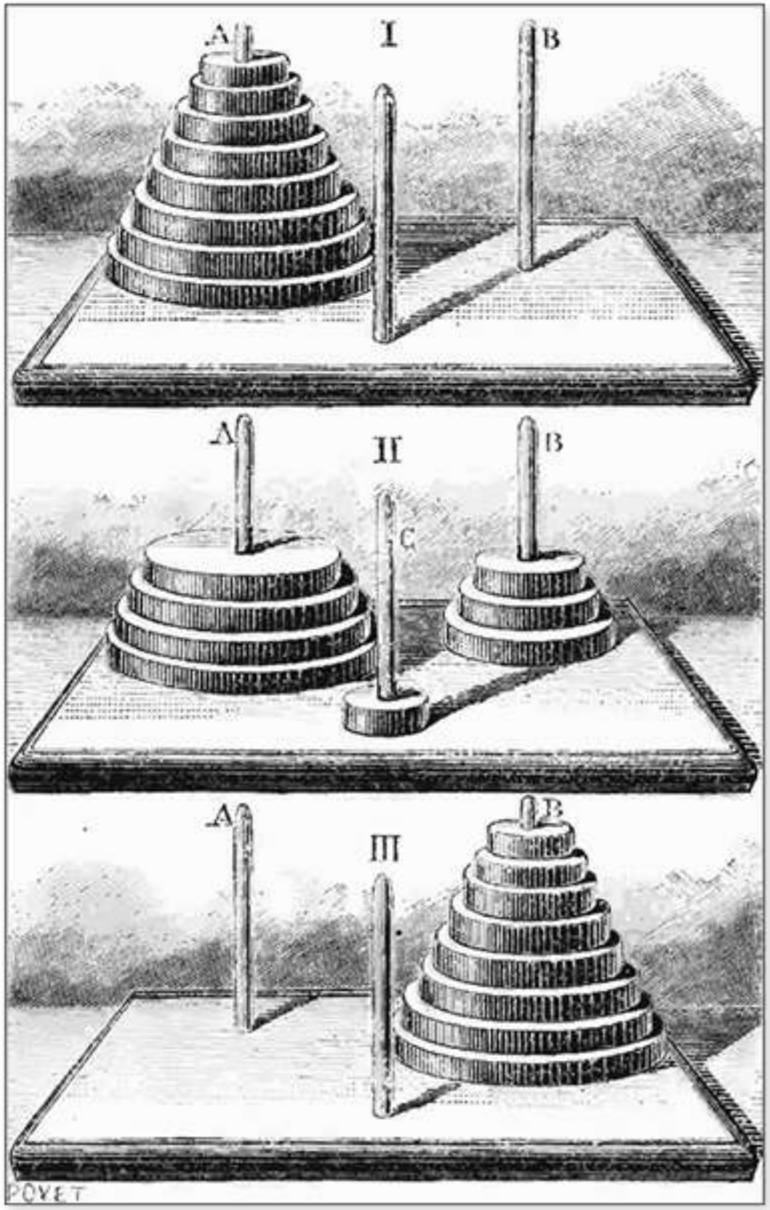
1883 vom französischen Mathematiker Edouard Lucas beschrieben

La Tour de Hanoï

[par Édouard LUCAS, publié à titre posthume en 1892]

Dans le grand temple de Bénarès, au-dessous du dôme qui marque le centre du monde, on aperçoit trois aiguilles de diamant, plantées dans une dalle d'airain, hautes d'une coudée et grosses comme le corps d'une abeille. Sur l'une de ces aiguilles, le dieu PARABAVASTÛ enfila, au commencement des siècles, soixante-quatre disques d'or pur, le plus large reposant sur le bronze et les autres, de plus en plus étroits, superposés jusqu'au sommet. C'est la *Tour Sacrée de* VISHNOU. Nuit et jour, les bonzes se succèdent sur les marches de l'autel, occupés à transporter la Tour Sacrée, de la première aiguille sur la troisième, étage par étage, sans jamais intervertir, sans jamais s'écarter des règles immuables imposées par BRAHMA. Quand tout sera fini, la Tour et les Brahmes tomberont et ce sera la fin des mondes. […]

La Tour d'Hanoï, nouvellement restaurée, se compose d'étages superposés et décroissants, en nombre variable, […] très étroits au sommet, plus larges à la base, percés à leur centre et agrémentés des couleurs tonkinoises.



La Tour de Hanoï

Le jeu consiste à démolir la Tour, étage par étage, et à la reconstruire dans un lieu voisin conformément aux règles indiquées. Les règles du jeu sont les suivantes : […]

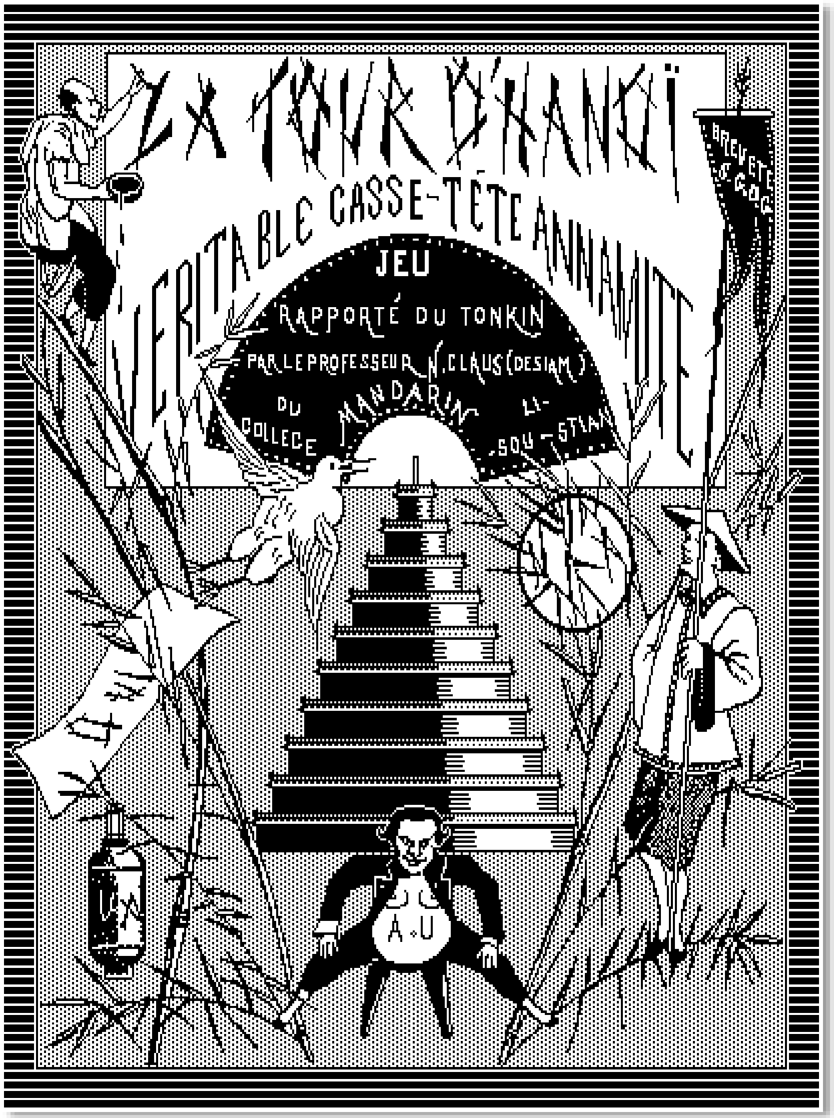
I. – On ne peut déplacer à chaque coup que l'étage supérieur d'une pile.

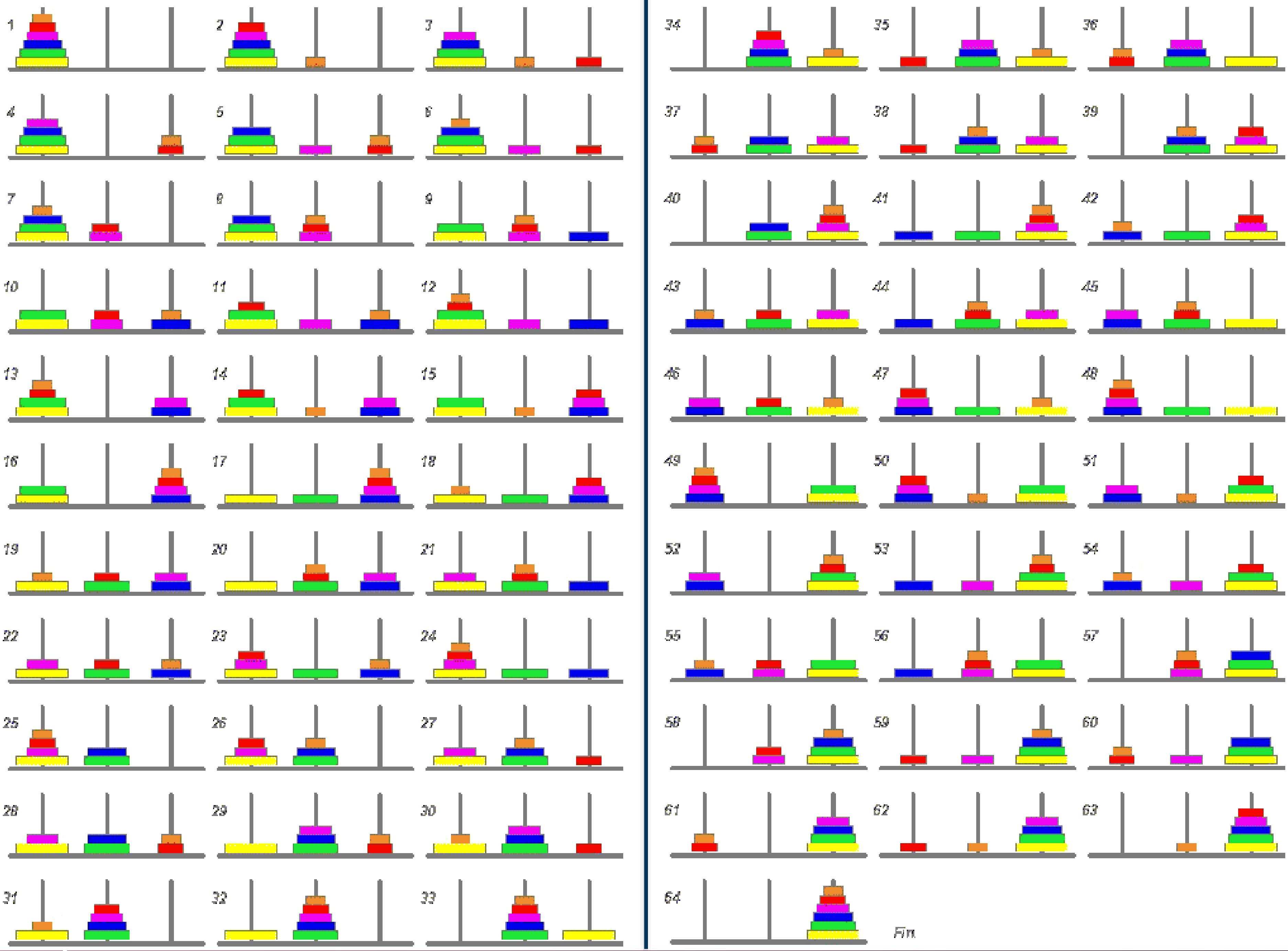
II. – On peut enlever l'étage supérieur d'une pile pour l'enfiler sur une tige relevée n'ayant encore aucun étage.

III.– On peut enlever l'étage supérieur d'une pile et le placer sur une autre pile; à la condition expresse que l'étage supérieur de celle-ci soit plus grand.

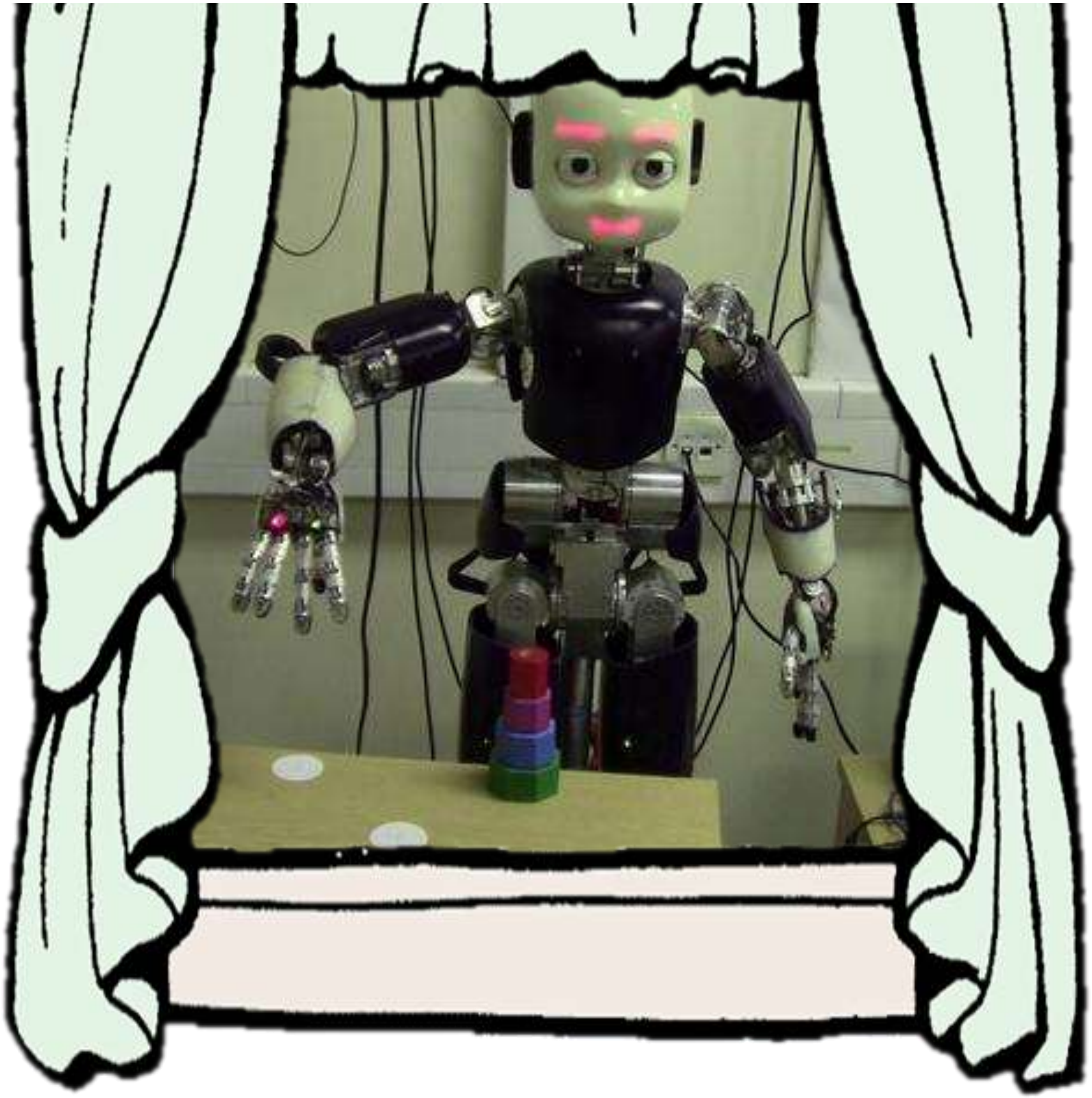
Ce jeu est donc la représentation sensible de la *Question de l'Etage,* si importante dans l'existence. Amusant et instructif, facile à apprendre et à jouer, à la ville, à la campagne, en voyage, il a pour but la vulgarisation des sciences. […]

La Tour d'Hanoï n'est, en réalité, que la représentation sensible de l'Arithmétique binaire; c'est une transformation du Boulier chinois de FO-CHI et du Baguenaudier. Nous avons déjà indiqué ces résultats dans une Conférence faite au théâtre de Blois, en 1884, pendant le Congrès de l'Association française pour l'Avancement des Sciences.




1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
Fin

Türme von Hanoi – Animation mit Roboter



Video [2:21] by Yanyu Su and Kyuhwa Lee, Imperial College London, http://vimeo.com/41611733
or www.youtube.com/watch?v=KOgmSpfcxCY (“iCub Block Manipulation”)

Türme von Hanoi

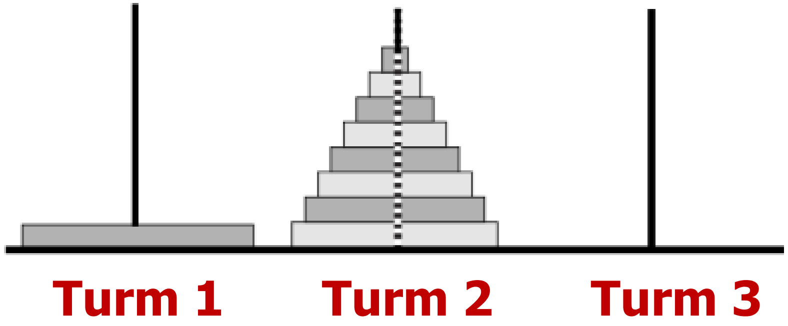
http://k46.kn3.net/taringa/7/E/0/A/8/4/LaNegraOprah/043.jpg

https://i.ytimg.com/vi/Ubyg9lhF4dQ/hqdefault.jpg

Türme von Hanoi – Lösungsidee

Gesucht ist ein Algorithmus, der angibt, wann welche Scheibe von wo nach wo zu bewegen ist

- Bedingung dafür, die unterste (grösste) Scheibe schliesslich irgendwann von Turm 1 wegzubewegen, z.B. nach Turm 3:
  - (a) Auf Turm 1 befindet sich nichts sonst
  - (b) Turm 3 ist leer



- Aus (a) und (b) folgt:
  - (c) Alle anderen Scheiben befinden sich auf Turm 2!

- → Es müssen zunächst die n-1 obersten Scheiben von Turm 1 nach Turm 2 gebracht werden

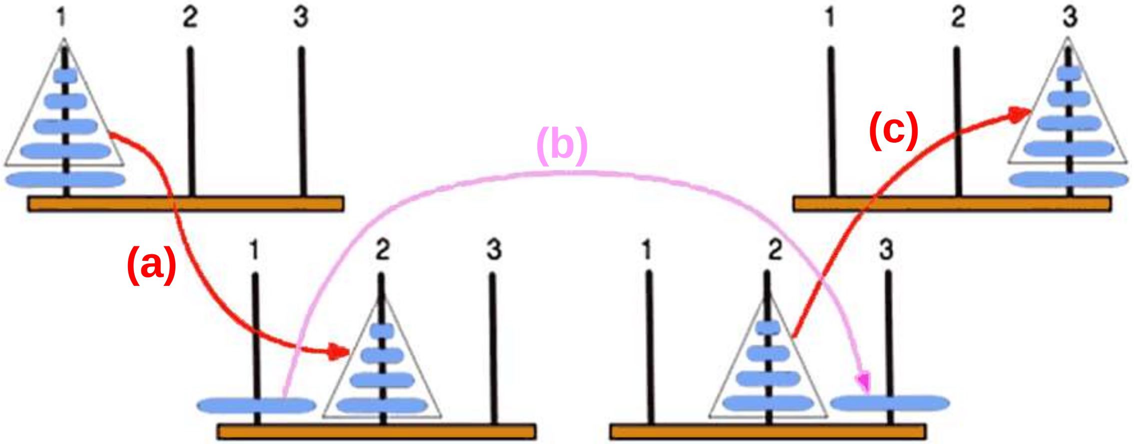
- Dies ist das gleiche Problem in kleinerer Dimension

  (denn offenbar beeinflusst die dabei nicht betrachtete unterste Scheibe auf Turm 1 nicht die Lösung des Teilproblems)

Rekursionsansatz

- „Das gleiche Problem in kleinerer Dimension"
- → Lösungsansatz für das Gesamtproblem:

(a) Bringe den „n-1 Turm" von 1 nach 2
(b) Bewege eine Scheibe von 1 nach 3
(c) Bringe den „n-1 Turm" von 2 nach 3



Wieso ist folgende ähnliche Überlegung falsch?
(a) Bewege die oberste (kleinste) Scheibe von 1 nach 2
(b) Bringe den restlichen „n-1 Turm" von 1 nach 3
(c) Bewege die eine (kleinste) Scheibe von 2 nach 3

Rekursionsansatz bei der Routenplanung?

Eine Denkübung

- „Das gleiche Problem in kleinerer Dimension"

- Man möchte auf kürzestem Weg von Zürich nach Mailand fahren und löst das Problem rekursiv: Es wird zunächst Bellinzona als Zwischenziel auf kürzestem Weg angesteuert, sodann von Bellinzona aus auf kürzestem Weg das Endziel Mailand.
- Wieso versagt dieser Ansatz hier im Unterschied zum Türme-von-Hanoi-Problem? Lässt sich das Problem doch irgendwie rekursiv angehen?

Das rekursive Java-Programm

3 Türme: Turm **1**, Turm **2**, Turm **3**

```
void bewege(int von, int nach) {
  System.out.println("Oberste Scheibe von Turm " +
                          von + " nach Turm " + nach);
}

void hanoi(int groesse, int von, int nach) {
  if (groesse == 1)
    bewege(von, nach);
  else {
    hanoi(groesse-1, von, 6-von-nach);
    bewege(von, nach);
    hanoi(groesse-1, 6-von-nach, nach);
  }
}
```

Könnte man die Rekursion nicht bei 0 statt 1 aufhören lassen? Wie sieht das dann aus?

Der „andere" Turm

Das ist ein Programm ohne Zuweisung! (→ funktionaler Programmierstil)

Veranschaulichung des Rekursionsbaums

Umsetzen eines Stapels der Höhe 4 von Turm 1 nach Turm 3:

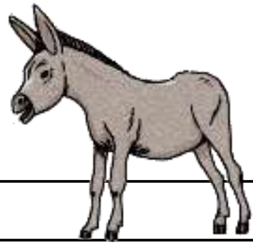
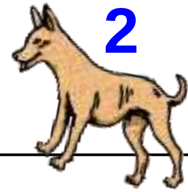
Depth-first

			hanoi(1,1,2)	bewege(1,2)
		hanoi(2,1,3)		bewege(1,3)
			hanoi(1,2,3)	bewege(2,3)
	hanoi(3,1,2)			bewege(1,2)
			hanoi(1,3,1)	bewege(3,1)
		hanoi(2,3,2)		bewege(3,2)
			hanoi(1,1,2)	bewege(1,2)
hanoi(4,1,3)				bewege(1,3)
			hanoi(1,2,3)	bewege(2,3)
		hanoi(2,2,1)		bewege(2,1)
			hanoi(1,3,1)	bewege(3,1)
	hanoi(3,2,3)			bewege(2,3)
			hanoi(1,1,2)	bewege(1,2)
		hanoi(2,1,3)		bewege(1,3)
			hanoi(1,2,3)	bewege(2,3)

Zeit

Programm als Anweisungsfolge für einen Roboter

Veranschaulichung des Rekursionsbaums

1  2  4  8 

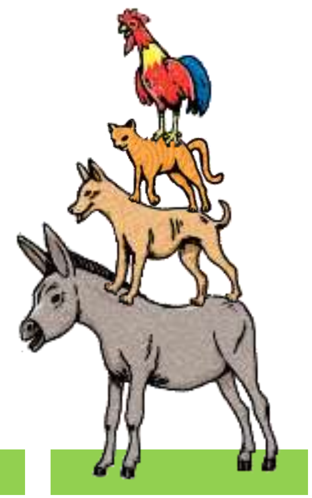
← Zählen im Dualsystem?

			hanoi(1,1,2)	bewege(1,2)
		hanoi(2,1,3)		bewege(1,3)
			,2,3)	bewege(2,3)
				bewege(1,2)
			,3,1)	bewege(3,1)
				bewege(3,2)
			,1,2)	bewege(1,2)
hanoi(4,				bewege(1,3)
			,2,3)	bewege(2,3)
				bewege(2,1)
			,3,1)	bewege(3,1)
				bewege(2,3)
			,1,2)	bewege(1,2)
		hanoi(2,1,3)		bewege(1,3)
			hanoi(1,2,3)	bewege(2,3)

Zeit

1 Esel
2 Hund
4 Katze
+ 8 Vogel

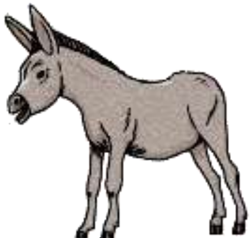
15 Schritte



3 2 1

Programm als Anweisungsfolge für einen Roboter

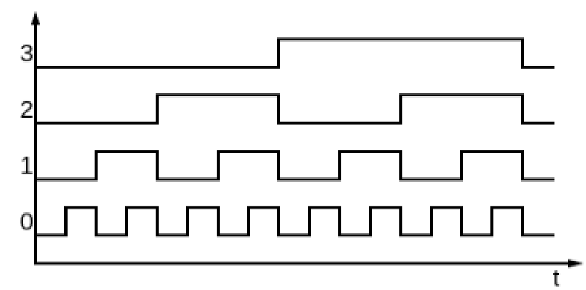
Getakteter Marschplan und der Gray-Code




Zeit	1	2	4	8
1	0	0	0	1
2	0	0	1	0
3	0	0	0	1
4	0	1	0	0
5	0	0	0	1
6	0	0	1	0
7	0	0	0	1
8	1	0	0	0
9	0	0	0	1
10	0	0	1	0
11	0	0	0	1
12	0	1	0	0
13	0	0	0	1
14	0	0	1	0
15	0	0	0	1

Gibt in jedem Zeittakt an, wer sich dann zu bewegen hat.

Das Ziel ist für jeden eindeutig, da die Regel einzuhalten ist (nie auf einem kleineren zur Ruhe kommen bzw. nicht dorthin, wo der Hahn ist); nur der Hahn braucht Orientierung (etwa: nicht direkt wieder zurückfliegen).



Fasst man jede 1 als einen Impuls auf, der eine jeder Spalte zugeordnete Binärstelle umschaltet, dann erhält man den sogenannten Gray-Code (für 4 Bit): 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100,... Zählt man umgekehrt im Gray-Code von 0000 an hoch, gibt diejenige eindeutige Bitposition, die sich dabei jeweils ändert, an, welche Scheibe bewegt werden soll.

Die Bremer Stadtmusikanten lassen grüssen

Volksmärchen, 1819 veröffentlicht in „Grimms Märchen". Es geht um vier im Alter schlecht behandelte Haustiere (Hahn, Katze, Hund und Esel), die fortlaufen, in Bremen Stadtmusikanten werden wollen, dort aber nie ankommen, weil es ihnen unterwegs gelingt, das Haus einer Räuberbande zu erobern, indem sie die Räuber durch eine Figurenpyramide sowie lauten „Gesang" erschrecken und vertreiben.

Allegorisch entsprechen die Tiere den im Dienst bei der Herrschaft alt gewordenen, abgearbeiteten und durch den Verlust an Leistungskraft nutzlos gewordenen Knechten und Mägden. Mit ihrem Aufbruch, ihrem Zusammenhalt und Mut schaffen sie das fast Unmögliche. Sie überlisten die Bösen, schaffen sich ein Heim und somit ein neues Leben. Und wenn sie nicht gestorben sind, dann leben sie noch heute.

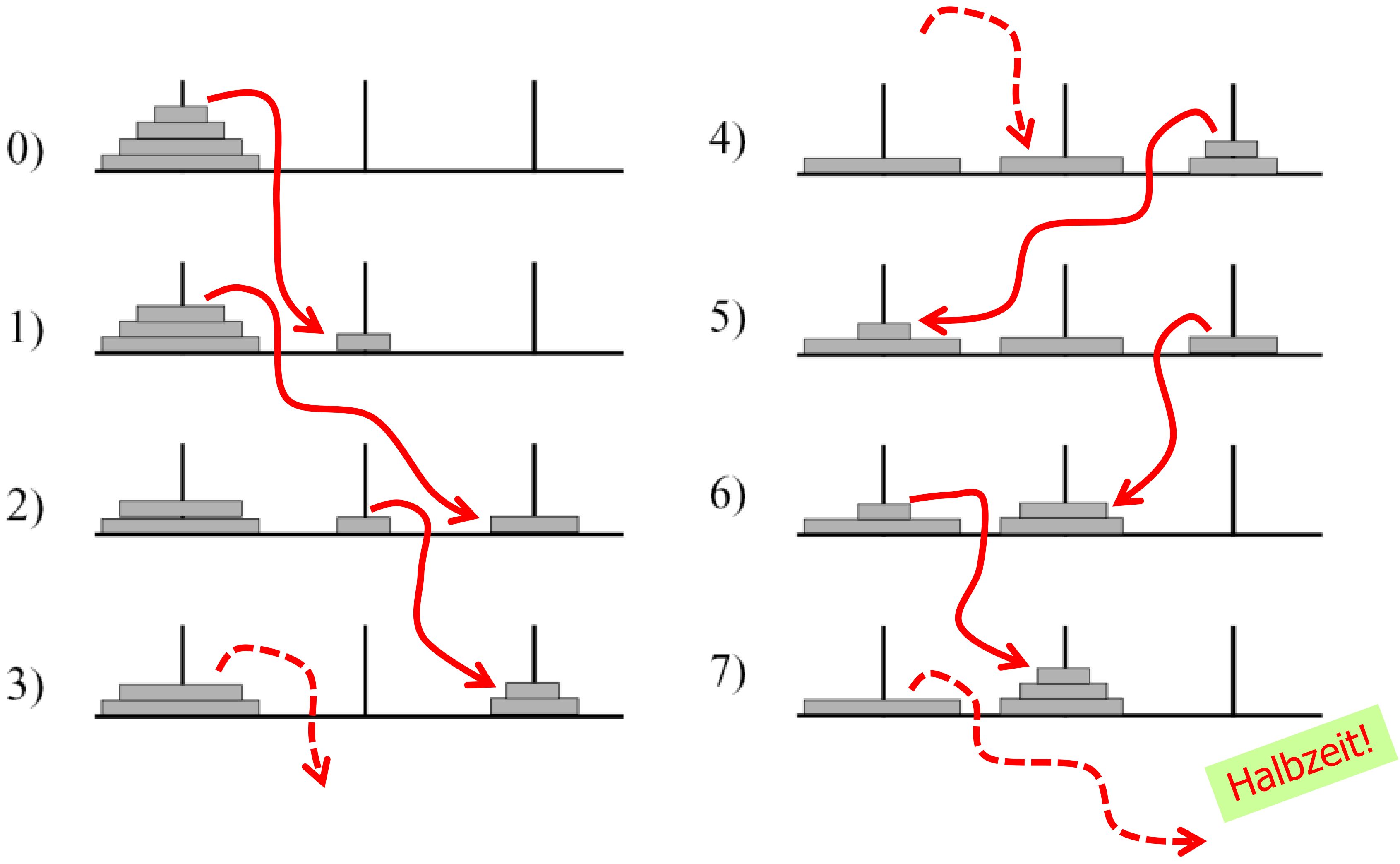
Mehr zum Märchen siehe https://de.wikipedia.org/wiki/Die_Bremer_Stadtmusikanten

Die Bremer Stadtmusikanten. Es hatte ein Mann einen Esel, der schon lange Jahre die Säcke unverdrossen zur Mühle getragen hatte, dessen Kräfte aber nun zu Ende giengen, so daß er zur Arbeit immer untauglicher ward. Da dachte der Herr daran, ihn aus dem Futter zu schaffen, aber der Esel merkte *daß kein guter Wind wehte*, lief fort und machte sich auf den Weg nach Bremen: dort, meinte er, könnte er ja Stadtmusikant werden. Als er ein Weilchen fortgegangen war, fand er einen Jagdhund auf dem Wege liegen, der *jappte wie einer, der sich müde gelaufen hat.* „Nun, was jappst du so, Packan?“ fragte der Esel. „Ach,“ sagte der Hund, „weil ich alt bin und jeden Tag schwächer werde, auch auf der Jagd nicht mehr fort kann, hat mich mein Herr wollen todt schlagen, da hab ich Reißaus genommen; aber womit soll ich nun mein Brot verdienen?“ „Weißt du was,“ sprach der Esel, „ich gehe nach Bremen und werde dort Stadtmusikant, geh mit und laß dich auch bei der Musik annehmen. Ich spiele die Laute, und du schlägst die Pauken.“ Der Hund wars zufrieden, und sie giengen weiter. Es dauerte nicht lange, so saß da eine Katze an dem Weg und machte *ein Gesicht wie drei Tage Regenwetter.* „Nun, was ist dir in die Quere gekommen, alter Bartputzer?“ sprach der Esel. „Wer kann da lustig sein, wenns einem *an den Kragen geht*,“ antwortete die Katze, „weil ich nun zu Jahren komme, meine Zähne stumpf werden, und ich lieber hinter dem Ofen sitze und spinne, als nach Mäusen herum jage, hat mich meine Frau ersäufen wollen; ich habe mich zwar noch fortgemacht, aber *nun ist guter Rath theuer*: wo soll ich hin?“ „Geh mit uns nach Bremen, du verstehst dich doch auf die Nachtmusik, da kannst du ein Stadtmusikant werden.“ Die Katze hielt das für gut und gieng mit. Darauf kamen die drei Landesflüchtigen an einem Hof vorbei, da saß auf dem Thor der Haushahn und schrie aus Leibeskräften. „Du schreist einem *durch Mark und Bein*,“ sprach der Esel, „was hast du vor?“ „Da hab ich gut Wetter prophezeit,“ sprach der Hahn, „weil unserer lieben Frauen Tag ist, wo sie dem Christkindlein die Hemdchen gewaschen hat und sie trocknen will; aber weil Morgen zum Sonntag Gäste kommen, so hat die Hausfrau doch kein Erbarmen, und hat der Köchin gesagt sie wollte mich Morgen in der Suppe essen, und da soll ich mir heut Abend den Kopf abschneiden lassen. Nun *schrei ich aus vollem Hals,* so lang ich noch kann.“ „Ei was, du Rothkopf,“ sagte der Esel, „zieh lieber mit uns fort, wir gehen nach Bremen, etwas besseres als den Tod findest du überall; du hast eine gute Stimme, und wenn wir zusammen musicieren, so muß es eine Art haben.“ Der Hahn ließ sich den Vorschlag gefallen, und sie giengen alle viere zusammen fort.

Sie konnten aber die Stadt Bremen in einem Tag nicht erreichen und kamen Abends in einen Wald, wo sie übernachten wollten. Der Esel und der Hund legten sich unter einen großen Baum, die Katze und der Hahn machten sich in die Äste, der Hahn aber flog bis in die Spitze, wo es am sichersten für ihn war. Ehe er einschlief, sah er sich noch einmal *nach allen vier Winden* um, da däuchte ihn er sähe in der Ferne ein Fünkchen brennen und rief seinen Gesellen zu es müßte nicht gar weit ein Haus sein, denn es scheine ein Licht. Sprach der Esel „so müssen wir uns aufmachen und noch hingehen, denn hier ist die Herberge schlecht.“ Der Hund meinte ein paar Knochen und etwas Fleisch dran, thäten ihm auch gut. Also machten sie sich auf den Weg nach der Gegend, wo das Licht war, und sahen es bald heller schimmern, und es ward immer größer, bis sie vor ein hell erleuchtetes Räuberhaus kamen. Der Esel, als der größte, näherte sich dem Fenster und schaute hinein. „Was siehst du, Grauschimmel?“ fragte der Hahn. „Was ich sehe?“ antwortete der Esel, „einen gedeckten Tisch mit schönem Essen und Trinken, und Räuber sitzen daran und lassens sich wohl sein.“ „Das wäre was für uns“ sprach der Hahn. „Ja, ja, ach, wären wir da!“ sagte der Esel. Da rathschlagten die Thiere wie sie es anfangen müßten, um die Räuber hinaus zu jagen und fanden endlich ein Mittel. **Der Esel mußte sich mit den Vorderfüßen auf das Fenster stellen, der Hund auf des Esels Rücken springen, die Katze auf den Hund klettern, und endlich flog der Hahn hinauf, und setzte sich der Katze auf den Kopf.** Wie das geschehen war, fiengen sie auf ein Zeichen insgesammt an ihre Musik zu machen: der Esel schrie, der Hund bellte, die Katze miaute und der Hahn krähte; dann stürzten sie durch das Fenster in die Stube hinein daß die Scheiben klirrten. Die Räuber fuhren bei dem entsetzlichen Geschrei in die Höhe, meinten nicht anders als ein Gespenst käme herein und flohen in größter Furcht in den Wald hinaus. Nun setzten sich die vier Gesellen an den Tisch, nahmen mit dem vorlieb, was übrig geblieben war, und *aßen als wenn sie vier Wochen hungern sollten*.

Wie die vier Spielleute fertig waren, löschten sie das Licht aus und suchten sich eine Schlafstätte, *jeder nach seiner Natur und Bequemlichkeit*. Der Esel legte sich auf den Mist, der Hund hinter die Thüre, die Katze auf den Herd bei die warme Asche, und der Hahn setzte sich auf den Hahnenbalken: und weil sie müde waren von ihrem langen Weg, schliefen sie auch bald ein. Als Mitternacht vorbei war, und die Räuber von weitem sahen daß kein Licht mehr im Haus brannte, auch alles ruhig schien, sprach der Hauptmann „wir hätten uns doch nicht sollen *ins Bockshorn jagen* lassen,“ und hieß einen hingehen und das Haus untersuchen. Der Abgeschickte fand alles still, gieng in die Küche, ein Licht anzuzünden, und weil er die glühenden, feurigen Augen der Katze für lebendige Kohlen ansah, hielt er ein Schwefelhölzchen daran, daß es Feuer fangen sollte. Aber die Katze verstand keinen Spaß, sprang ihm ins Gesicht, spie und kratzte. Da erschrack er gewaltig, lief und wollte zur Hinterthüre hinaus, aber der Hund, der da lag, sprang auf und biß ihn ins Bein: und als er über den Hof an dem Miste vorbei rannte, gab ihm der Esel noch einen tüchtigen Schlag mit dem Hinterfuß; der Hahn aber, der vom Lärmen aus dem Schlaf geweckt und munter geworden war, rief vom Balken herab „kikeriki!“ Da lief der Räuber, was er konnte, zu seinem Hauptmann zurück und sprach „ach, in dem Haus sitzt eine gräuliche Hexe, die hat mich angehaucht und mit ihren langen Fingern mir das Gesicht zerkratzt: und vor der Thüre steht ein Mann mit einem Messer, der hat mich ins Bein gestochen: und auf dem Hof liegt ein schwarzes Ungethüm, das hat mit einer Holzkeule auf mich losgeschlagen: und oben auf dem Dache, da sitzt der Richter, der rief bringt mir den Schelm her. Da machte ich daß ich fortkam.“ Von nun an getrauten sich die Räuber nicht weiter in das Haus, den vier Bremer Musikanten gefiels aber so wohl darin, daß sie nicht wieder heraus wollten. Und der das zuletzt erzählt hat, *dem ist der Mund noch warm.*

Der Ablauf bei hanoi(4,1,3)



Die dynamische Aufrufkette

Schnappschuss des Berechnungszustandes

hanoi(4,1,3)

```
if … bewege …
hanoi(3,1,2)
bewege(1,3)
hanoi(3,2,3)
```

hanoi(3,2,3)

```
if … bewege …
hanoi(2,2,1)
bewege(2,3)
hanoi(3,1,3)
```

hanoi(2,2,1)

```
if … bewege …
hanoi(1,2,3)
bewege(2,1)
hanoi(1,3,1)
```

hanoi(1,2,3)

```
bewege(2,3)
hanoi(0,2,1)
bewege(2,3)
hanoi(0,1,3)
```

bewege(2,3)

```
println(2,3)
```

Tiefe der dynamischen Aufrufkette

```
void hanoi(int groesse, int von, int nach) {
  if (groesse == 1)
    bewege(von, nach);
  else {
    hanoi(groesse-1, von, 6-von-nach);
    bewege(von, nach);
    hanoi(groesse-1, 6-von-nach, nach);
  }
}
```

- Es existieren gleichzeitig mehrere Instanzen der hanoi-Methode
- Jede wartet auf Fertigstellung der von ihr gerufenen Methode an der Aufrufstelle
- Mit der eigenen Instanz wird fortgefahren, wenn die Aufrufkette wieder bis dorthin abgebaut worden ist

Zeitkomplexität des Hanoi-Algorithmus

- Nach der Legende ist das $\boxed{\text{Weltende}}$ erreicht, wenn die Mönche ihre Aufgabe gelöst haben
- Wann ist das bei unserem Algorithmus der Fall?
  - n = 64 Scheiben
  - Eine Scheibe / Minute
- „Anfangsverdacht" für die Zeitkomplexität (= Anzahl der Elementarschritte, hier: „bewege") empirisch:

n	t(n)
1	1
2	3
3	7
4	15
:	:

- Vermutung: $\boxed{t(n) = 2^n - 1}$
- Rekursionsformel: $t(n) = \begin{cases} 1 & \text{für } n = 1 \\ 2\ t(n-1) + 1 & \text{sonst} \end{cases}$
- Korrektheit der Rekursionsformel ist klar: t(n-1) in der ersten Halbzeit, nochmal t(n-1) in der zweiten und 1 dazwischen
- Zur Korrektheit der geschlossenen Formel 2^n-1 zeige induktiv: 2^n-1 erfüllt die Rekursionsformel

Zum Weltuntergang

- Da haben wir noch einmal Glück gehabt!
  - $t(64) = 2^{64}-1$ min $\approx 1.8 \times 10^{19}$ min $\approx 1.28 \times 10^{16}$ Tage $\approx 3.5 \times 10^{13}$ Jahre
  - Die Welt existiert aber erst seit ca. 1.38×10^{10} Jahren
  - Der Weltuntergang steht also noch nicht unmittelbar bevor...

- Algorithmen exponentieller Zeitkomplexität $t(n)= c^n$ lassen sich in der Praxis oft selbst für „vernünftige“ Problemgrössen n auch auf sehr schnellen Computern kaum (jemals!) ausführen
  - Sie sind inhärent ineffizient

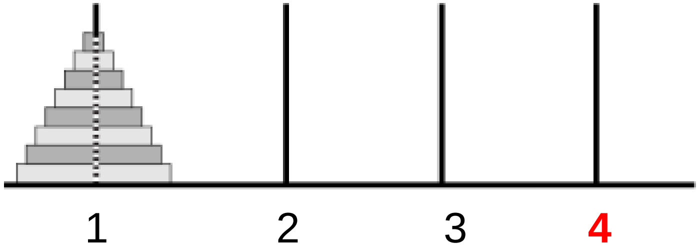
Effizientere und einfachere Algorithmen für das Problem?

- Sind die Mönche vielleicht schlecht beraten?
- D.h.: gibt es einen effizienteren Algorithmus (der die geforderten Nebenbedingungen des Problems einhält)?
  - Ja? (→ Algorithmus angeben!)
  - Nein? (→ Beweis, dass nicht!)
  - Oder: weiss vielleicht niemand, ob „ja“ oder „nein“ gilt?
- Noch eine Frage: Gibt es einen (einfachen) nicht-rekursiven Algorithmus für das Problem?
  - Antwort: Ja, aber dieser wird hier nicht verraten

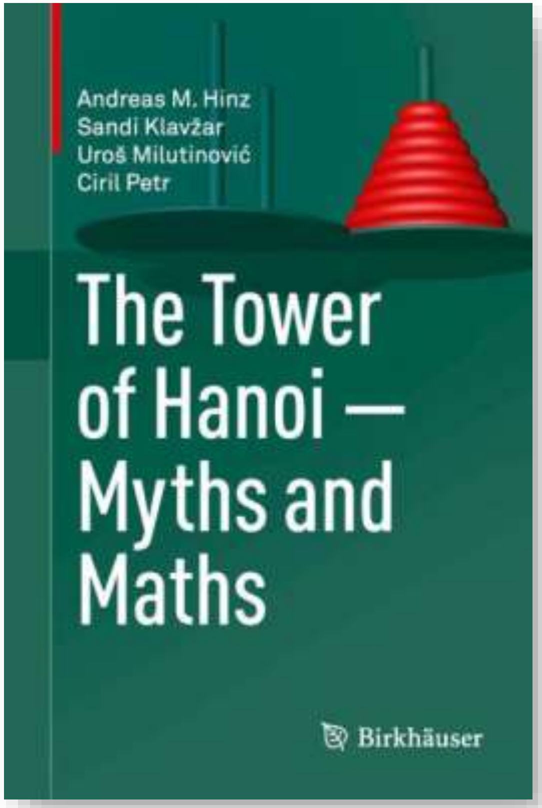
May the 4th be with you!

- Angenommen, man hätte einen (anfangs leeren) weiteren Turm als Zwischenspeicher – wie schnell (und nach welchem Algorithmus!) kann man dann die 64 Scheiben umstapeln? (Tipp: 18433 Züge scheinen jedenfalls zu genügen.)

n	t(n)
1	1
2	3
3	5
4	9
5	13
6	17
7	25
8	33
...	...
25	577
...	...
30	1025
...	...



- Zu den Türmen von Hanoi gibt es sogar ganze Bücher:



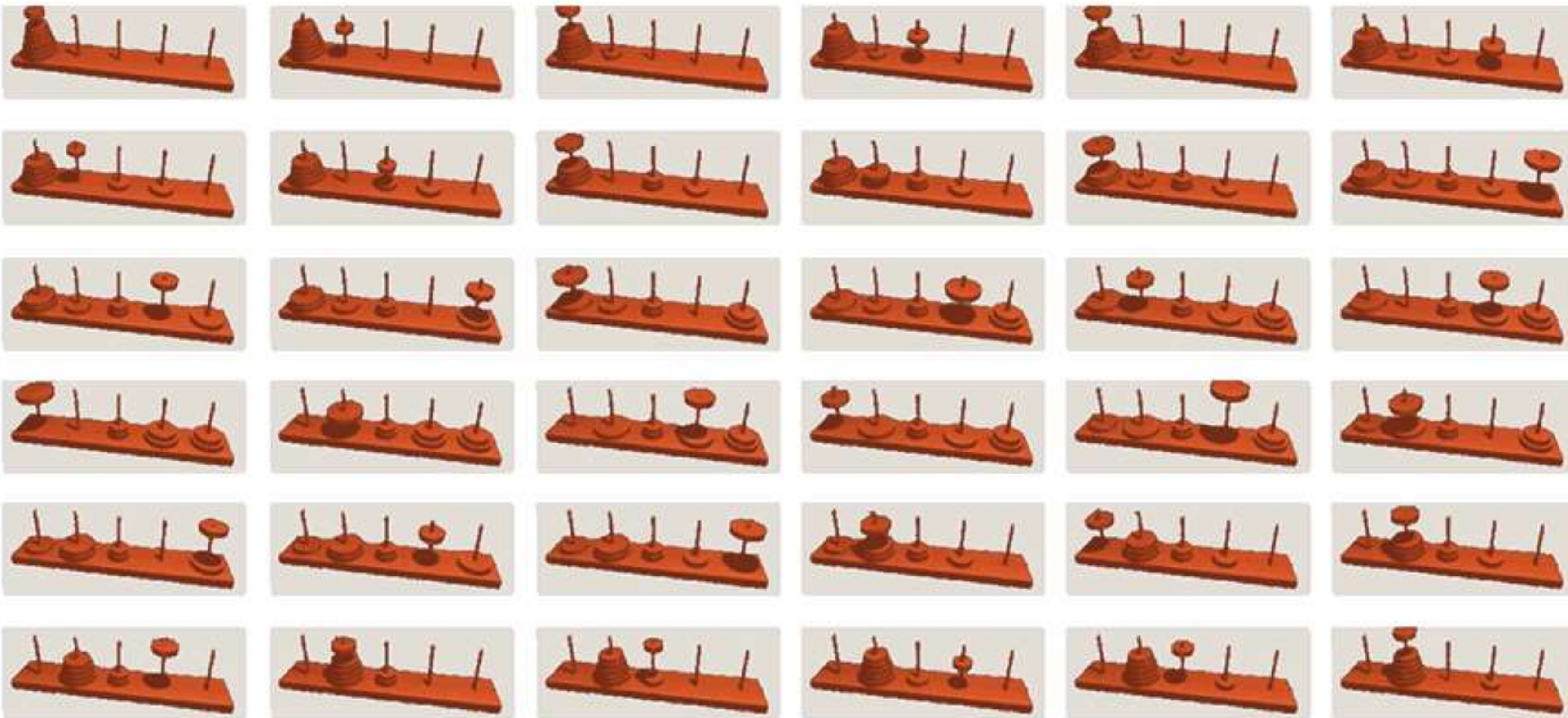
5 Türme, 7 Scheiben – May the Force be with you!

1 → 2	4 → 5	5 → 4
1 → 3	1 → 4	5 → 2
1 → 4	2 → 4	1 → 2
2 → 3	1 → 2	4 → 2
1 → 2	4 → 1	3 → 4
1 → 5	4 → 2	3 → 2
		4 → 2

Algorithmus durch Zuschauen lernen?

Animation: http://towersofhanoi.info/Animate.aspx

5 Türme, 7 Scheiben – May the Force be with you!



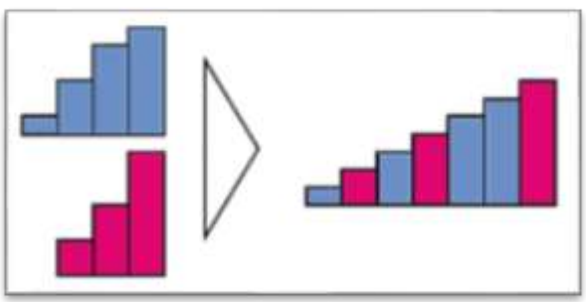
1 → 2; 1 → 3; 1 → 4; 2 → 3; 1 → 2; 1 → 5; 4 → 5; 1 → 4; 2 → 4; 1 → 2; 4 → 1; 4 → 2; 5 → 4; 5 → 2; 1 → 2; 4 → 2; 3 → 4; 3 → 2; 4 → 2

Mergesort – ein Beispiel für „divide et impera"

[„Sortieren durch Verschmelzen"]

„Merge" = einfädeln, verflechten, verschmelzen

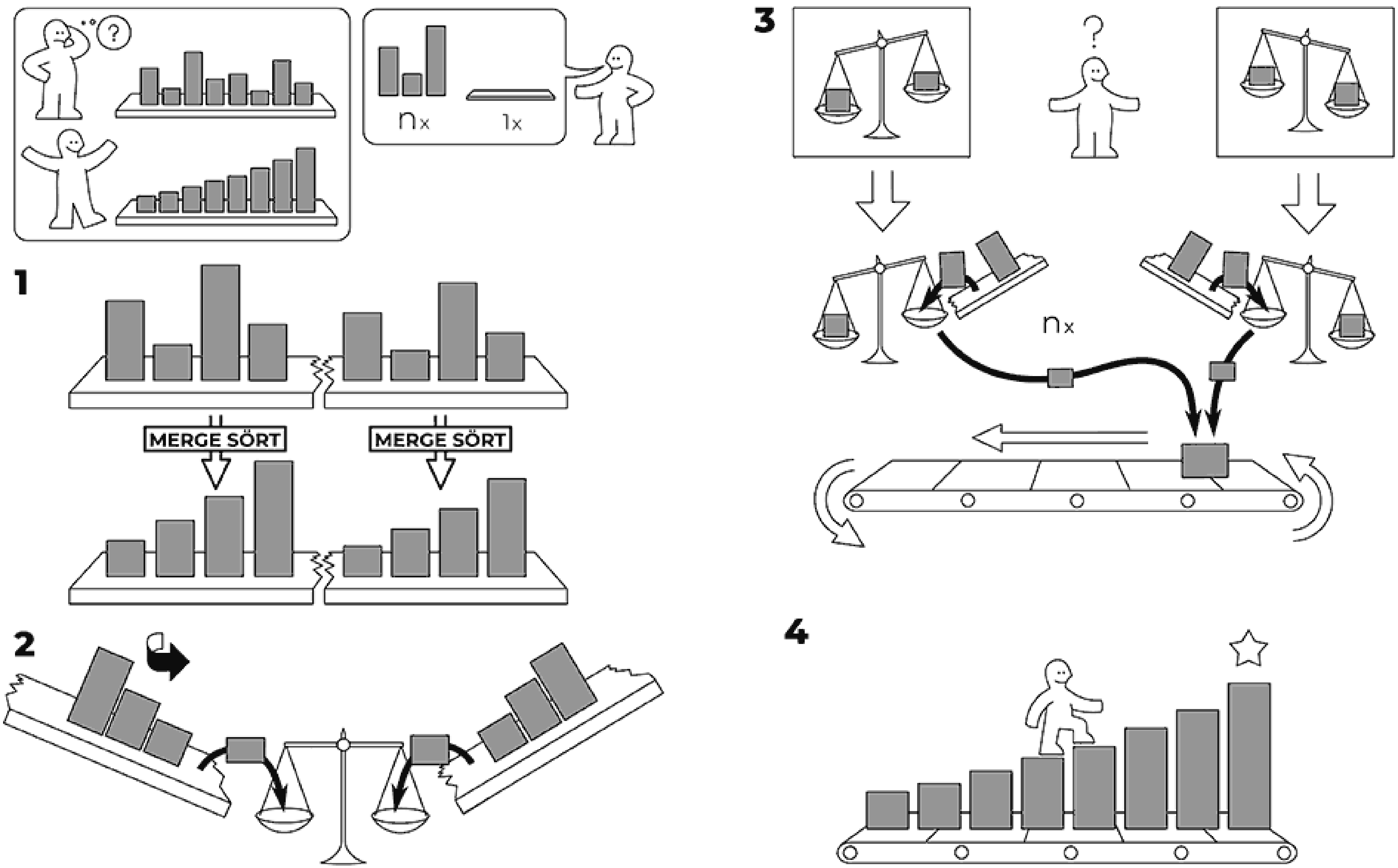
→ Kombination von zwei sortierten Teilfolgen zu einer einzigen sortierten Folge durch fortgesetzte Auswahl des kleineren der beiden Anfangselemente



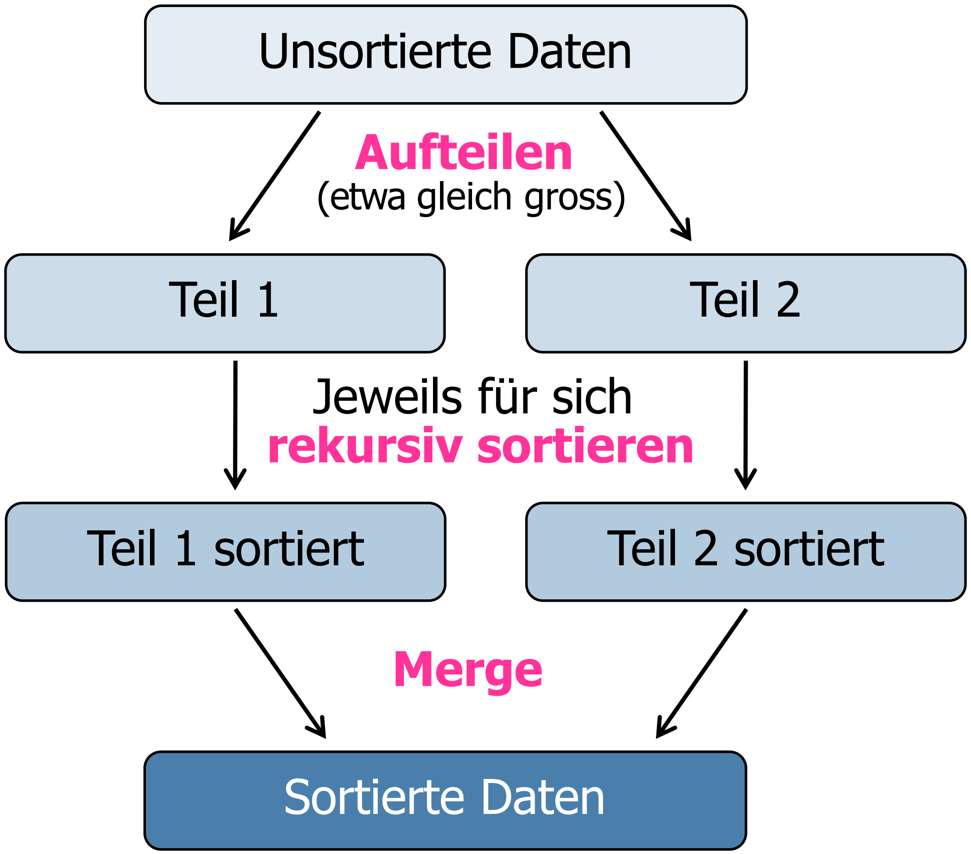
Mergesort-Erfinder John von Neumann sprach zunächst von „meshing" (vermaschen, verschränken, ineinandergreifen) statt „merging". Das Wort „mesh" ist über das Germanische „mask" mit dem Deutschen Wort „Masche" verbunden; weder „merge" noch „mesh" hängen aber mit „to mix" bzw. „mischen" zusammen; letztere stammen vom Lateinischen „miscere" (vermischen) bzw. „mixtum".

MERGE SÖRT

idea-instructions.com/merge-sort/
v1.1, CC by-nc-sa 4.0



Mergesort – Prinzip (divide et impera)



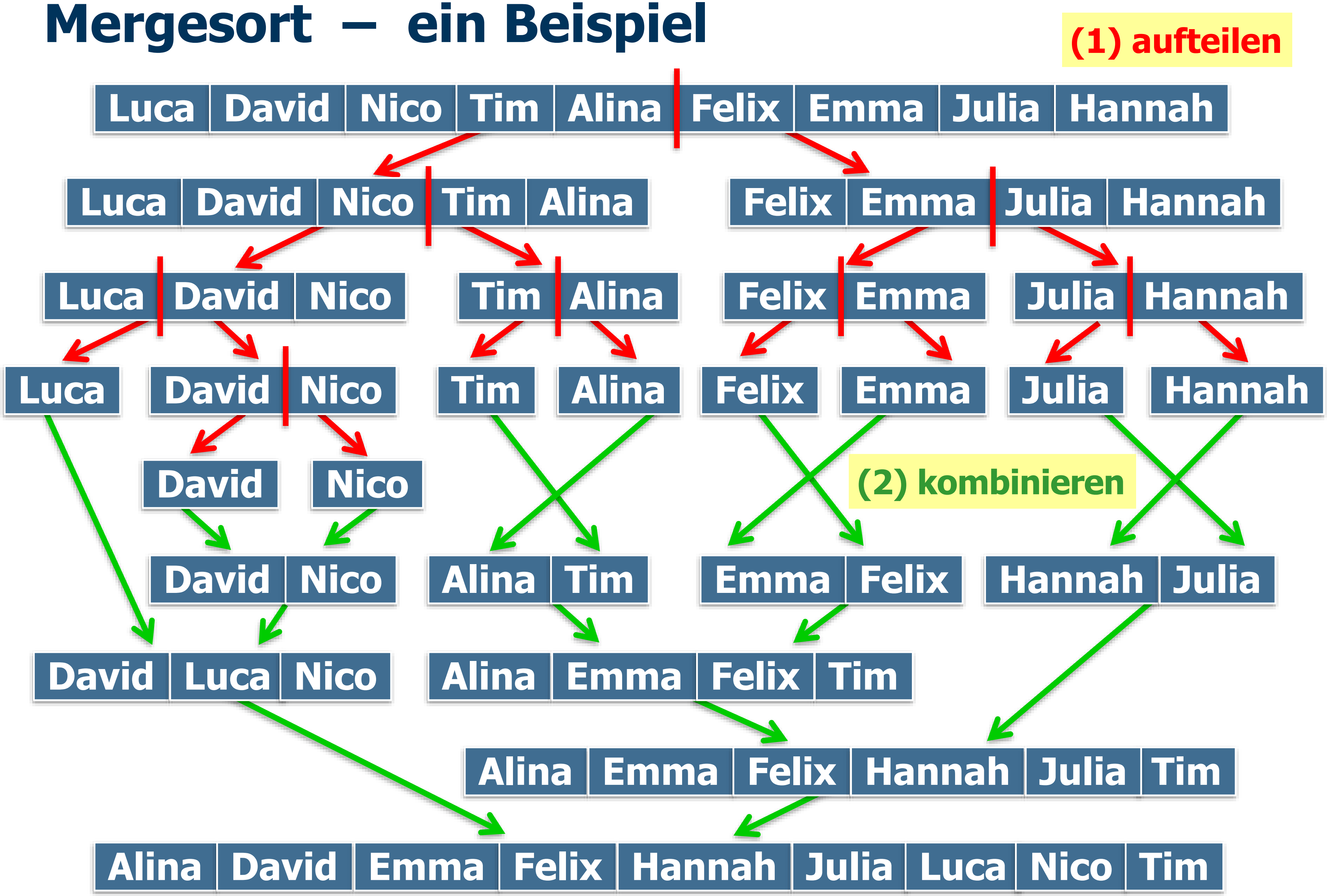
Muss nicht exakt halbiert werden

1) Erste Hälfte der Daten rekursiv mit Mergesort sortieren
2) Zweite Hälfte der Daten rekursiv mit Mergesort sortieren
3) Mergen („kombinieren") der beiden sortierten Hälften

→ Folgen der Länge 1 nicht sortieren (Rekursionsabbruch)

Das Mergesort-Verfahren wurde 1945 von John von Neumann vorgestellt; es eignet sich auch dann gut, wenn die Daten auf (nur sequentiell zugreifbarem) Magnetband vorliegen und nicht alle Daten in den Hauptspeicher geladen werden können; es wurde früher sogar in mechanischen Sortiergeräten eingesetzt.

Mergesort – ein Beispiel



Ors eirt, beis eilp!

SORTIERBEISPIEL

SORTIER BEISPIEL

SOR TIER BEIS PIEL

S OR TI ER BE IS PI EL

S OR IT ER BE IS IP EL

ORS EIRT BEIS EILP

EIORRST BEEIILPS

BEEEIIILOPRRSST

Hızlı Sıralama günümüzde yaygın olarak kullanılan bir sıralama algoritmasıdır. [...] Aynı işlemleri sağdaki ve soldaki bölümlere ayrı ayrı yapılır. Sonuç şöyle : B E E E I I I L O P R R S S T [Vikipedi : https://tr.wikipedia.org/wiki/H%C4%B1zl%C4%B1_s%C4%B1ralama]

Mergesort – rekursiver Ansatz

Der rekursive top-down Mergesort-Algorithmus in Java:

```
void mergesort(int li, re) {
  ...
  if (...) {
   m = (li+re)/2;
   mergesort(li,m);
   mergesort(m+1,re);
   merge(li,m,re)
  ...
```

li, re und m stellen hier Indizes eines Array dar

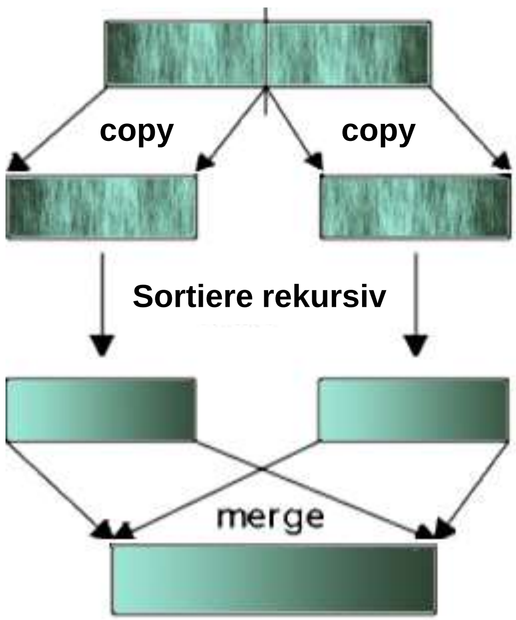
Denkübung: ist die Bestimmung von m korrekt, auch wenn li+re eine ungerade Zahl ergibt?

Mergesort vs. Quicksort

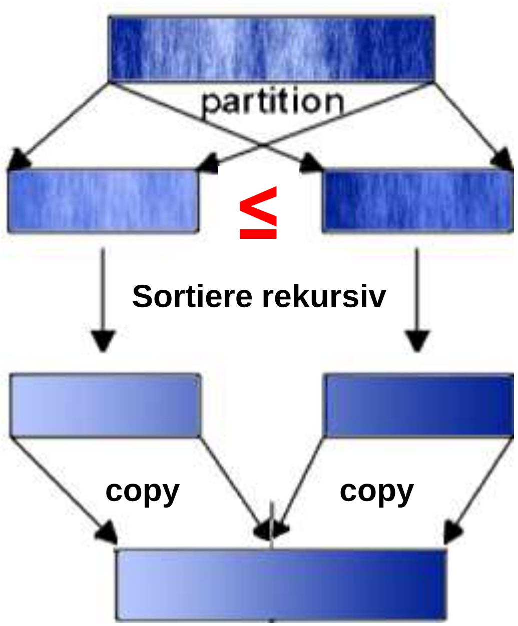
"I think Quicksort is the only really interesting algorithm that I've ever developed" – Tony Hoare

- Quicksort beruht ebenfalls auf dem Divide-et-impera-Prinzip
- Statt einer nachgelagerten Merge-Phase nutzt Quicksort allerdings eine vorgelagerte Partition-Phase: Danach sind alle Elemente der linken „Hälfte" kleiner oder gleich denen der rechten

Mergesort



Quicksort



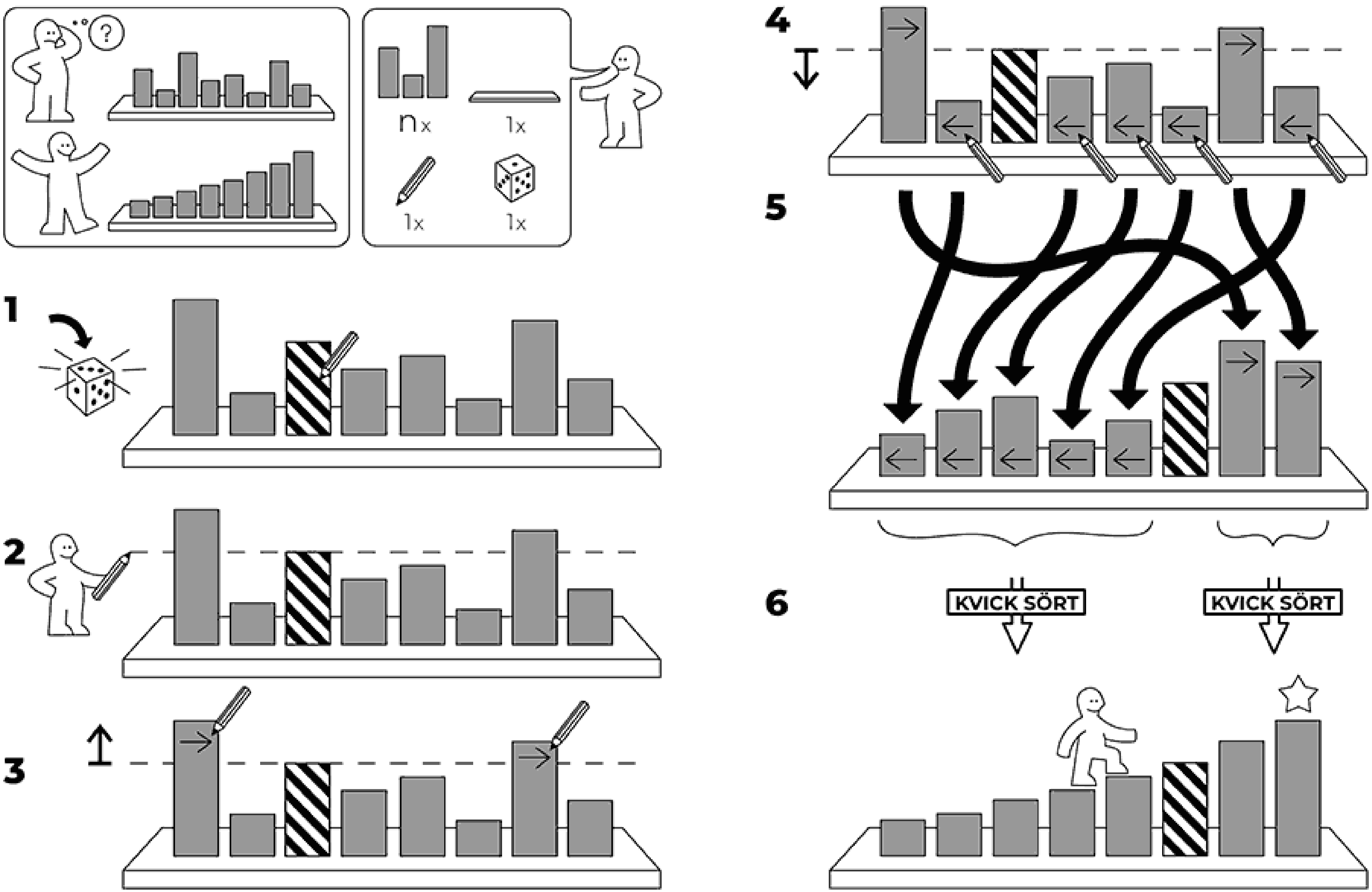
Die Bestimmung eines geeigneten „Pivot-Elementes" bei Quicksort, das die Grenze der beiden Partitionen ausmacht, ist etwas „tricky"; wir behandeln das nicht näher.

Der (seltene) worst case verursacht quadratischen Aufwand; im Mittel hat Quicksort allerdings (genau wie Mergesort) nur einen Aufwand proportional zu n log n.

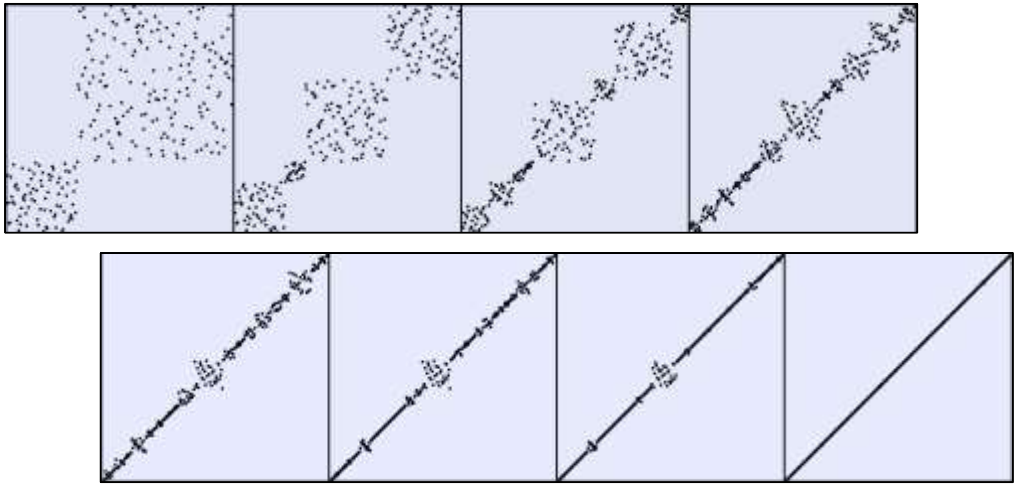
KVICK SÖRT

idea-instructions.com/quick-sort/
v1.1, CC by-nc-sa 4.0

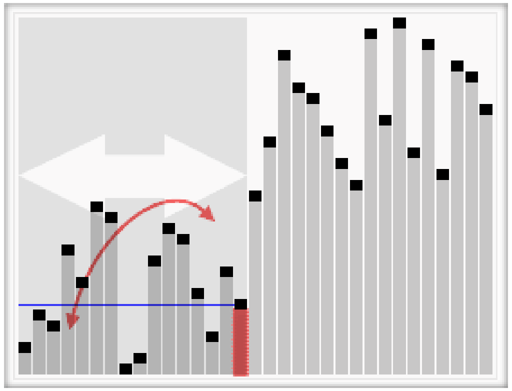
IDEA



Quicksort Basics
[Wikipedia]



- Bild: Acht Schnappschüsse; die Zahlen von 1 bis n (in zufälliger Ausgangsreihen-folge) werden quicksortiert
- Quicksort ist für ein breites Spektrum von praktischen Anwendungen der bevorzugte Sortieralgorithmus, weil er schnell ist und, sofern Rekursion zur Verfügung steht, einfach zu implementieren ist
- Quicksort ist ein In-Place-Verfahren: Es vertauscht die Elemente der zu sortierenden Liste nur innerhalb der Liste und kopiert sie nicht in zusätz-lichen Speicherplatz (es benötigt allerdings für jede Rekursionsebene zusätzlichen Platz auf dem Stack)
- Bei de.wikipedia.org/wiki/Quicksort kann diese → Animation studiert werden
  - *Eine zufällige Permutation von Integerwerten wird mit Quicksort sortiert; die blauen Linien zeigen den Wert des rot markierten Pivotelements im jeweiligen Rekursionsschritt*



Historische Notiz

Quicksort-Erfinder Tony Hoare (* 1934)

"National Physical Laboratory in Teddington […] offered me a post as a Senior Scientific Officer to work on a project just started for automatic translation from scientific Russian into English. They were writing a translation program in machine code for the Pilot ACE computer. […] The first problem of machine translation that interested me was how to sort the words of each incoming Russian sentence into ascending order. Recall that the main stores of computers of those days were large enough to hold one sentence, but certainly not large enough to hold a dictionary. The dictionary was held on magnetic tape, so to look up a single Russian word would require a spin through the magnetic tape, on average covering half its length. It was clearly a better idea to spin through the tape only once for each whole sentence. This required you to sort the words in the sentence into alphabetical order before you start the tape spinning. I was thinking about this in my little room in Moscow State University, and I tried to write down my thoughts in Mercury Autocode, which I had learnt the previous year at Oxford while I was studying statistics. My first idea I rejected as too slow. It was rapidly followed by my second idea, which later when I got back to England, I wrote up as my second ever published scientific article. Since then, it has become famous under the title «Quicksort»."

1960: Tony Hoare mit 26 in Moskau

Tony Hoare: Quicksort, Algol und Rekursion

Anlässlich seiner Verleihung des Turing Awards hielt Tony Hoare im Oktober 1980 eine Rede ("The Emperor's Old Clothes") – hier einige kurze Auszüge daraus:

I start my story in August 1960, when I became a programmer with a small computer manufacturer, a division of Elliott Brothers (London) Ltd. […]. My first task was to implement for the new Elliott 803 computer, a library subroutine for a new fast method of internal sorting just invented by D L Shell […]. My boss and tutor, Pat Shackleton, was very pleased with my completed program. I then said timidly that I thought I had invented a sorting method that would usually run faster than Shellsort, without taking much extra store. He bet me sixpence that I had not. Although my method was very difficult to explain, he finally agreed that I had won my bet.

I wrote several other tightly coded library subroutines but after six months I was given a much more important task – that of designing a new advanced high level programming language for the company's next computer, the Elliott 503, which was to have the same instruction code as the existing 803 but run sixty times faster […]. By great good fortune there came into my hands a copy of the Report on the International Algorithmic Language ALGOL 60. Of course, this language was obviously too complicated for our customers. How could they ever understand all those begins and ends when even our salesmen couldn't?

Around Easter 1961, a course on ALGOL 60 was offered in Brighton, England, with Peter Naur, Edsger Dijkstra, and Peter Landin as tutors. […] It was there that I first learned about recursive procedures and saw how to program the sorting method which I had earlier found such difficulty in explaining. It was there that I wrote the procedure, immodestly named QUICKSORT […]. After the ALGOL course in Brighton, Roger Cook was driving me and my colleagues back to London when he suddenly asked, "Instead of designing a new language, why don't we just implement ALGOL 60?" We all instantly agreed – in retrospect, a very lucky decision for me.

Tony Hoare – Oral History

Nachfolgend einige kurze Auszüge aus der „Oral History of Sir Antony Hoare", produziert 2006 vom Computer History Museum. Das gesamte 24 Seiten umfassende Interview kann man hier nachlesen: http://archive.computerhistory.org/resources/text/Oral_History/Hoare_Sir_Antony/102658017.05.01.pdf

What was the first program you ever wrote?

Hoare: Back in 1958 I attended a course in Mercury Autocode, which was the programming language used on a computer that the University [Oxford] was just purchasing from Ferranti. I wrote a program. I chose my own exercise program. It was a solution of two person game, using a technique which I found in a book on game theory by von Neumann and Morgenstern. It did, I think – well, I don't know whether it worked or not. It certainly ran to the end, but I forgot to put in any check on whether the answers it produced was correct, and the calculations were too difficult for me to do by hand afterwards.

What was programming like in those days?

Hoare: Very different from today. The programs were all prepared on punched cards or paper tape. It might take a day even to get them punched up from the coding sheets, and then submitted to a computer again, probably in a batch, maybe the following day. It would take a long time if there were any faults in the program, to find out where they were.

You just submitted a hand-written program and somebody typed it up for you?

Hoare: That's right, yes. Yes. We weren't so good at typing in those days.

And the back of all this, you were developing interesting algorithms, like the Quicksort algorithm that everyone knows now. When did that fit in?

Hoare: Well, that was so. I think Quicksort is the only really interesting algorithm that I ever developed, and I had already developed that when I was studying at Moscow State University. I got a letter from the National Physical Laboratory, which at that time was just starting a project for the automatic translation of Russian into English. They heard of me somehow indirectly, and so they wrote to me offering me a job as Senior Scientific Officer to work on this project. I thought I might as well find out what's going on in this area, and so I looked up the Russian literature on the subject. I met several of the people in Moscow who were working on the machine translation. I wrote my first published article in Russian, in a journal called "The Machine Translation". That certainly excited my interest in the topic of computers and languages. [...] In those days, the dictionary, in which you had to look up in order to translate from Russian to English, was stored on a long magnetic tape, and it took several minutes for the tape to be read from the beginning to the end. Well, this dictionary was stored in alphabetical order, starting with A and ending with Z in English, and therefore it paid to sort the words of the sentence into the same alphabetical order before consulting the dictionary so that you could look up all the words in the sentence on a single pass of the magnetic tape. You didn't have to rewind the tape in order to look up the next word, because it was already in the right alphabetic order.

But there were existing sorting algorithms. Were they already using those?

Hoare: Oh, I didn't know anything about what existed in those days. I thought with my knowledge of Mercury Autocode, I'll be able to think up how I would conduct this preliminary sort. After a few moments, I thought of the obvious algorithm, which is now called bubble sort, and rejected that, because that was obviously rather slow, and thought of Quicksort as the second thing I thought of. It didn't occur to me that this was anything very difficult. It was all an interesting exercise in programming.

In 1968, you wrote a very important paper on the axiomatic approach to computer programming, which has since become known as "Hoare Logic". Would you like to just say something about what led up to that?

Yes, I was interested, as indeed many people were at that time, in making good the perceived deficiency of the Algol report, that while the syntax was extremely carefully and formally defined, the semantics was left a little vaguer. [...] The idea that I put forward, based on the ideas of mathematical logic which I studied at university, was that I put forward a set of axioms which describe the properties of the implementation without describing exactly how it worked. It would be possible, I hoped, to state those properties sufficiently precisely that programmers would be able to write programs using only those properties, and leave the implementers the freedom to implement the language in different ways, but at the same time taking responsibility for the fact that their implementation actually satisfied the properties that the program was relying on.

[...] your work on communicating sequential processes?

Hoare: The idea of a communicating process is that instead of calling its components as subroutine, or a method, you'd actually communicate the values that you wanted to transmit to it by some input or output channel, and it would communicate its results back again by a similar medium. The reason for this, as I say, was a technological advance in the hardware: the advent of the microprocessor. The microprocessor at that time was a cheap and small machine, with not very much store, but in principle capable of communicating with other microprocessors of a similar nature along wires. I thought some simple metal wires, which would communicate signals from the pin one of these machines to the pin of another. It was easy to predict that the fastest and cheapest way of making a large and fast computer would be to assemble a large number of very cheap microprocessors together, and allow them to co-operate with each other on a single task by communicating along wires that connected them. For this, a new method, I should say, discipline of programming, a new paradigm, a new architecture of programs would be appropriate. And perhaps a new language for expressing the programs. That was how the communicating sequential processes came to take a leading role in my subsequent research.

Is there anything you'd like to say about your teaching?

Hoare: Well, I right from the beginning took the view that computer science, and in particular computer programming, was a very good kind of education in training the mind in habits of rigorous thinking. A deep concentration is really needed in order to write a program that actually works, and that perhaps it might play the same sort of role in education that the study of the classical languages, Latin and Greek used to play in previous centuries, and unfortunately no longer do. So I'd always hoped that programming and computer science would be regarded as subjects fit for education; interesting and challenging even for people who weren't going to eventually apply the knowledge professionally.

Mergesort parallelisiert

Wir gehen in dieser Vorlesung nicht auf das Java-Paket „concurrent" ein; Nachfolgendes dient nur der Illustration, wie man damit im Prinzip Mergesort parallelisieren kann:

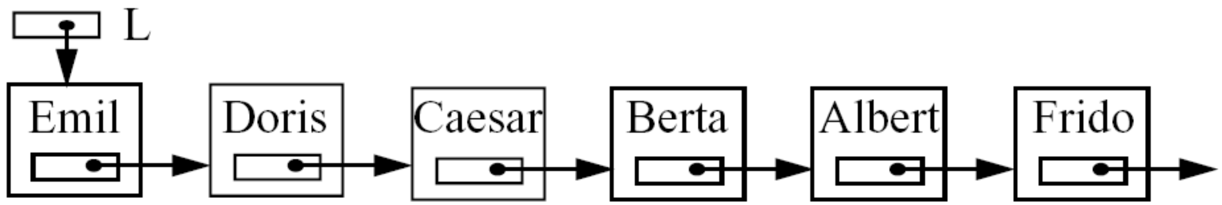
```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;

class SortTask extends RecursiveAction {…
void compute(…) { …
  if … // evtl. Rekursionsabbruch
  { invokeAll(new SortTask(li, m),
              new SortTask(m+1, re));
    merge(li, re);
…
ForkJoinPool pool = new ForkJoinPool();
pool.invoke(new SortTask(0, data.length));
…
```

Wenn der Bereich [li, re] zu klein ist, dann sortiert man lieber nicht parallel, sondern sequentiell (und evtl. auch nicht rekursiv)

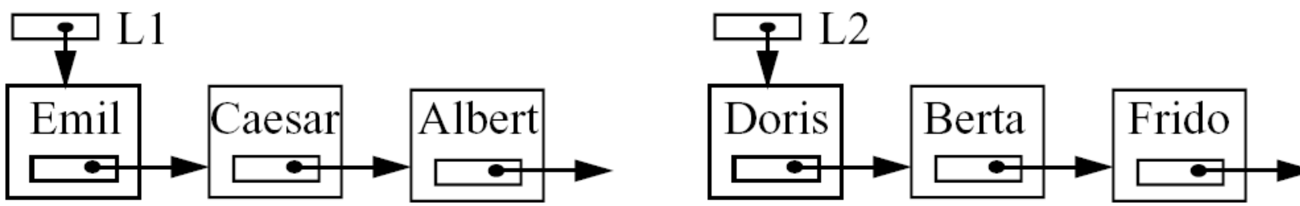
Merge wird hier aber nicht auch noch parallelisiert, so dass alleine auf oberster Ebene dafür n Schritte nötig sind

"This algorithm is a trivial modification from the serial version, but its speedup is not impressive … A parallel merge algorithm to not only parallelize the recursive division of the array, but also the merge operation leads to a better parallel sort…" [Wikipedia]

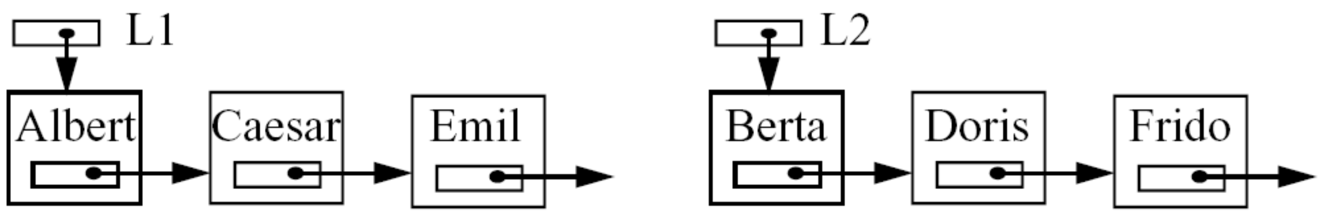
Mergesort mit verketteten Listen



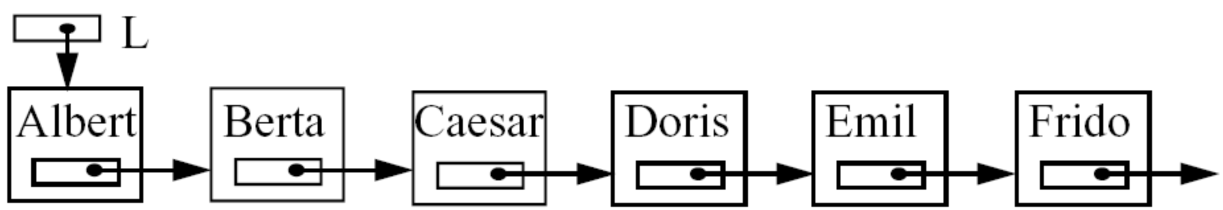
Sofern die Liste mehr als ein Element enthält:
→ Aufspalten der Liste in zwei Teillisten (Durchlaufen und abwechselnd in L1 oder L2 einfügen):



Die Listen L1 und L2 jeweils rekursiv sortiert:



L1 und L2 durchlaufen und das jeweils kleinere Objekt in eine neu aufgebaute Liste hinten anfügen („merge“):



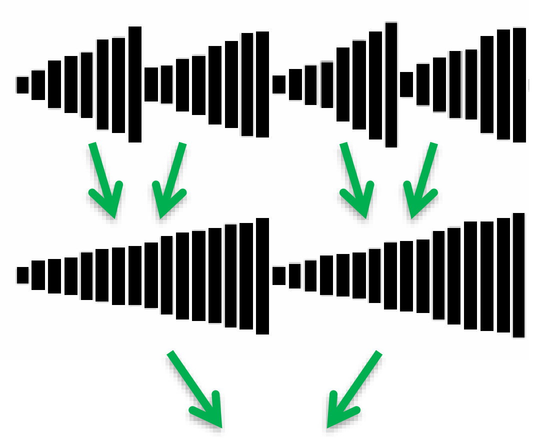
Nicht-rekursive Variante: Bottom-up-Mergesort

1) Durchlaufe die Folge der Elemente und erzeuge geordnete Paare durch Mergen benachbarter Elemente („Teilfolgen der Länge 1")
   - Es werden sortierte Teilfolgen der Länge 2 erzeugt

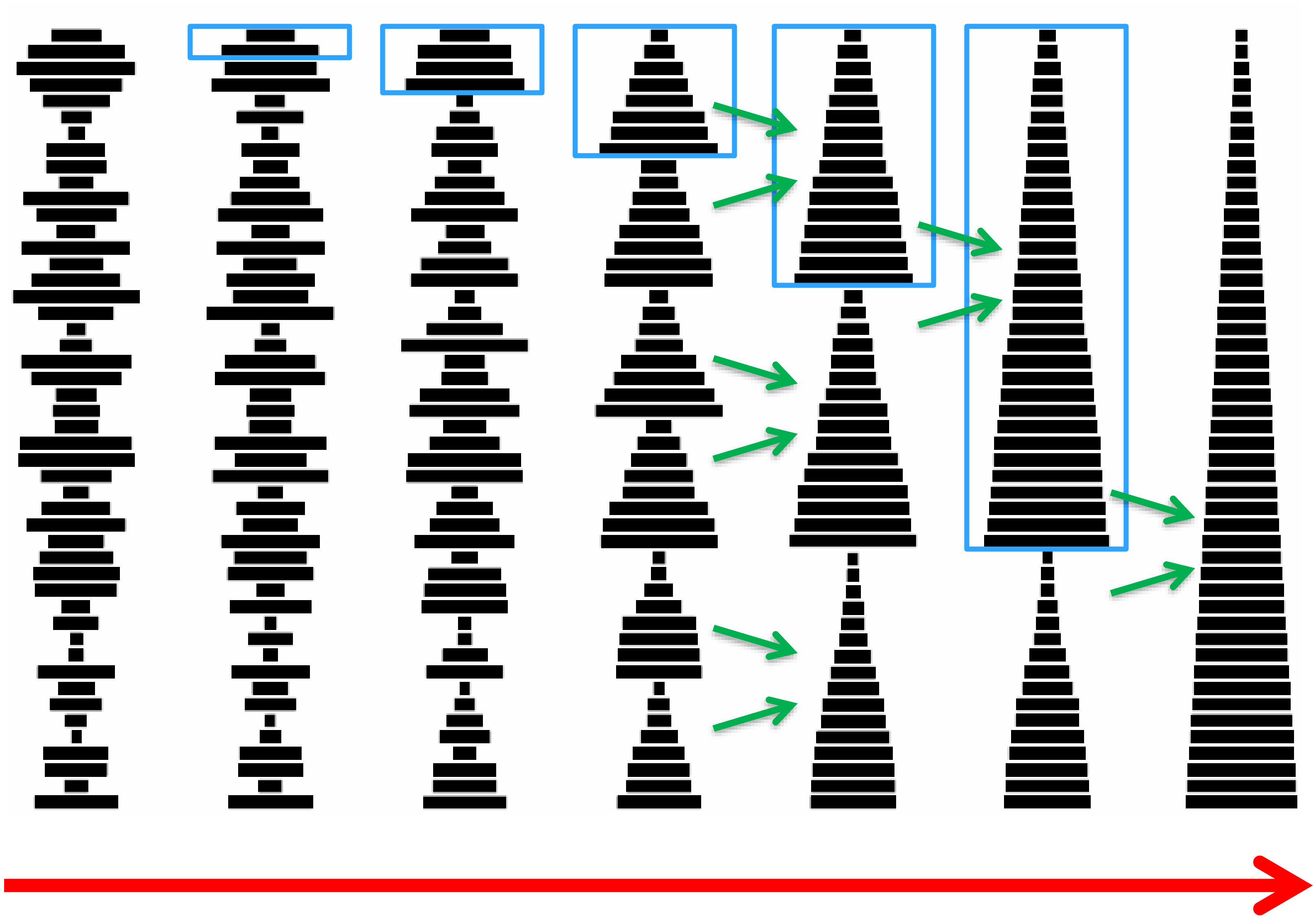
2) Durchlaufe die Folge erneut und erzeuge geordnete Quadrupel durch Mergen benachbarter sortierter Teilfolgen der Länge 2
   - Es werden sortierte Teilfolgen der Länge 4 erzeugt

3) …

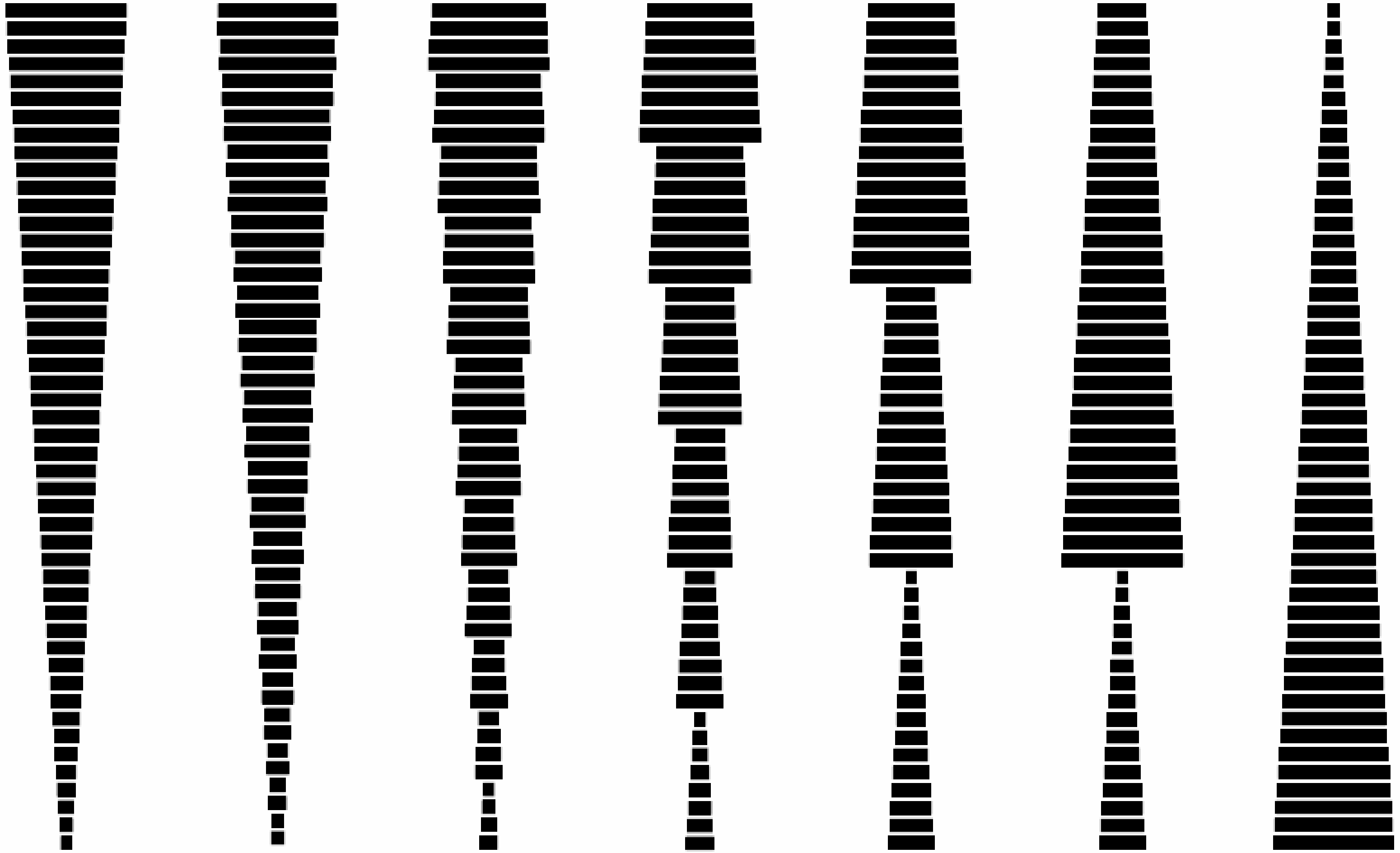
→ Bis eine sortierte Folge der Länge n erzeugt wird



Sortieren einer zufälligen Folge



Umgekehrt sortierte Ausgangsfolge (gemein!)



Ebenfalls nur 6 Durchläufe durch die Folge der 48 Elemente

Bottom-up-Mergesort – Effizienz

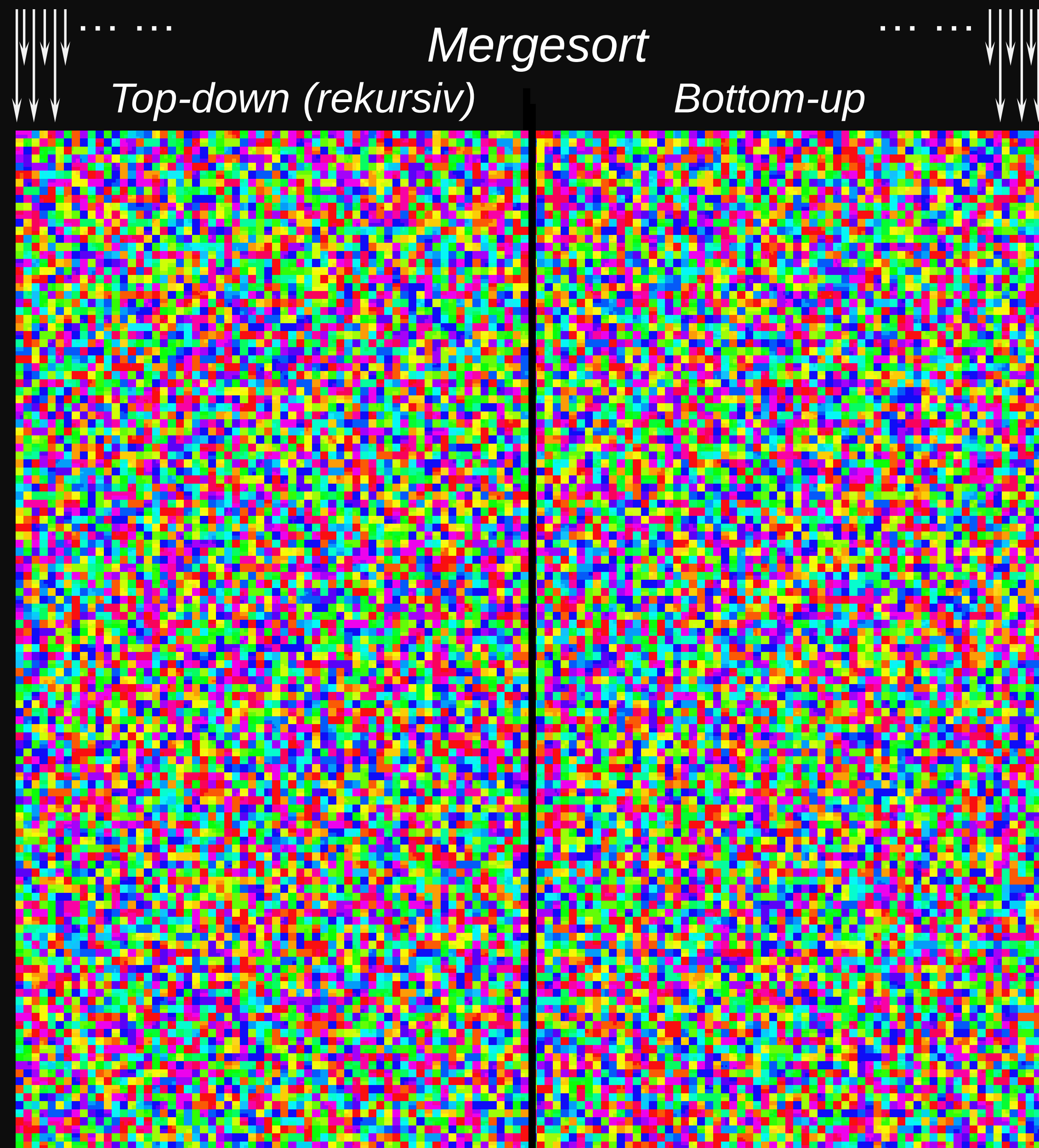
- Nach k Durchläufen hat man sortierte Teillisten der Länge 2^k
- → Nach $\log n$ Durchläufen hat man die Länge n (→ fertig!)
- Ein Durchlauf erfordert jeweils n „Rechenschritte"
  - Pro Schritt: Vergleichen zweier Werte, Erhöhen eines Index...
- Mergesort benötigt also (grössenordnungsmässig) $n \log n$ „Schritte"
  - Unabhängig von den konkreten Daten
- Top-down- und Bottom-up-Mergesort führen i.w. die gleichen Merge-Operationen aus, allerdings in unterschiedlicher Reihenfolge!

- Beim rekursiven Top-down-Mergesort gibt es folgenden Zwischenzustand: | sortiert | unsortiert |
  - Dieser Zustand kann beim Bottom-up-Mergesort nicht auftreten, dort sind alle Teile stets etwa „gleich weit"

Bottom-up-Mergesort – Effizienz (2)

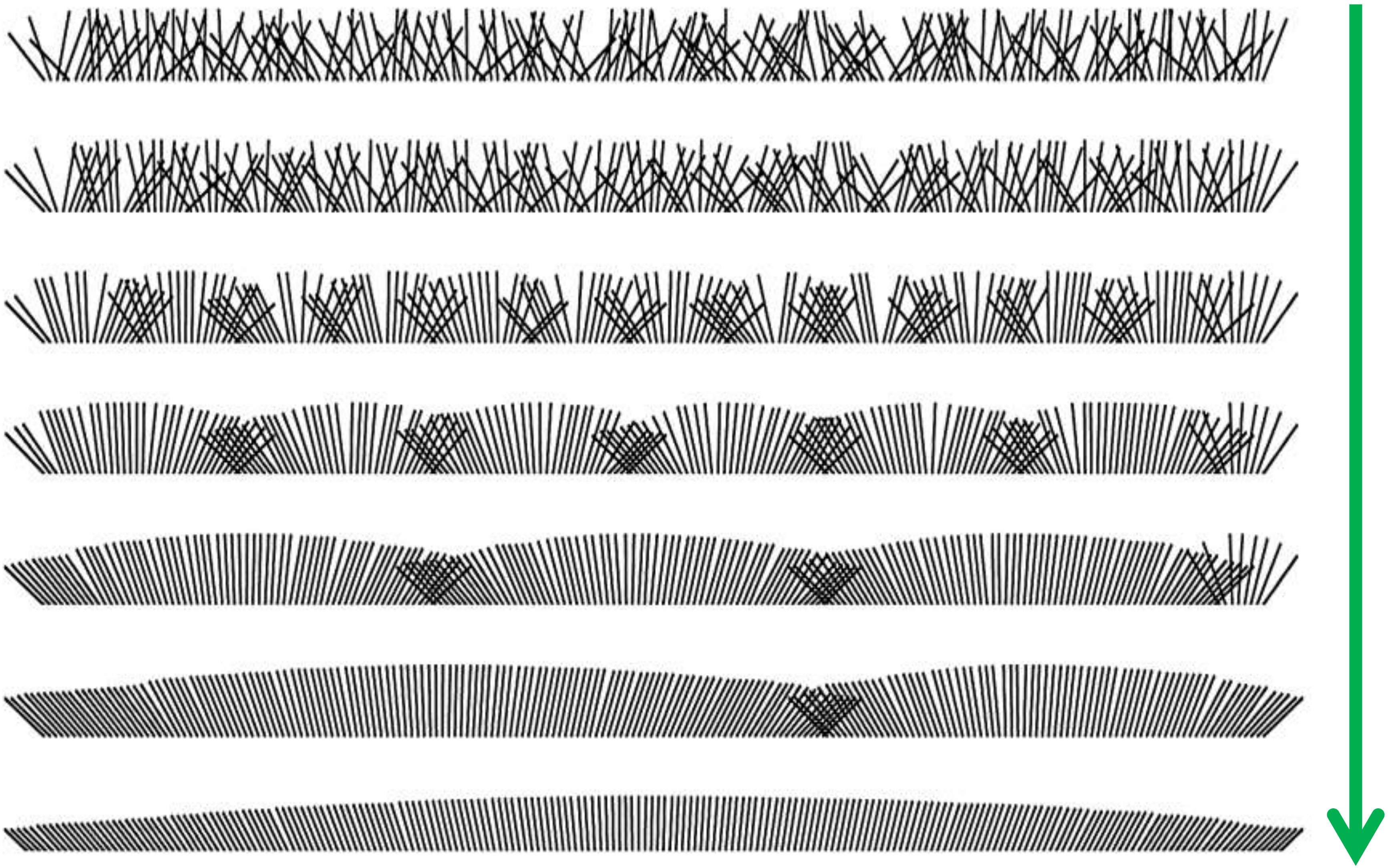
- Zeitmessungen bei einer konkreten Implementierung:
  - $n = 2^{22} \rightarrow 1.09$ s
  - $n = 2^{25} \rightarrow 9.94$ s
  - $n = 2^{29} \rightarrow 183$ s
  - $n = 2^{30} \rightarrow 1240$ s
- Beachte: $(2^{29} \times \log 2^{29}) / (2^{25} \times \log 2^{25}) = 16 \times 29/25 = 18.56$; dies stimmt gut mit der Beobachtung $183 / 9.94 = 18.41$ überein; die analytische Formel $n \log n$ sagt das Laufzeitwachstum gut vorher
- Allerdings fällt dann $n = 2^{30}$ aus der Reihe: Hier passten offenbar nicht mehr alle Werte in den Hauptspeicher; das Betriebssystem musste dafür (langsameren) Hintergrundspeicher allozieren

Jede Spalte anfangs eine andere Zufallspermutation



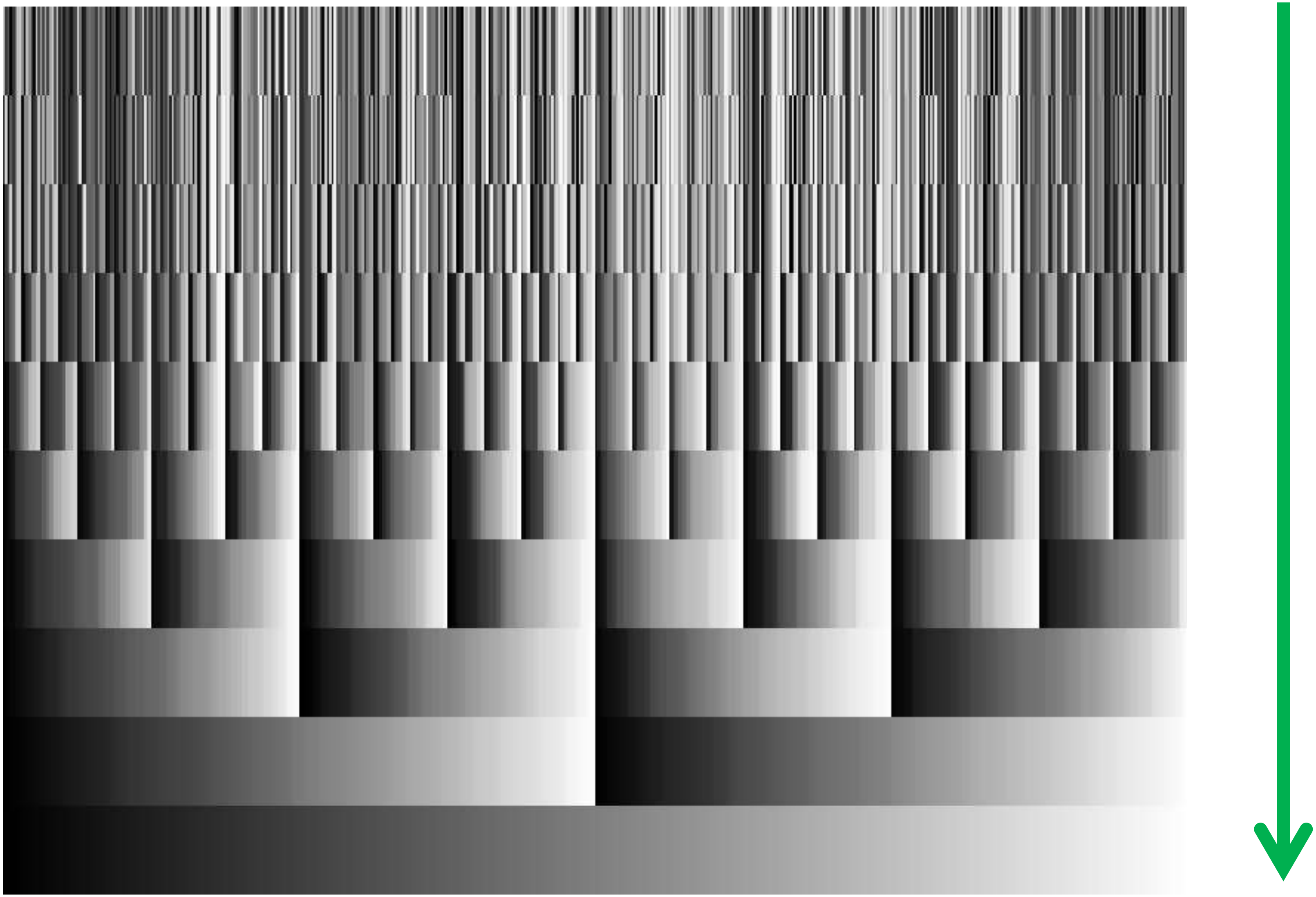
Video: https://i.imgur.com/TNIXT9P.mp4

Eine andere Visualisierung von Mergesort



Robert Sedgewick , http://mbostock.github.io/protovis/ex/sort.html

Noch eine andere Visualisierung von Mergesort



www.flickr.com/photos/_robson_/8235692086/

Mergesort – Historie

Donald Knuth analysiert 1970 ein Manuskript von John von Neumann von 1945, in dem dieser Mergesort beschreibt – einige Auszüge folgen auf den nächsten Seiten

Computing Surveys, Vol. 2, No. 4, December 1970

Von Neumann's First Computer Program

DONALD E. KNUTH

Stanford University, Stanford, California*

An analysis of the two earliest sets of instruction codes planned for stored program computers, and the earliest extant program for such a computer, gives insight into the thoughts of John von Neumann, the man who designed the instruction sets and wrote the program, and shows how several important aspects of computing have evolved. The paper is based on previously unpublished documents from the files of Herman H. Goldstine.

Key words and phrases: electronic computers, computer history, stored program computers, machine organization and architecture, sorting, latency time, ENIAC, EDVAC, order code, programming techniques

CR categories: 1.2, 6.0

Von Neumann's First Computer Program

INTRODUCTION

A handwritten document now in the possession of Dr. Herman H. Goldstine contains what is probably the earliest extant program for a stored program digital computer. Its author, the remarkably talented mathematician John von Neumann (1903-1957), was in the process of refining the stored program concept as he was writing this code; so his program represents a significant step in the evolution of computer organization as well as of programming techniques. ...

The program we will study is not what we might expect an "ordinary" mathematician to have written; it does not solve a partial differential equation! Instead, it deals with what was considered at that time to be the principal example of a nonnumeric application for computers, namely, the problem of sorting data into nondecreasing order.

Von Neumann chose this application for good reason. He had sketched out an instruction code for a stored program computer, with numerical applications uppermost in his mind; there was no question that his proposed device could do the requisite arithmetic operations. The key question was whether or not the proposed instruction set provided a satisfactory means of logical control for complex processes, and so he felt that a sorting program would be a most instructive test case. …

We should realize that the historical interest of this program is in great measure due to its connection with the development of instruction codes for stored program computers; it is not the earliest instance of a computer program. We have Lady Lovelace's description of a program for calculating Bernoulli numbers that Babbage wrote for his Analytical Engine; A. M. Turing's construction of his abstract Universal Machine …

Von Neumann’s First Computer Program (2)

Von Neumann’s first draft report on the EDVAC proposed building a serial computer with three 32-bit registers and 8192 32-bit words of auxiliary memory. … Each 32-bit word was either a number or an instruction code; the first bit was 0 for numbers and 1 for instructions. …

Von Neumann’s ... manuscript, written in ink, is 23 pages long; the first page still shows traces of the penciled phrase “TOP SECRET,” which was subsequently erased. (In 1945, work on computers was classified, due to its connections with military problems.) …

Von Neumann begins his memo by defining the idea of sorting records into order, and of merging two strings of records that have been sorted separately into a single sorted sequence. Then he states the purpose of the program: “We wish to formulate code instructions for sorting and for meshing [i.e. merging], and to see how much control capacity they tie up and how much time they require.” …

Like nearly all programs, this one has a bug: The second-last instruction “CON 1” actually belongs two lines earlier. If von Neumann had had an EDVAC on which to run this program, he would have discovered debugging! ...

After having written the program, he assigned actual addresses to the subscripted ones. In order to make the code relocatable, for use as a general open subroutine, he assigned the addresses relative to an unspecified starting location e. His address assignments are shown in Figure 2 at the right of the instructions.

→ nächste Seite

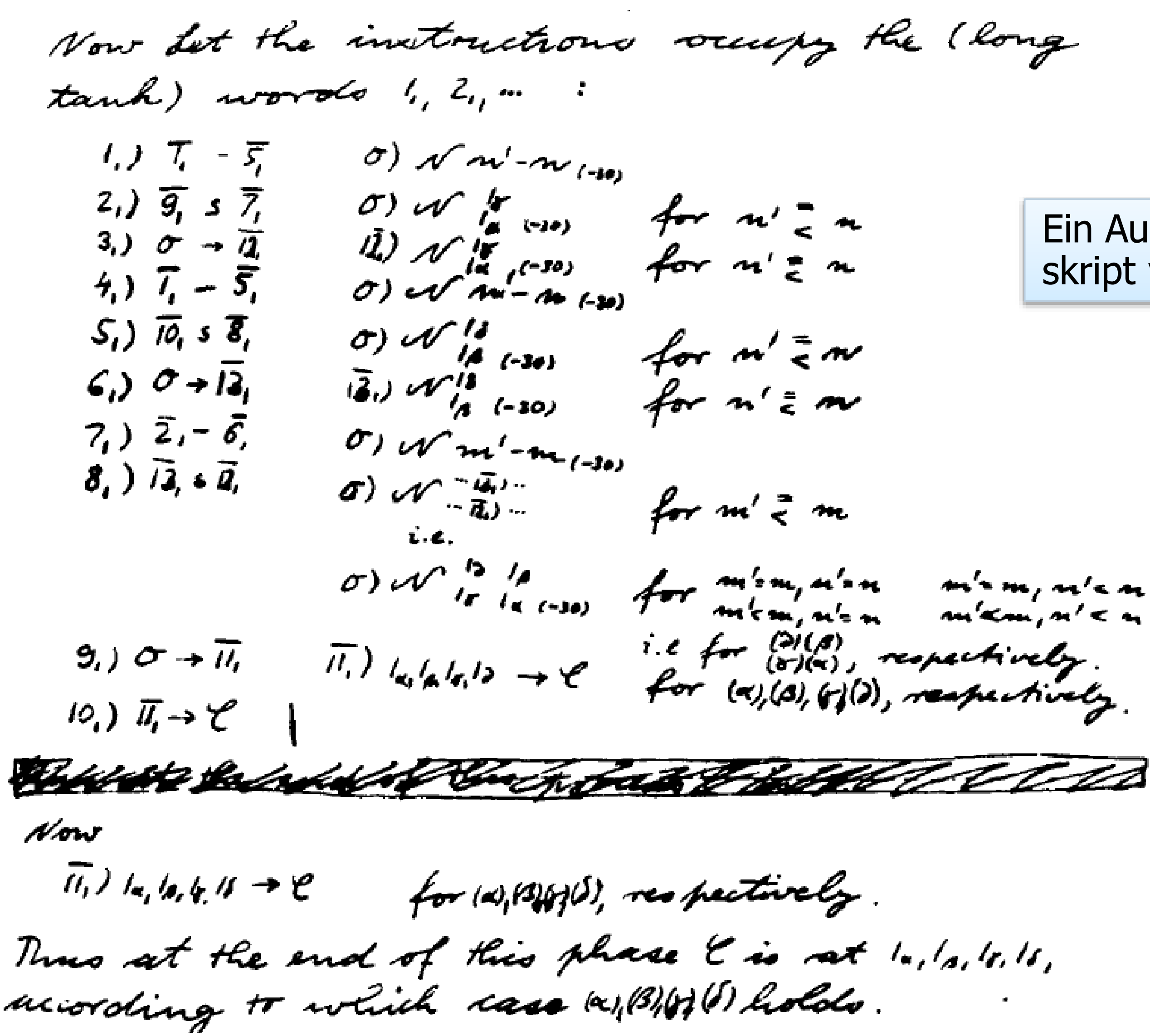
EDVAC (Electronic Discrete Variable Automatic Computer) wurde aufbauend auf den Erfahrungen mit ENIAC ab 1944 geplant und bis 1949 realisiert; das Programm residierte im Hauptspeicher.

Von Neumann's First Computer Program (3)

```
N         RST  1
M         RST  1
XPTR      RST  1
YPTR      RST  1
ZPTR      RST  1
SIZE      RST  1
XKEY      RST  1
YKEY      RST  1
NPRIME    RST  1
MPRIME    RST  1
LALPHA    RST  1
LBETA     RST  1
LGAMMA    RST  1
LDELTA    RST  1
SWITCH    RST  1
LALPHA1   RST  1
LALPHA2   RST  1
ZERO      RST  1
MONE      RST  1
ONE       RST  1
MOVEIN    RST  1
MOVEOUT   RST  1
RETURN    RST  1
BUFFER    RST  p+1
TEMP1     EQU  BUFFER+1
TEMP2     EQU  BUFFER+2
COMPARE   SUB  NPRIME,N           e+27
          SEL  LGAMMA,LALPHA      e+28
          STO  TEMP1              e+29
          SUB  NPRIME,N           e+30
          SEL  LDELTA,LBETA       e+31
          STO  TEMP2              e+32

          SUB  MPRIME,M           e+33
          SEL  TEMP2,TEMP1        e+34
          STO  SWITCH             e+35
          JMP  SWITCH             e+36
ALPHA     SUB  YKEY,XKEY          e+43
          SEL  LALPHA1,LALPHA2    e+44
          STO  SWITCH             e+45
          JMP  SWITCH             e+46
BETA      SUB  ZERO,ZERO          e+39
          TRA  ALPHA+1            e+40
GAMMA     SUB  MONE,ZERO          e+41
          TRA  ALPHA+1            e+42
DELTA     TRA  EXIT               e+81
ALPHA1    SET  MOVEIN,XPTR        e+47
          SET  MOVEOUT,ZPTR       e+48
          PIK  1,RETURN           e+49
          TRA  BACK1              e+50
          JMP  MOVEIN             e+51
BACK1     ADD  NPRIME,ONE         e+56
          STO  NPRIME             e+57
          SET  XKEY,BUFFER+p      e+58
          ADD  XPTR,SIZE          e+59
          STO  XPTR               e+60
          ADD  ZPTR,SIZE          e+61
          STO  ZPTR               e+62
          TRA  COMPARE            e+63
ALPHA2    SET  MOVEIN,YPTR        e+64
          SET  MOVEOUT,ZPTR       e+65
          PIK  1,RETURN           e+66
          TRA  BACK2              e+67
          JMP  MOVEIN             e+68

BACK2     ADD  MPRIME,ONE         e+73
          STO  MPRIME             e+74
          SET  YKEY,BUFFER+p      e+75
          ADD  YPTR,SIZE          e+76
          STO  YPTR               e+77
          ADD  ZPTR,SIZE          e+78
          STO  ZPTR               e+79
          TRA  COMPARE            e+80
BRING     EQU  NPRIME
MERGE     PIK  3,BRING            e+0
          PIK  1,XKEY,**          e+1
          PIK  1,YKEY,**          e+2
          TRA  BACK3              e+3
          SET  BRING,XPTR         e+4
          SET  BRING+1,YPTR       e+5
          JMP  BRING              e+6
BACK3     PIK  14,NPRIME          e+11
          CON  0                  e+12
          CON  0                  e+13
          CON  ALPHA              e+14
          CON  BETA               e+15
          CON  GAMMA              e+16
          CON  DELTA              e+17
          TRA  **                 e+18
          CON  ALPHA1             e+19
          CON  ALPHA2             e+20
          CON  0                  e+21
          CON  -1                 e+22
          PIK  p+1,BUFFER,**      e+23
          PUT  p,BUFFER,**        e+24
          CON  1                  e+25
          TRA  COMPARE            e+26
```

Von Neumanns Sortierprogramm (von Donald Knuth in eine besser lesbare Form gebracht)

Von Neumann's First Computer Program (4)


Now let the instructions occupy the (long tank) words 1, 2, ... :

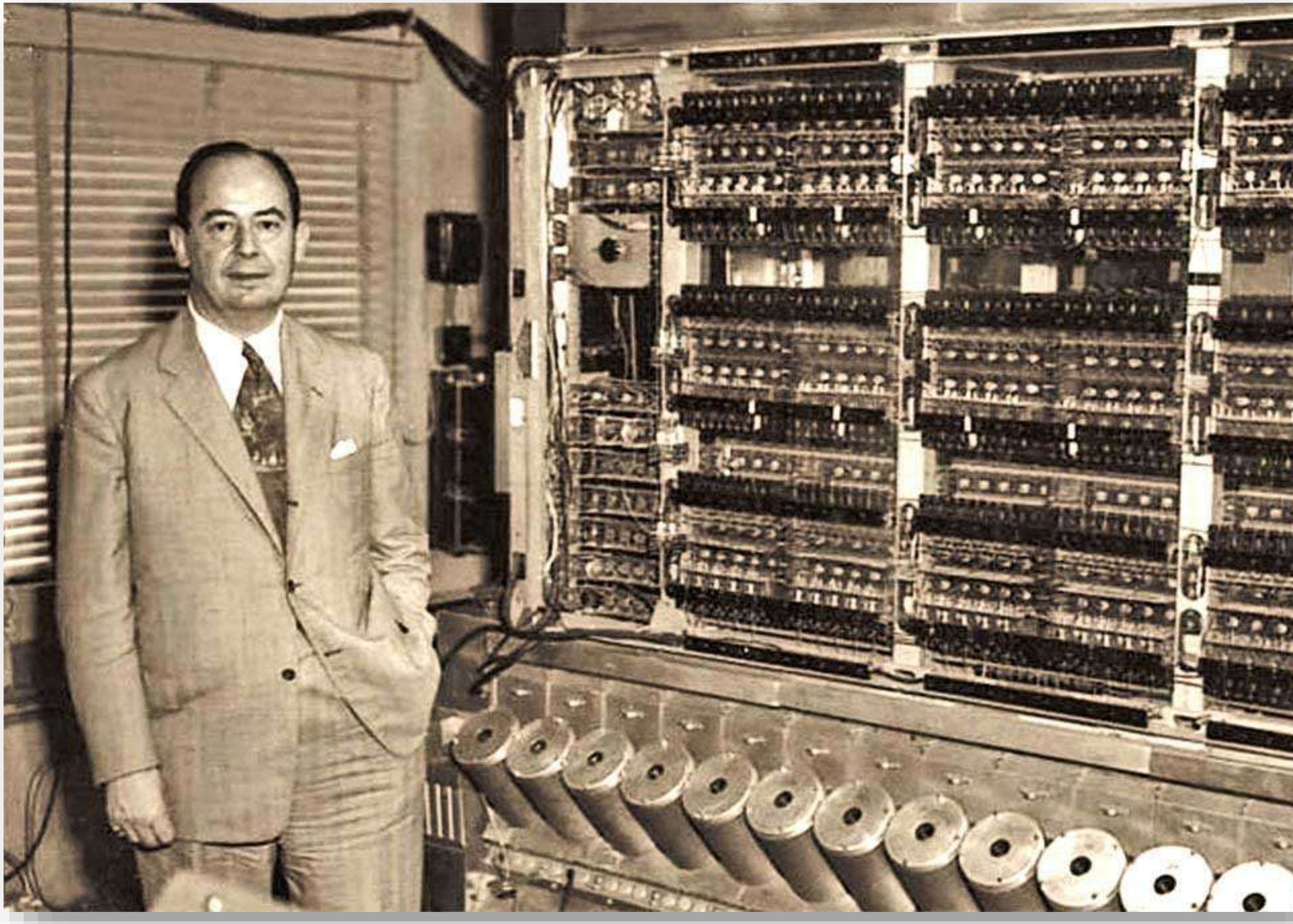
1,) $\bar{1}_1 - \bar{5}_1$ σ) $N\ m'-m\ (-30)$
2,) $\bar{9}_1\ s\ \bar{7}_1$ σ) $N^{1\varepsilon}_{1\alpha}\ (-30)$ for $m' \geq< n$
3,) $\sigma \to \bar{12}_1$ $\bar{12}$) $N^{1\varepsilon}_{1\alpha}\ (-30)$ for $m' \geq< n$
4,) $\bar{1}_1 - \bar{5}_1$ σ) $N\ m'-m\ (-30)$
5,) $\bar{10}_1\ s\ \bar{8}_1$ σ) $N^{1\delta}_{1\beta}\ (-30)$ for $m' \geq< m$
6,) $\sigma \to \bar{13}_1$ $\bar{13}_1$) $N^{1\delta}_{1\beta}\ (-30)$ for $n' \geq< m$
7,) $\bar{2}_1 - \bar{6}_1$ σ) $N\ m'-m\ (-30)$
8,) $\bar{13}_1\ s\ \bar{11}_1$ σ) $N^{\cdots \bar{13}_1) \cdots}_{\cdots \bar{12}_1) \cdots}$ for $m' \geq< m$
i.e.
σ) $N^{1\delta\ 1\beta}_{1\varepsilon\ 1\alpha}\ (-30)$ for $m' \geq m, n' \geq n$; $m' \geq m, n' < n$; $m' < m, n' \geq n$; $m' < m, n' < n$
9,) $\sigma \to \bar{11}_1$ $\bar{11}_1$) $1_\alpha, 1_\beta, 1_\varepsilon, 1_\delta \to C$ i.e for (δ)(β) (ε)(α), respectively.
10,) $\bar{11}_1 \to C$ for (α),(β),(ε),(δ), respectively.

Now
$\bar{11}_1$) $1_\alpha, 1_\beta, 1_\varepsilon, 1_\delta \to C$ for (α),(β),(ε),(δ), respectively.

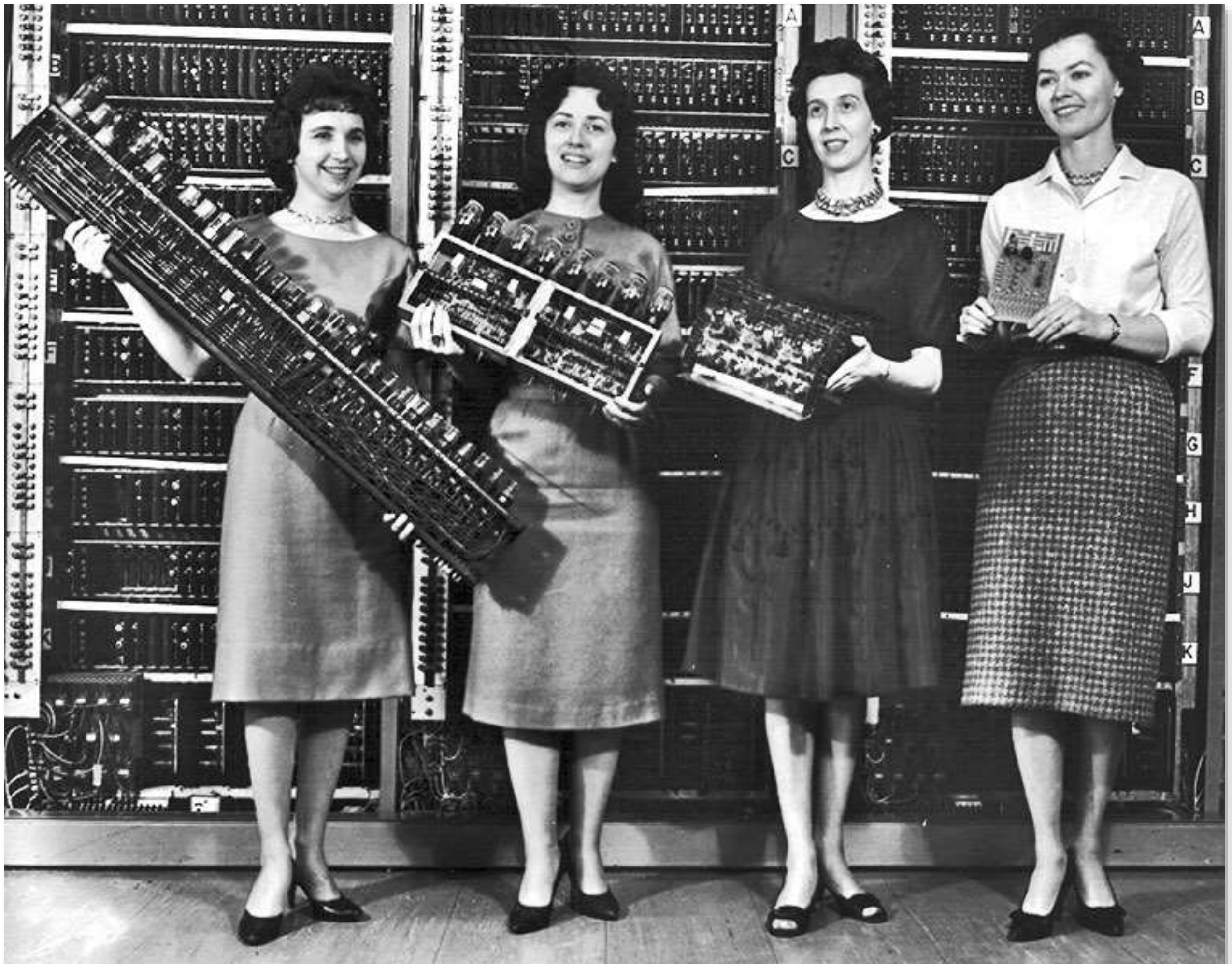
Thus at the end of this phase C is at $1_\alpha, 1_\beta, 1_\varepsilon, 1_\delta$, according to which case (α),(β),(ε),(δ) holds.

Ein Ausschnitt aus dem Manuskript von John von Neumann

John von Neumann mit dem EDVAC



ENIAC, EDVAC,...



https://en.wikipedia.org/wiki/File:Women_holding_parts_of_the_first_four_Army_computers.jpg

Left: Patsy Simmers, holding *ENIAC* board. Next: Mrs. Gail Taylor, holding *EDVAC* board, Mrs. Milly Beck, holding *ORDVAC* board, Mrs. Norma Stec, holding *BRLESC-I* board.

Dieses bekannte Bild von 1962 soll die fortschreitende Miniaturisierung bei der Hardware von Computern über die Zeit illustrieren, hier dargestellt an Baugruppen zur Repräsentation einer einzigen Dezimalziffer verschiedener Maschinen des „Ballistic Research Laboratory" (BRL) der US-Streitkräfte in Aberdeen, Maryland, USA. 1962 wurde deren neuester Rechner, BRLESC-I (1727 Elektronenröhren und 853 Transistoren; Hauptspeicher von 4096 Wörtern zu 72 Bit), in Betrieb genommen.

ENIAC, EDVAC,...

http://apgnews.com/wp-content/uploads/2017/06/Ladies-1_cropped.jpg

June 7, 2017: Retired ARL employees Patsy Simmers (left) and Norma Stec (right) pose with a 1962 photo of themselves. They viewed part of the Electronic Numerical Integrator and Computer, or ENIAC, which was one of the world's earliest general-purpose electronic computers. In the photo, taken 55 years ago, Simmers is holding the same piece of the ENIAC.

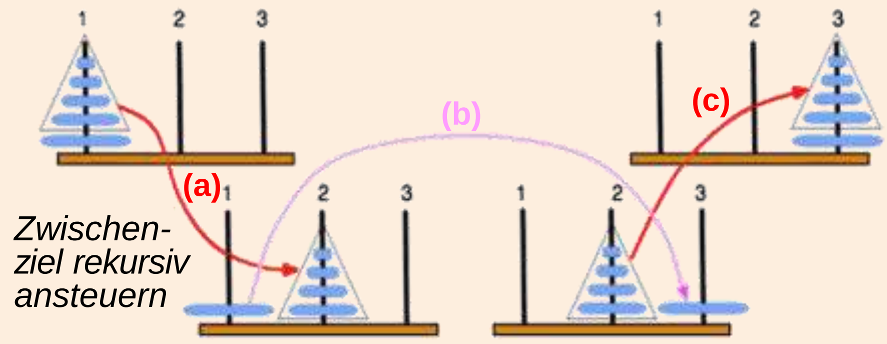
ENIAC, EDVAC, BRL…

Für welche Probleme beim Ballistic Research Laboratory (BRL) Computer wie EDVAC eingesetzt wurden, wurde 1961 in einem mehr als 1000 Seiten und 222 Systembeschreibungen umfassenden Report *A Third Survey of Domestic Electronic Digital Computing Systems* von Martin H. Weik vom BRL (Aberdeen Proving Ground, Maryland) so erläutert:

Exterior ballistics problems such as high altitude, solar and lunar trajectories, computation for the preparation of firing tables and guidance control data for Ordnance weapons, including free flight and guided missiles. *Interior ballistic problems*, including projectile, propellant and launcher behavior, e.g. physical characteristics of solid propellants, equilibrium composition and thermodynamic properties of rocket propellants, computation of detonation waves for reflected shock waves, vibration of gun barrels and the flow of fluids in porous media. *Terminal ballistic problems*, including nuclear, fragmentation and penetration effects in such areas as explosion kinetics, shaped charge behavior, ignition, and heat transfer. *Ballistic measurement problems*, including photogrammetric, ionospheric, and damping of satellite spin calculations, reduction of satellite doppler tracking data, and computation of satellite orbital elements. *Weapon systems evaluation problems*, including anti-aircraft and anti-missile evaluation, war game problems, linear programming for solution of Army logistical problems, probabilities of mine detonations, and lethal area and kill probabilities of mine detonations, and lethal area and kill probability studies of missiles.

Resümee des Kapitels

- Rekursives Problemlösen
  - Zwischenziel als Teilproblem
  - Beispiel: „Türme von Hanoi“



- Mergesort
  - Rekursiver Lösungsansatz
  - Top-down ↔ bottom-up
  - Mit verketteter Liste bzw. Array
  - Zeitaufwand proportional zu *n log n*
  - Struktureller Vergleich mit Quicksort