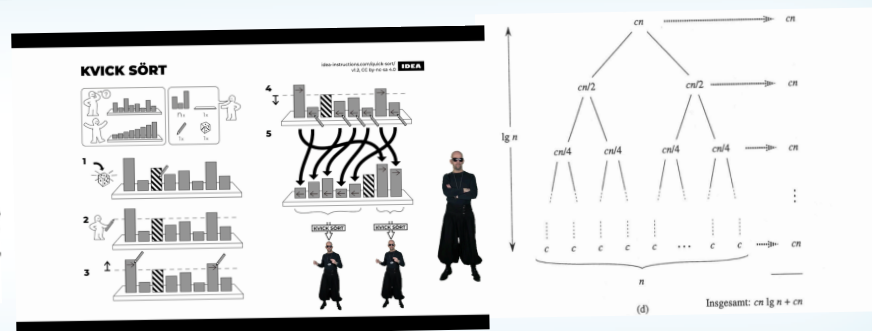


8	9	10	11	12	13	14	15	16
...	1	2	2	3	1	2	3	6
k								
2	3	4	5					
4	5	7	∞					
i								

1	2	3
1	2	3

R



Kapitel 5.7: *Sortieren in Linearzeit* *Algorithmen und Datenstrukturen* *WS 2024/25*

Prof. Dr. Sándor Fekete

Laufzeitschranke fürs Sortieren

Satz 5.6. *Für n Objekte $x_1, \dots, x_n$*

benötigt man zum Sortieren mindestens $\Omega(n \log(n))$,

wenn man die Objekte nur paarweise vergleichen kann.

Countingsort

Eingabe: Array $A[1], \dots, A[n]$ mit Schlüsselwerten aus $\{1, \dots, k\}$

Ausgabe: Sortierte Kopie $B[1], \dots, B[n]$ von $A[1], \dots, A[n]$

```
1: function COUNTINGSORT( $A, B, k$ )
2:   for  $i \leftarrow 1$  to  $k$  do
3:      $C[i] \leftarrow 0$                                 ▷ Zählerarray initialisieren
4:   for  $j \leftarrow 1$  to  $n$  do
5:      $C[A[j]] \leftarrow C[A[j]] + 1$                   ▷ Anzahl Elemente mit Wert  $A[j]$  um 1 erhöhen
6:   for  $i \leftarrow 1$  to  $k$  do
7:      $C[i] \leftarrow C[i] + C[i - 1]$                   ▷ Anzahl Elemente mit Wert höchstens  $i$  zählen
8:   for  $j \leftarrow n$  downto  $1$  do
9:      $B[C[A[j]]] \leftarrow A[j]$                       ▷ Element  $A[j]$  an die richtige Stelle in  $B$  schreiben
10:     $C[A[j]] \leftarrow C[A[j]] - 1$ 
```

Satz 5.15. Für n Objekte mit Schlüsseln aus $\{1, \dots, k\}$ benötigt Countingsort (s. Alg. 5.14) die Zeit $O(n + k)$. Insbesondere benötigt man für $k \in O(n)$ die Zeit von $\Theta(n)$.

Radixsort

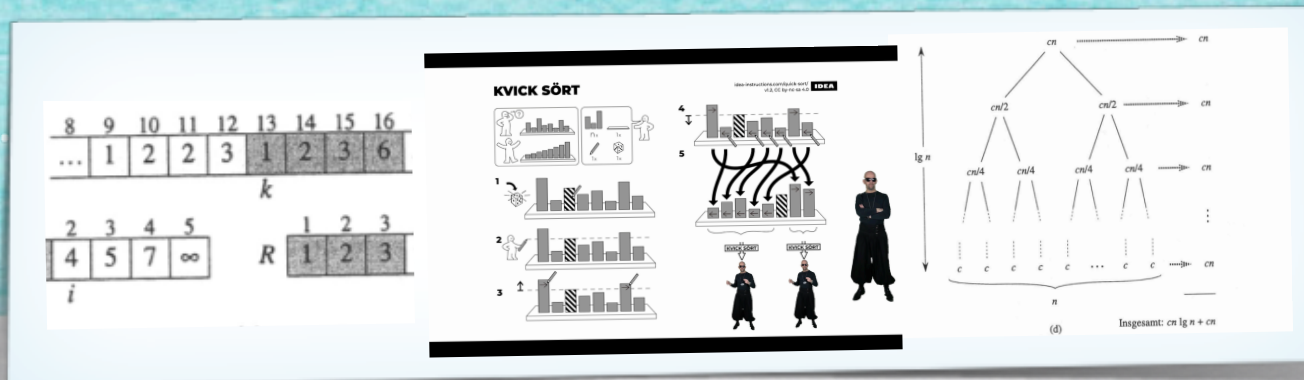
329	720	720	329
457	355	329	355
657	436	436	436
839	→ 457	→ 839	→ 457
436	657	355	657
720	329	457	720
355	839	657	839

Eingabe: n Zahlen mit je d Ziffern, die k verschiedene Werte annehmen können, $A[i], \dots, A[n]$

Ausgabe: Sortiertes Array

```
1: function RADIXSORT( $A, d$ )
2:   erzeuge (mit 0 initialisiertes) Array  $B$ 
3:   for  $i \leftarrow 1$  to  $d$  do
4:     COUNTINGSORT( $A, B, i$ )
```

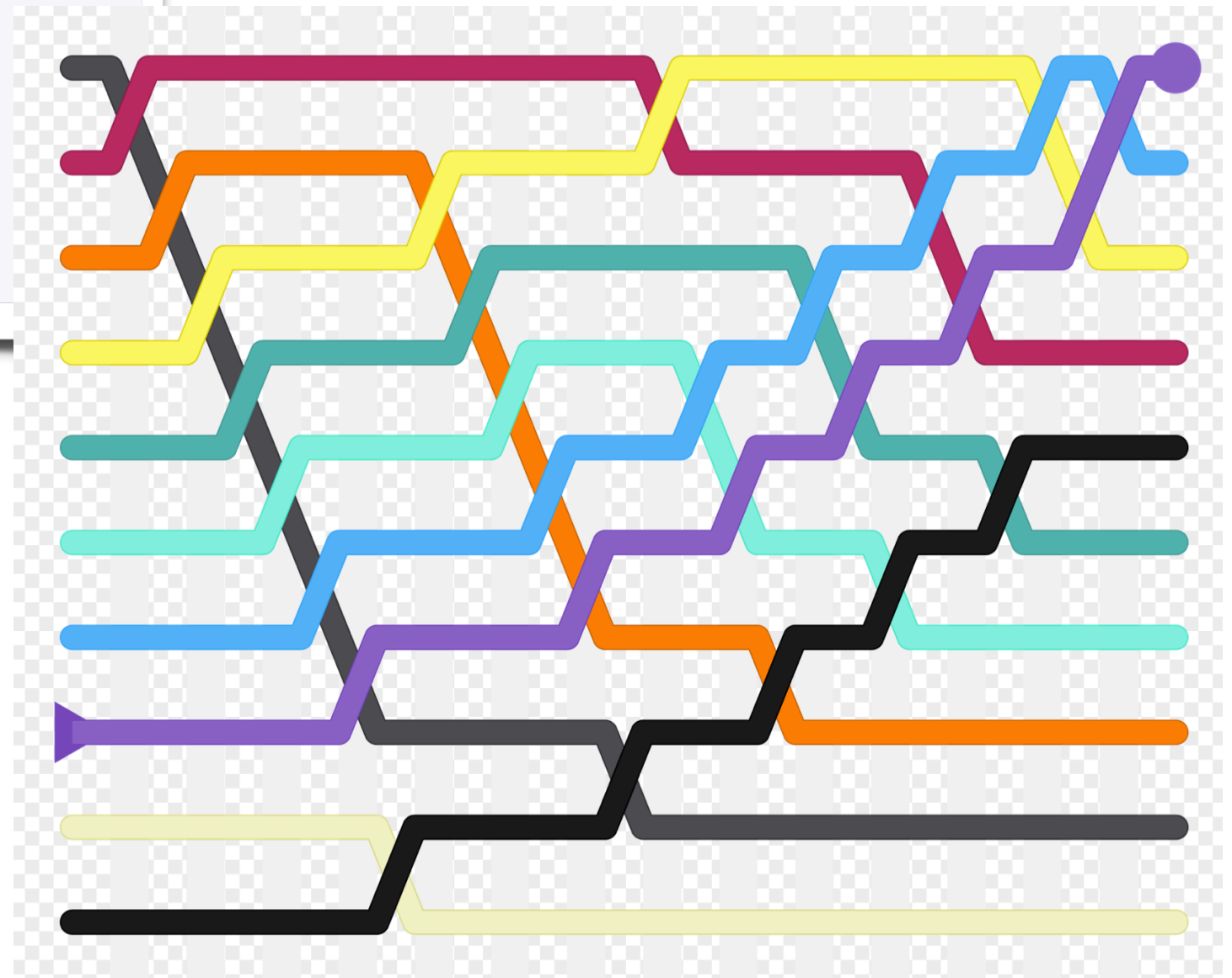
Satz 5.19. *Radixsort (s. Alg. 5.18) liefert ein korrektes Ergebnis in der Zeit $\Theta(d \cdot (n+k))$.*



Kapitel 5.8: *Paralleles Sortieren* *Algorithmen und Datenstrukturen* *WS 2024/25*

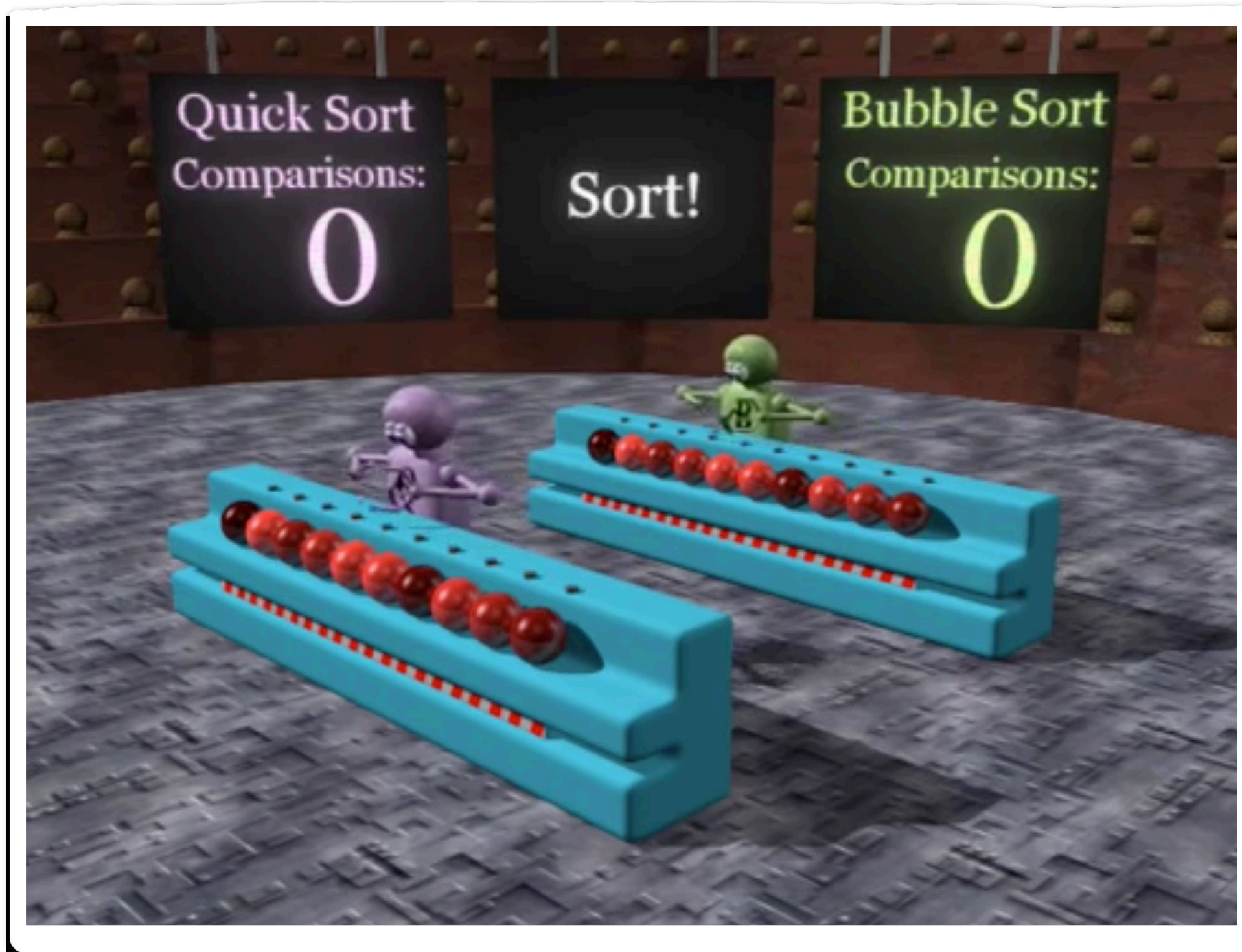
Prof. Dr. Sándor Fekete

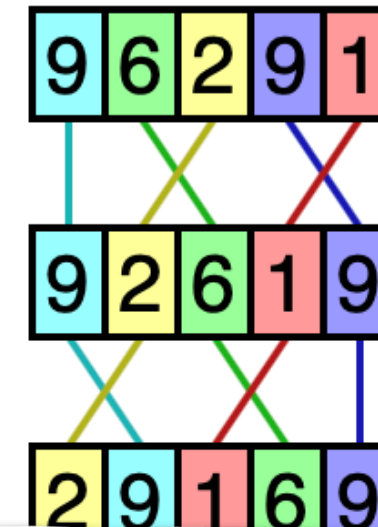
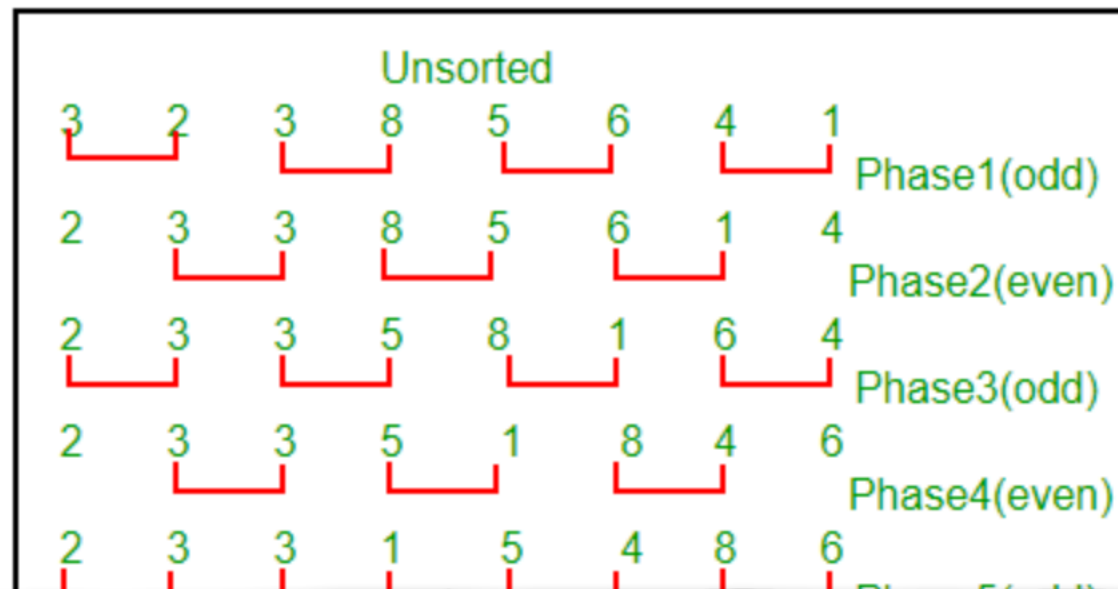

```
bubbleSort(Array A)
  for (n=A.size; n>1; --n){
    for (i=0; i<n-1; ++i){
      if (A[i] > A[i+1]){
        A.swap(i, i+1)
      } // Ende if
    } // Ende innere for-Schleife
  } // Ende äußere for-Schleife
```



Pmdumuid



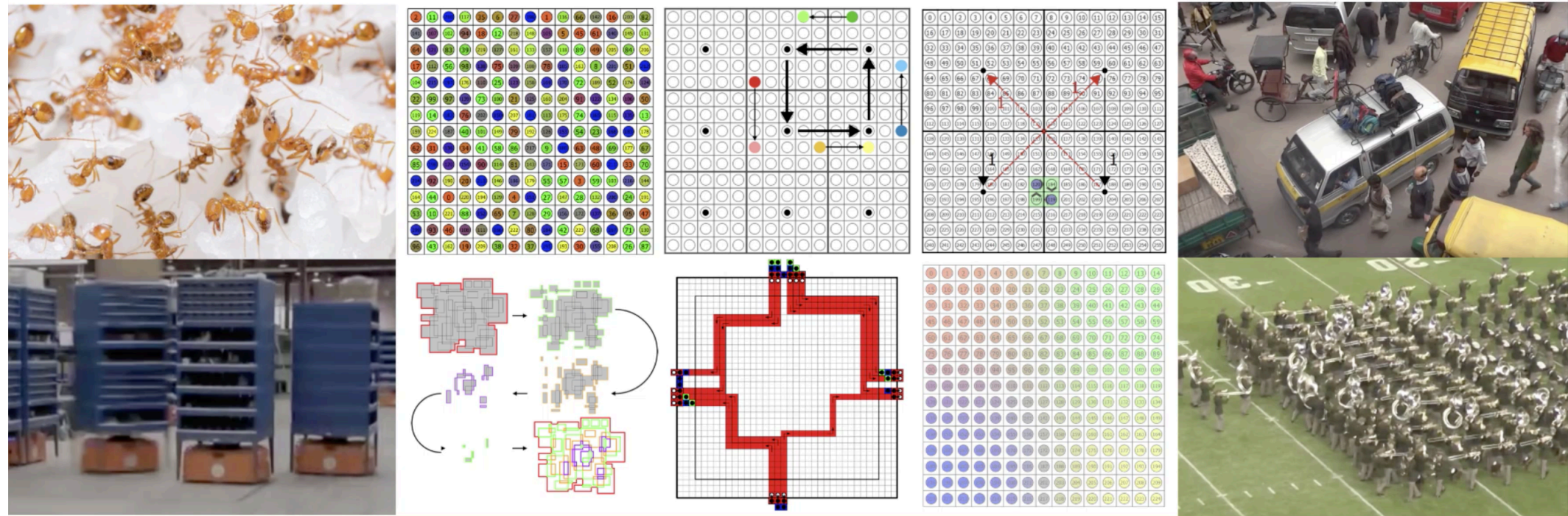




Satz 5.25

Paralleles Bubblesort sortiert die Zahlen in n parallelen Runden.
Die Gesamtzahl der Vergleiche ist $\Theta(n^2)$.

Beweis: Selbst!



Coordinated Motion Planning: The Video

**Aaron Becker, Sándor P. Fekete, Phillip Keldenich,
Matthias Konitzny, Lillian Lin, Christian Scheffer**

COMPUTER RECREATIONS

Analog gadgets that solve a diversity of problems and raise an array of questions

by A. K. Dewdney

Exactly one year ago a collection of analog gadgets in these pages set off an avalanche of similar devices from inspired readers. I am still extricating myself from a vast heap of wood boards, rubber bands, strings, balls of polystyrene, fish tanks, lead weights, canisters, tubing and stopcocks. In the process I have, I think, identified the best of these gadgets and have arranged them into a kind of gallery through which the reader is invited to wander.

Analog gadgets are mechanical devices that solve specific problems by virtue of the fact that their construction or behavior is analogous to the elements of the problem. For example, the June 1984 column described SAG, the Spaghetti Analog Gadget. Lengths

of uncooked spaghetti were used as analogs for numbers. To sort the numbers in decreasing order, gather the spaghetti into a bundle held vertically and bring it down rather sharply on a tabletop; select the longest rod. Continued selection of the longest remaining rod produces the desired decreasing sequence of numbers. In addition to SAG, I presented gadgets for finding shortest paths, convex hulls and minimum trees. I even displayed a gadget for factoring numbers that consisted of mirrors and a laser beam.

The latest collection includes several ingenious new gadgets for solving problems in statistics, network theory, algebra and arithmetic. On leaving the gallery I shall reexamine some important issues that arise from the prospect

of analog computing: How accurate are analog computers and how much time do they really take to compute? Are there some analog computers that outperform digital machines?

The first of the new gadgets solves a certain problem in statistics by means of a wood board, rubber bands, nails and a smooth, rigid rod. A set of data points plotted on a sheet of graph paper may present a linear trend to the eye. If a linear relation really governs the points, what straight line best displays the relation? The gadget suggested by Marc Hawley of Mount Vernon, Ind., supplies one possible answer:

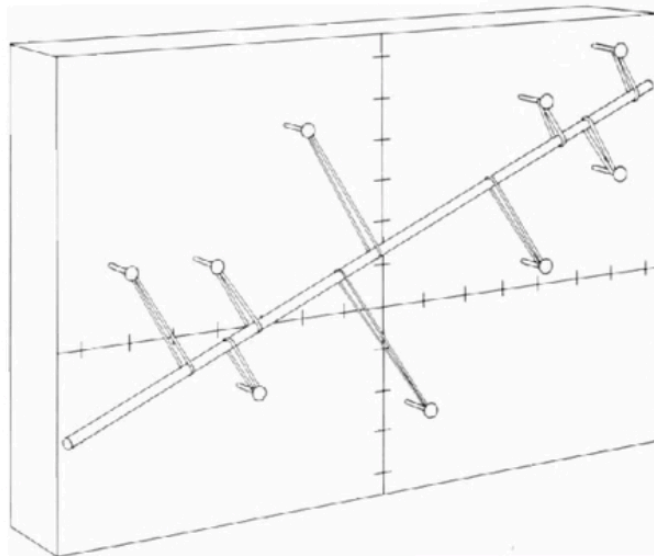
Plot the data points on a wood surface and drive a nail partway into the wood at each point. Next, slip a number of uniform rubber bands onto the rod, one band for each nail. Fit the rod approximately into place and pull each band over one of the nails. When the rod is released, it wiggles and shivers quickly into an equilibrium position [see illustration on this page].

The equilibrium position minimizes the total energy of the system; therefore the sum of the distances from the nails to the rod has also been minimized. In terms of such distances, the rod's final position indicates the straight line that best fits the data. It is not such distances but their squares that appear in the formulas for linear regression used by statisticians. Hawley's gadget computes something at least as complicated.

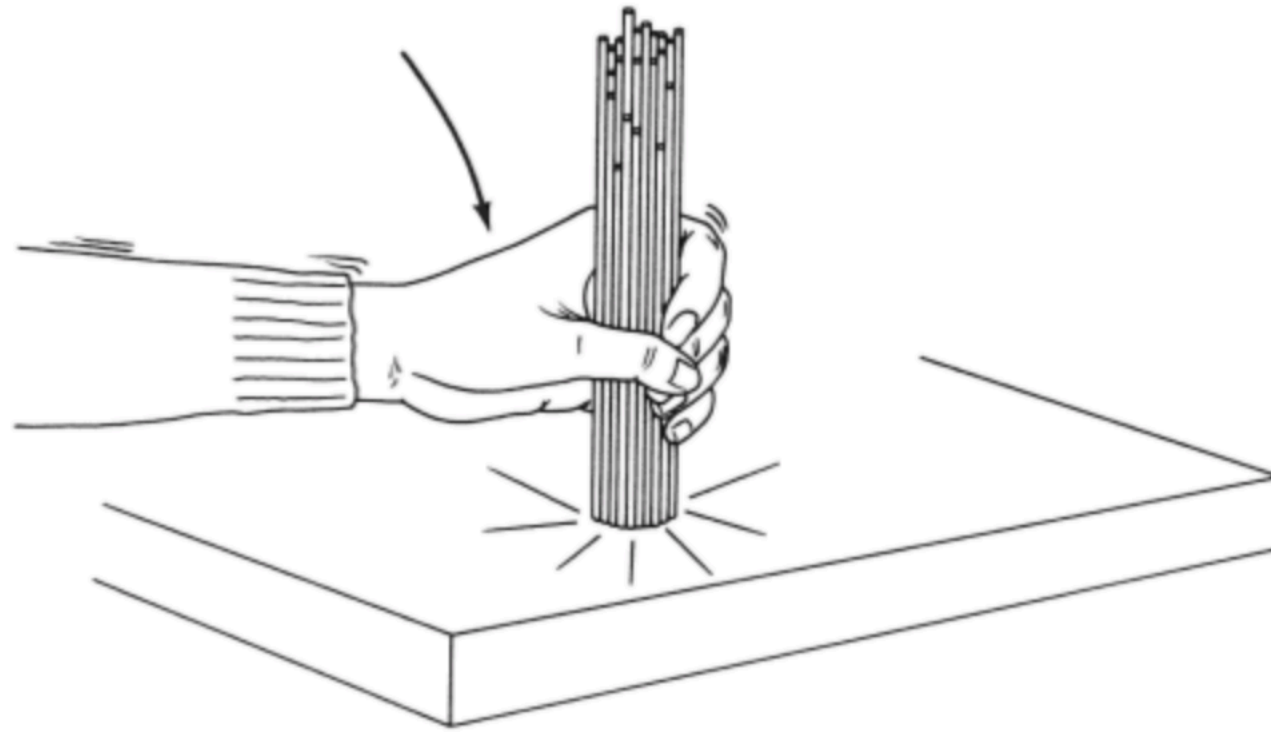
A charming string gadget was suggested by Jos Wennmacker of Nijmegen, Holland. Suppose we wish to know the longest path any message might travel in a communications network shaped like a tree. This path will be what combinatorial mathematicians call the diameter of the tree. In order to find the diameter Wennmacker reconstructs the tree by knotting together an analogy out of pieces of string. Each string is scaled to a specific communications line of the network. Two simple steps complete the computation. Pick up the string tree at any node and allow it to dangle freely. Now pick up the tree anew at the lowest node and dangle it once more. The longest path in the tree runs from the top node to the bottom one [see illustration on page 22].

When I initially encountered Wennmacker's gadget, my immediate reaction was, "It can't be that simple. Surely I must continue to select the bottom node and dangle the tree several more times." But one does not need to do that. Kudos in this column for the most elegant argument.

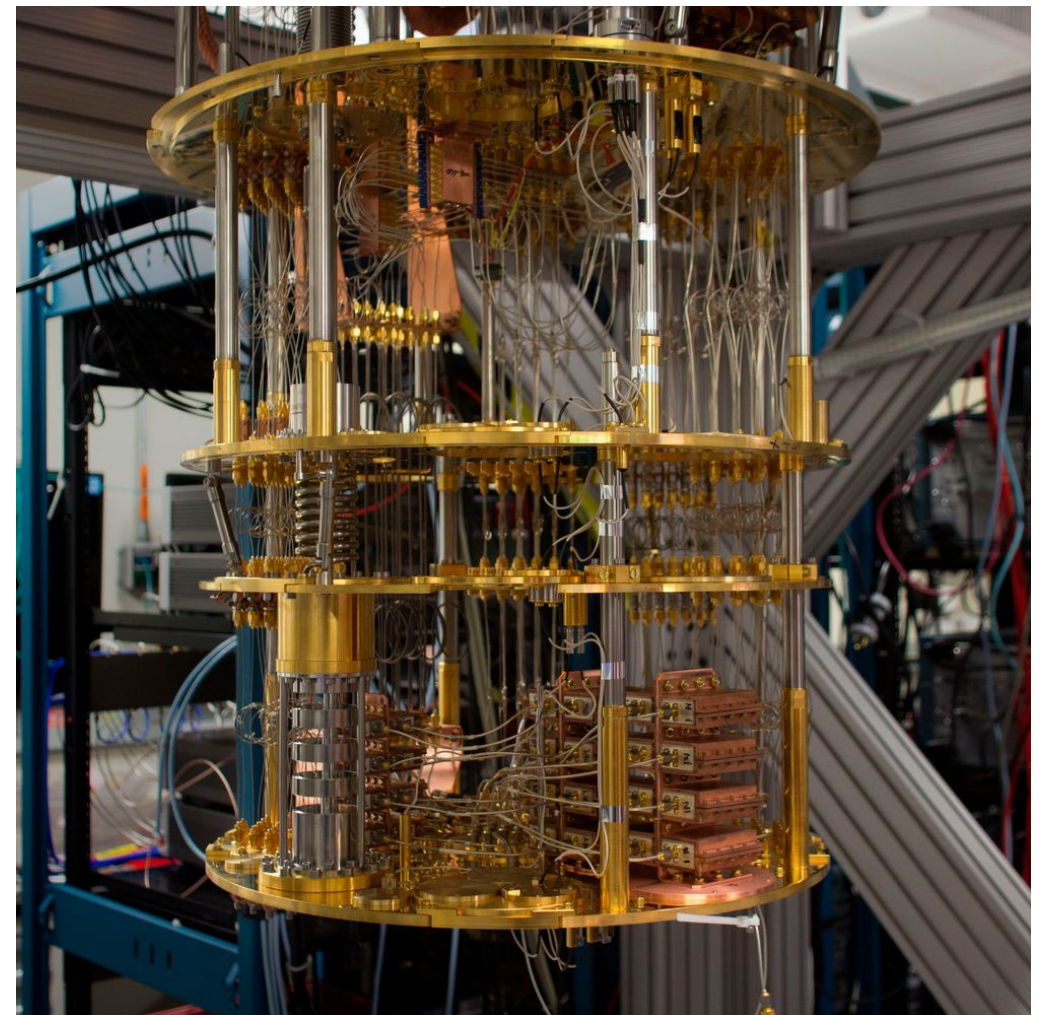
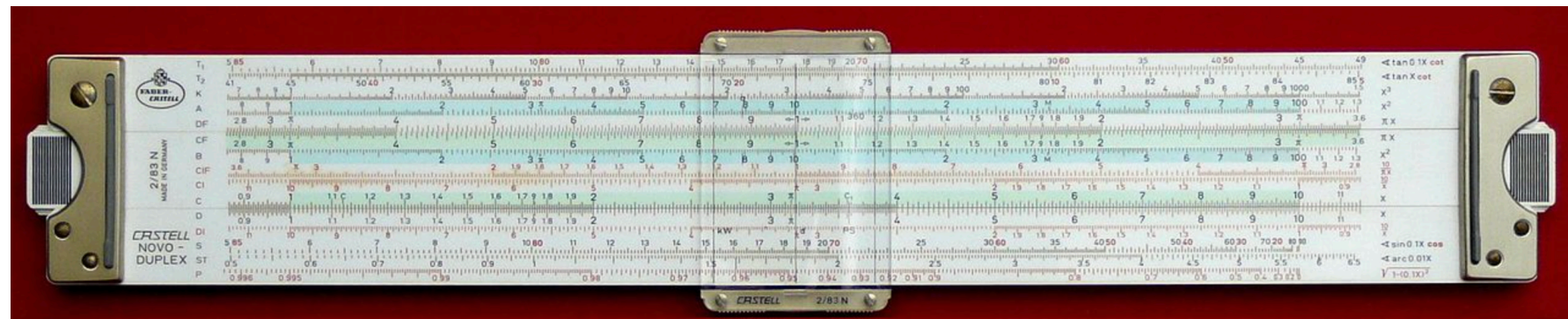
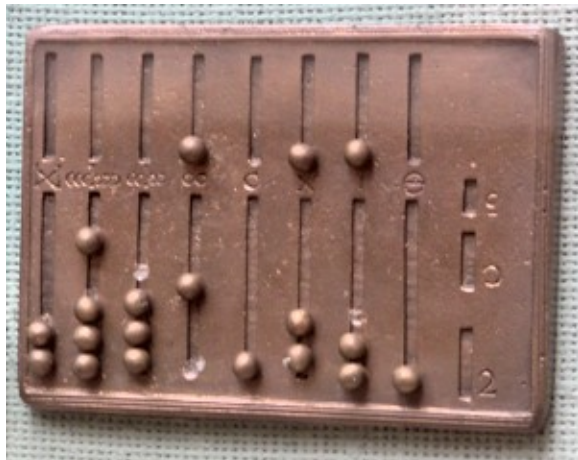
In last June's column I stated that the problem of finding the longest path in an arbitrary network was what theo-



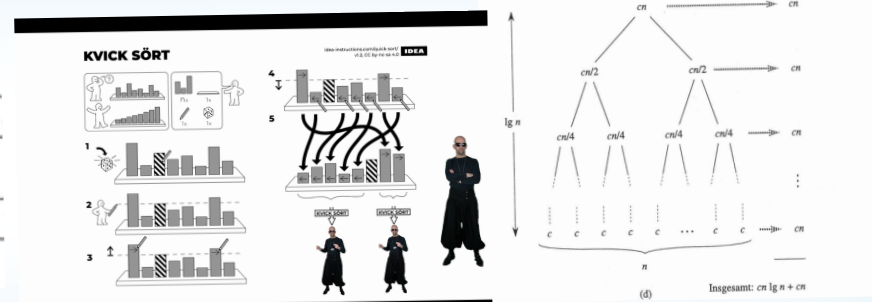
A gadget for finding the line that best fits a series of data points



1. For each number, break off a piece of spaghetti, whose length equals the number. This takes linear time.
2. Take all spaghetti in your fist, and slam their lower sides on the table, so they all point up starting at the table. This takes constant time, thanks to the multi-processing properties of our universe.
3. Lower your other hand on the bundle of spaghetti. Take out the one which you touch first. This is clearly the longest one. Continue to lower your hand and to remove spaghetti, laying them out next to each other, until all spaghetti have been processed. Again, this takes linear time.
4. Transcribe the spaghetti lengths back to the respective numbers, using linear time. Done!



8	9	10	11	12	13	14	15	16
...	1	2	2	3	1	2	3	6
k								
2	3	4	5					
4	5	7	∞					
i								



Kapitel 5.10:

Unernste Verfahren

Algorithmen und Datenstrukturen

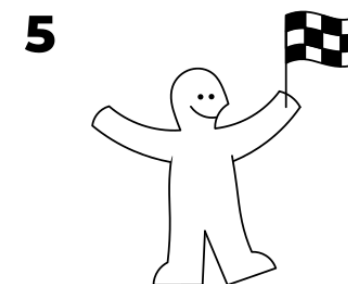
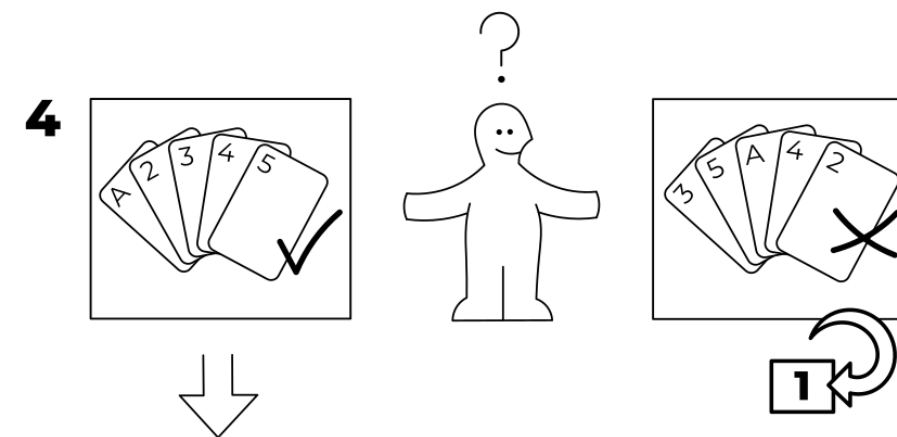
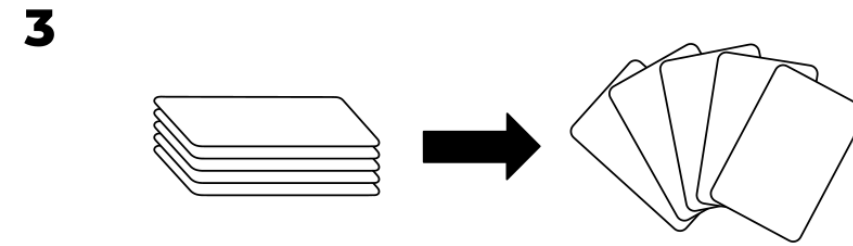
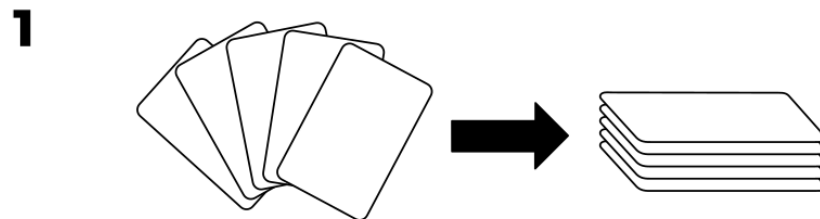
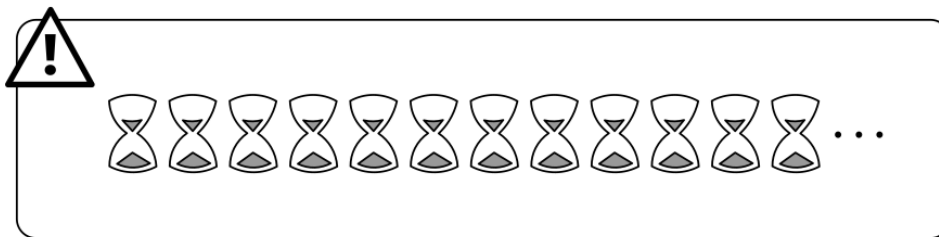
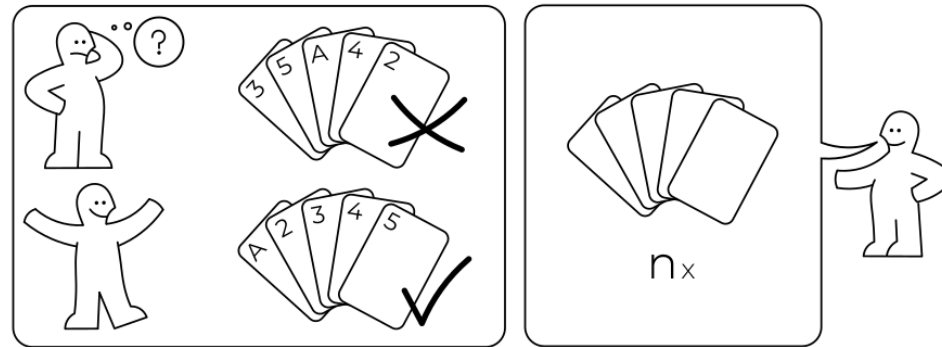
WS 2024/25

Prof. Dr. Sándor Fekete

BOGO SÖRT

idea-instructions.com/bogo-sort/
v1.2, CC by-nc-sa 4.0

IDEA



ALGORITHMUS 5.28 (Bogobogosort)

INPUT: n Zahlen $A[1], \dots, A[n]$

OUTPUT: Sortierter Array

1. $i := 2$

2. WHILE ($i \leq n$) DO

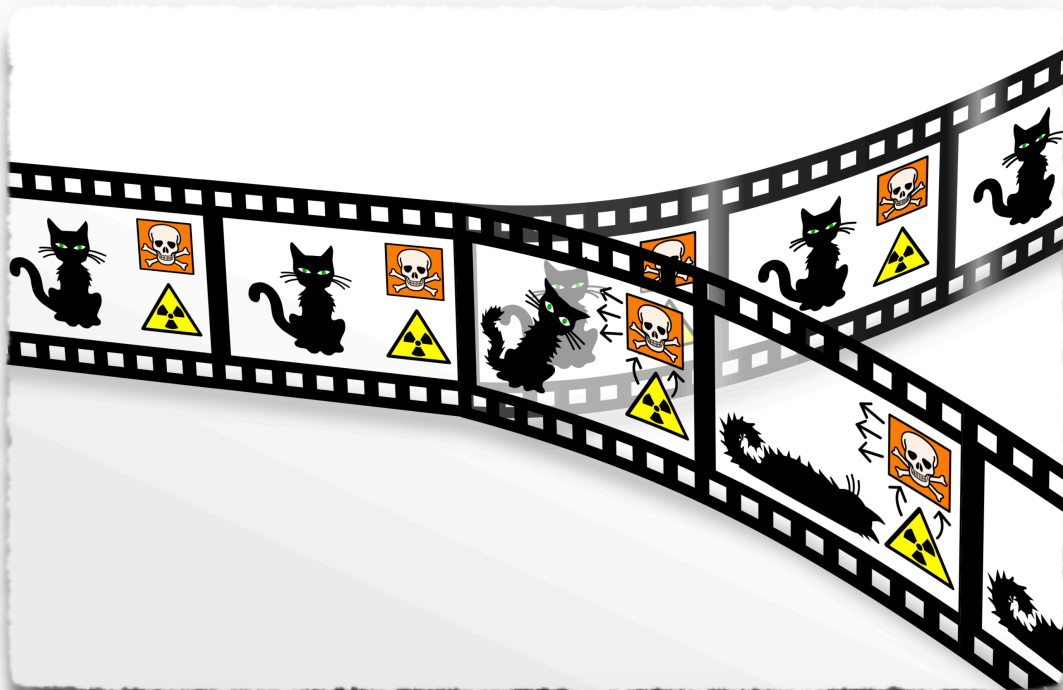
2.1 wähle Zufallspermutation der Elemente $A[1], \dots, A[i]$

2.2 IF (Permutation ist sortiert)

2.2.1 $i := i + 1$

2.3 ELSE

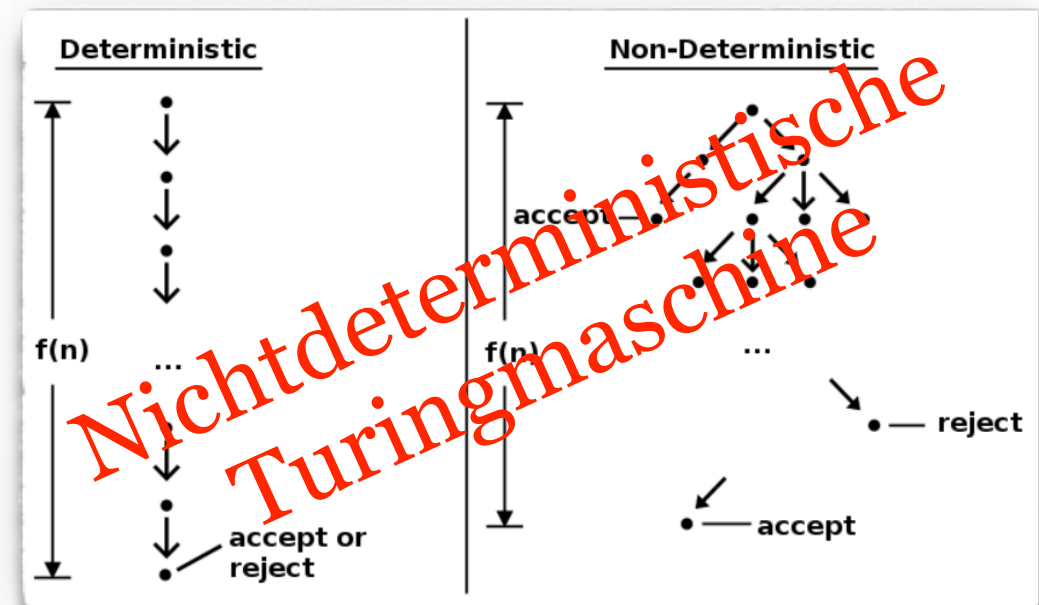
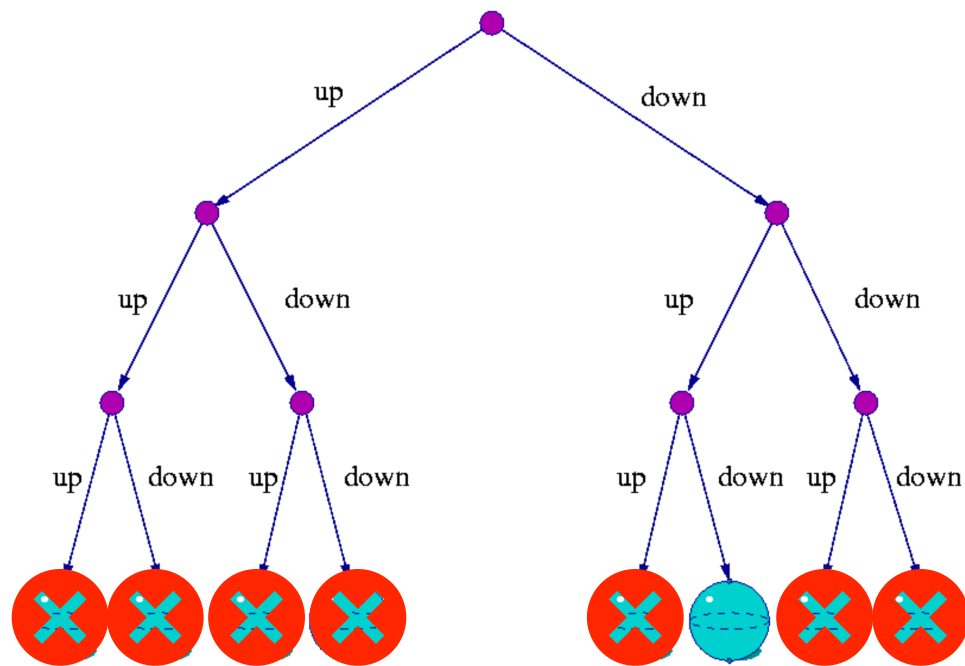
2.3.1 $i := 2$



ALGORITHMUS 5.30 (Quantum Bogosort)

INPUT : n Zahlen $A[1], \dots, A[n]$
 OUTPUT : Sortierter Array in geeignetem Universum

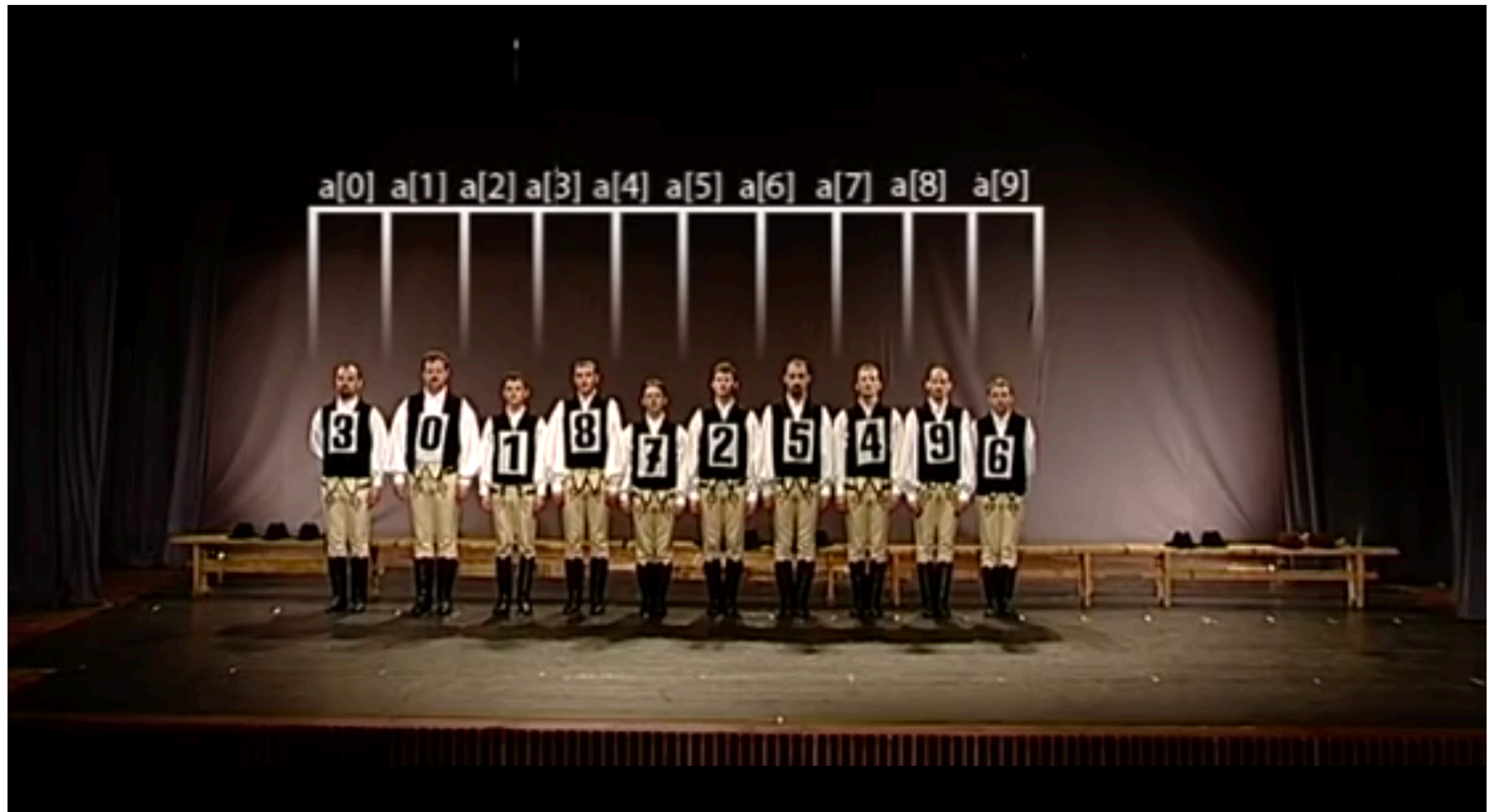
1. Permutiere zufällig
2. IF (Permutation unsortiert)
 - 2.1 Zerstöre Universum

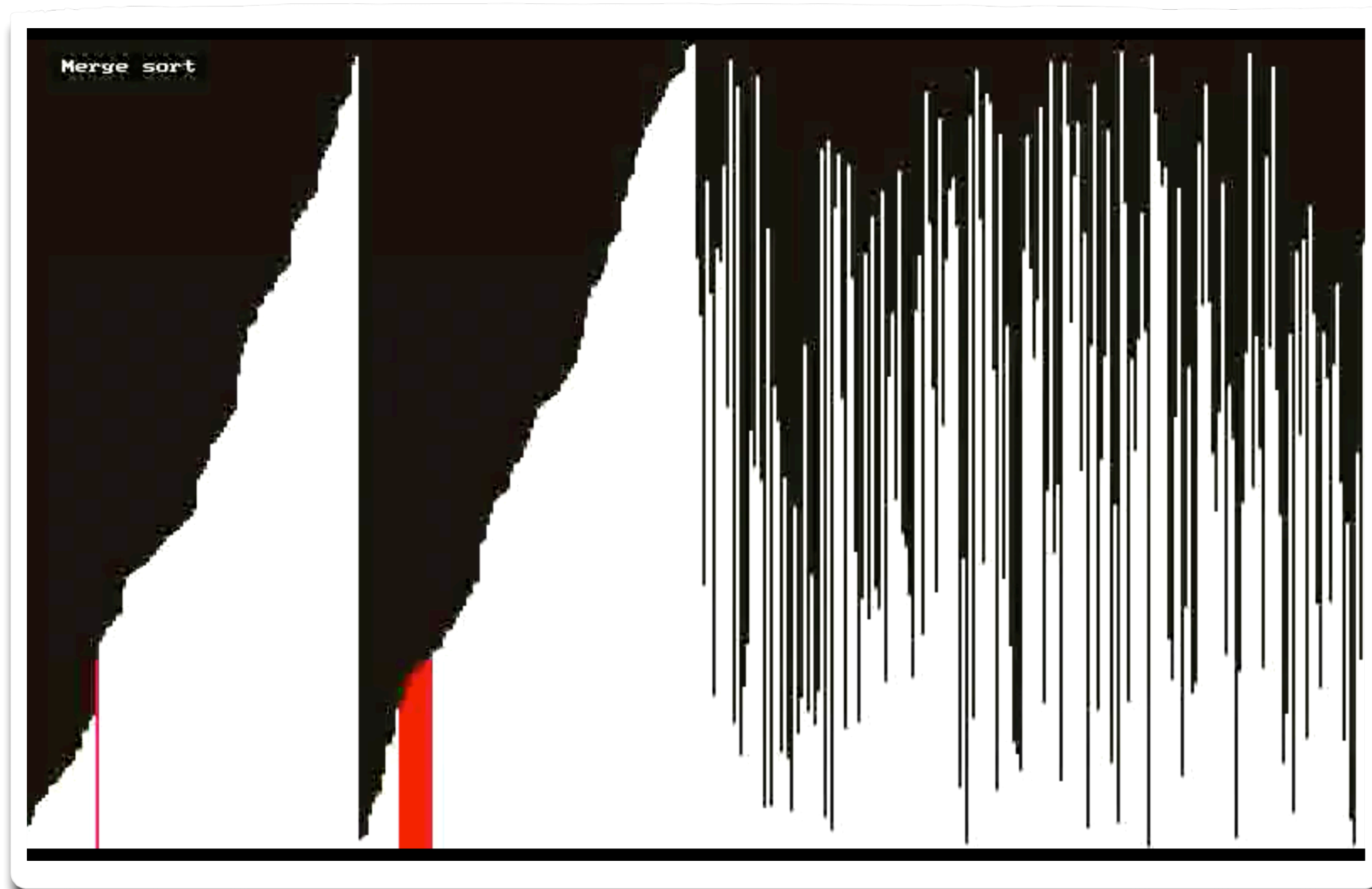


8	9	10	11	12	13	14	15	16
...	1	2	2	3	1	2	3	6
k								
2	3	4	5					
4	5	7	∞					
i								

1	2	3
1	2	3
R		

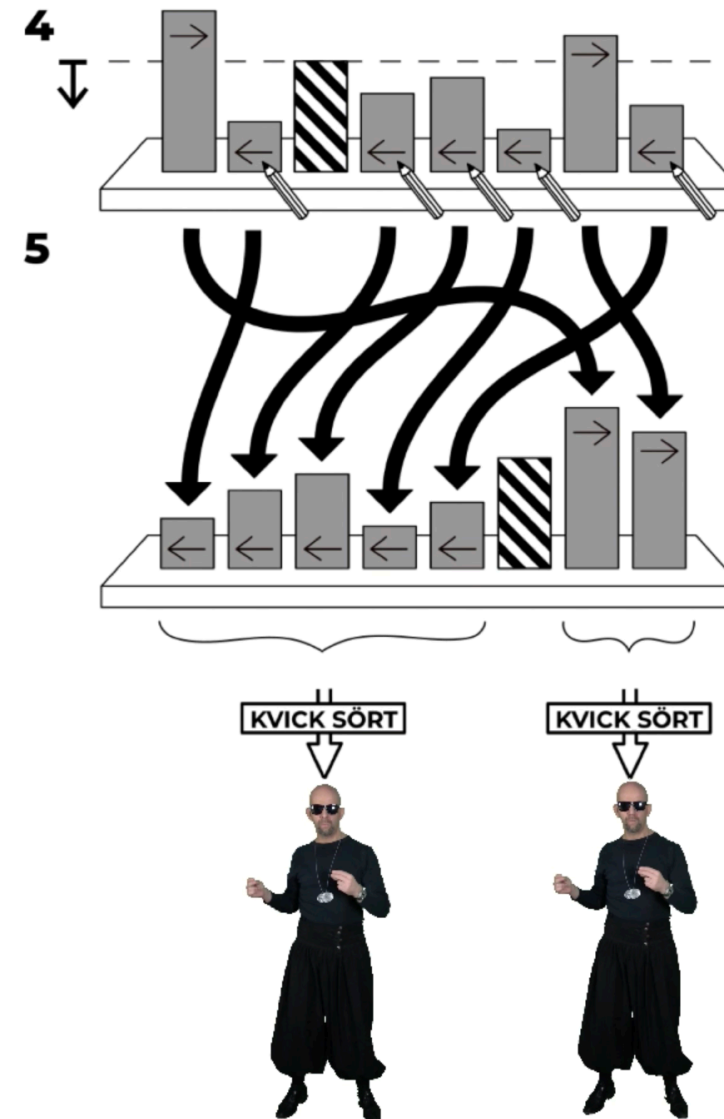
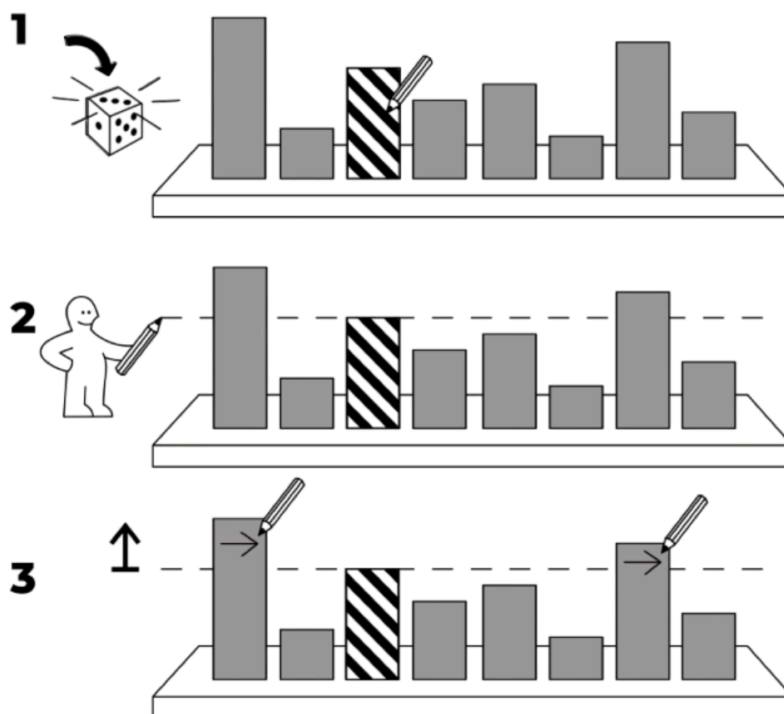
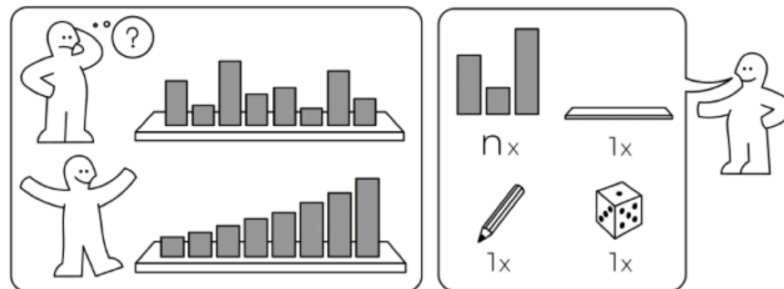






KVICK SÖRT

idea-instructions.com/quick-sort/
v1.2, CC by-nc-sa 4.0 **IDEA**



Ausblick!



Algorithmen und Datenstrukturen 2

Sommersemester 2021

[Startseite](#)

[Vorlesungen](#)

[Organisation](#)

[Kapitel ▾](#)

[Kontakt](#)

[Archiv](#)

Startseite

Startseite / Algorithmen und Datenstrukturen 2

Algorithmen und Datenstrukturen 2

Die Vorlesung **Algorithmen und Datenstrukturen 2** ist eine Wahlpflichtveranstaltung für Studierende der Informatik, Wirtschaftsinformatik, Informations- und Systemtechnik; außerdem ist sie offen für interessierte Studierende anderer Studiengänge.

Algorithmen sind das methodische Herz der theoretischen und praktischen Informatik; Datenstrukturen ermöglichen die effiziente Umsetzung von Algorithmen und den effizienten Zugriff auf Input- und Outputdaten. In dieser weiterführenden Vorlesung werden die folgenden grundlegenden Begriffe erarbeitet:

- Elementare Aspekte zu Heuristiken
- Exakte Verfahren: Dynamic Programming, Branch-and-Bound
- Approximationsalgorithmen
- Komplexitätsaspekte

Neu

He

Voraussichtl.
Veranstaltung

Bitte melden
Wir nutzen
Informations-
sowie m
Braunschweig



Technische
Universität
Braunschweig



Vielen Dank!

s.fekete@tu-bs.de