

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	...	1	2	3		
$i$								
				$R$	1	2	3	



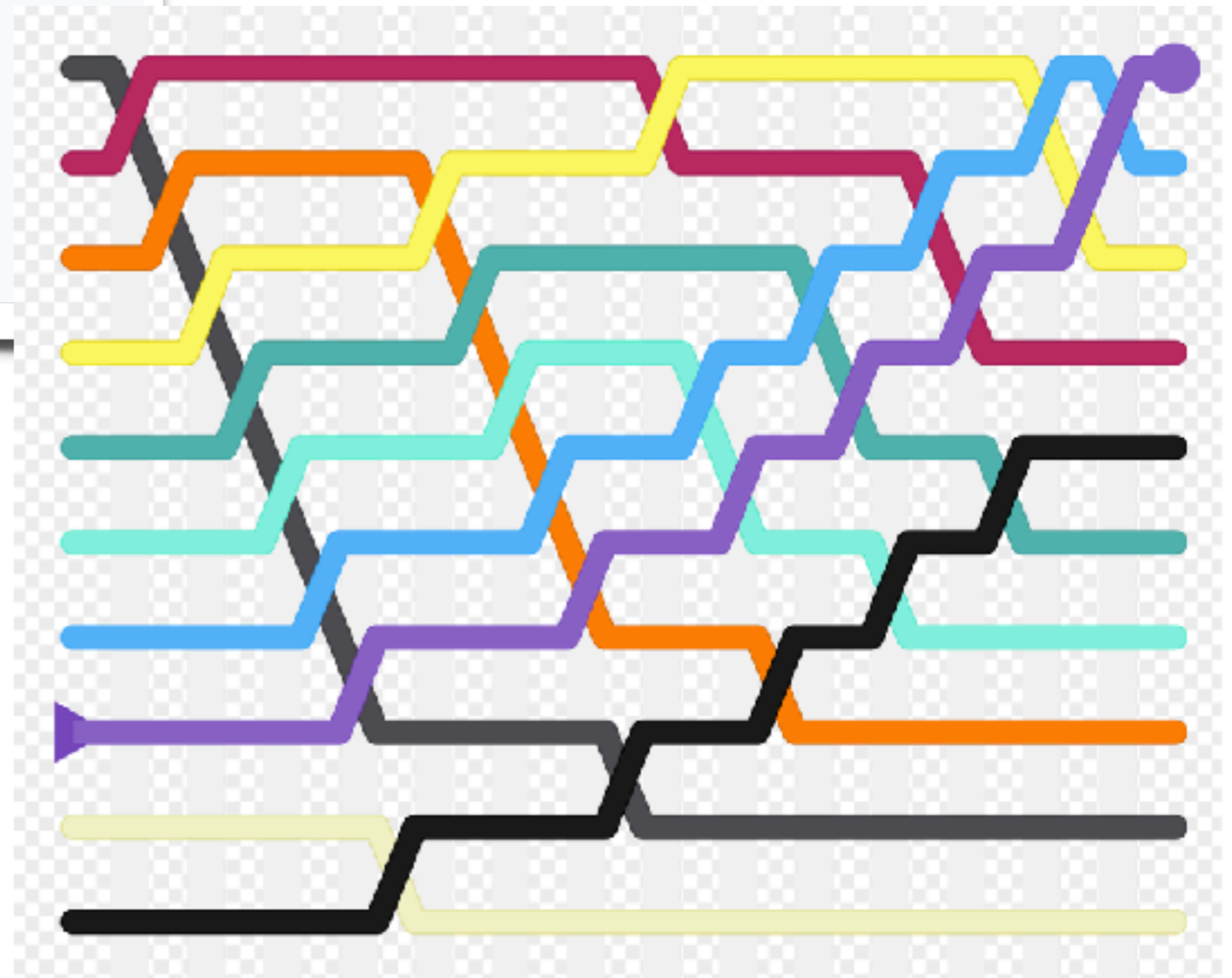
*Kapitel 5.9:*  
*Paralleles Sortieren*  
*Algorithmen und Datenstrukturen*  
*WS 2020/21*

**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
```

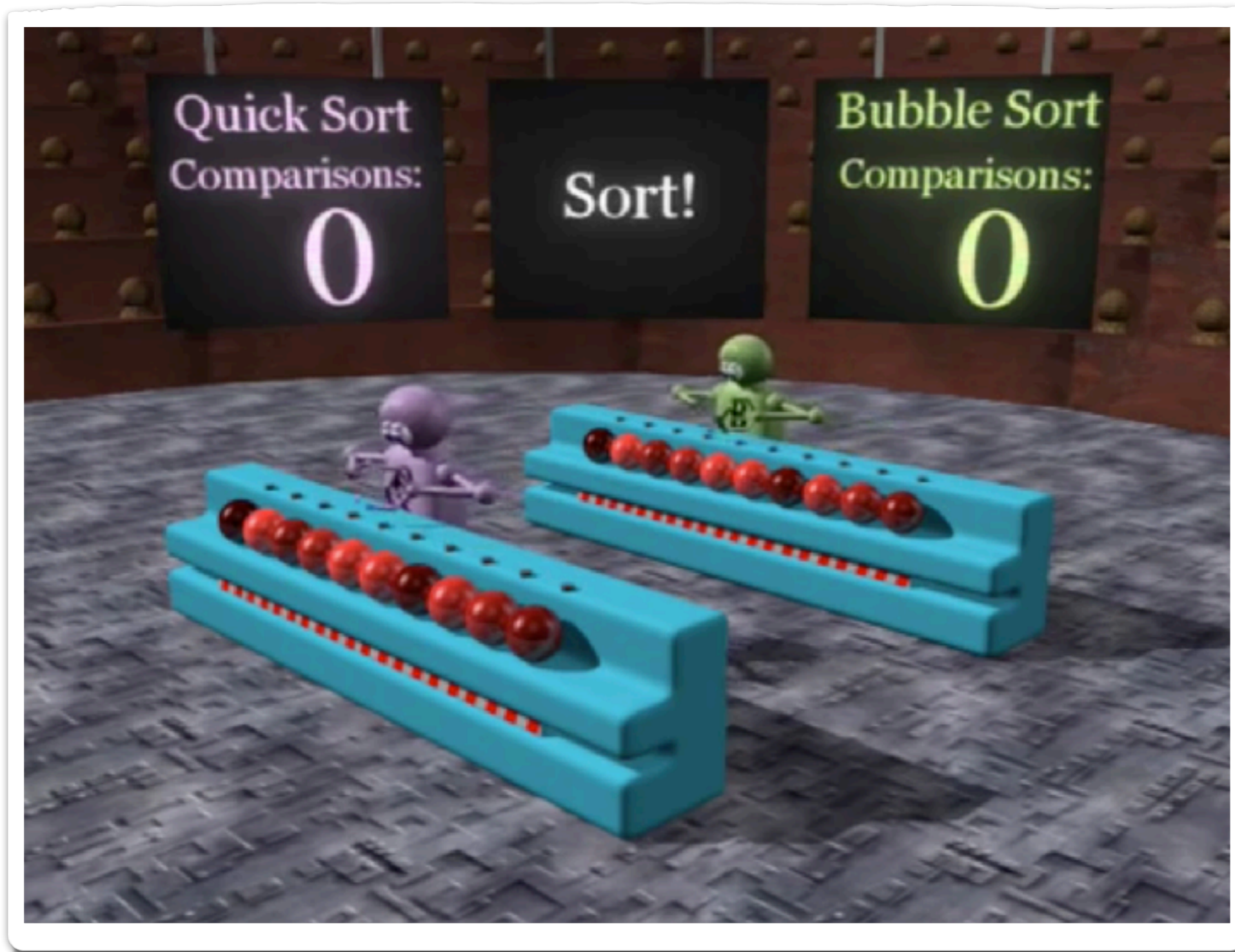
```
bubbleSort(Array A)
  for (n=A.size; n>1; --n){
    for (i=0; i<n-1; ++i){
      if (A[i] > A[i+1]){
        A.swap(i, i+1)
      } // Ende if
    } // Ende innere for-Schleife
  } // Ende äußere for-Schleife
```





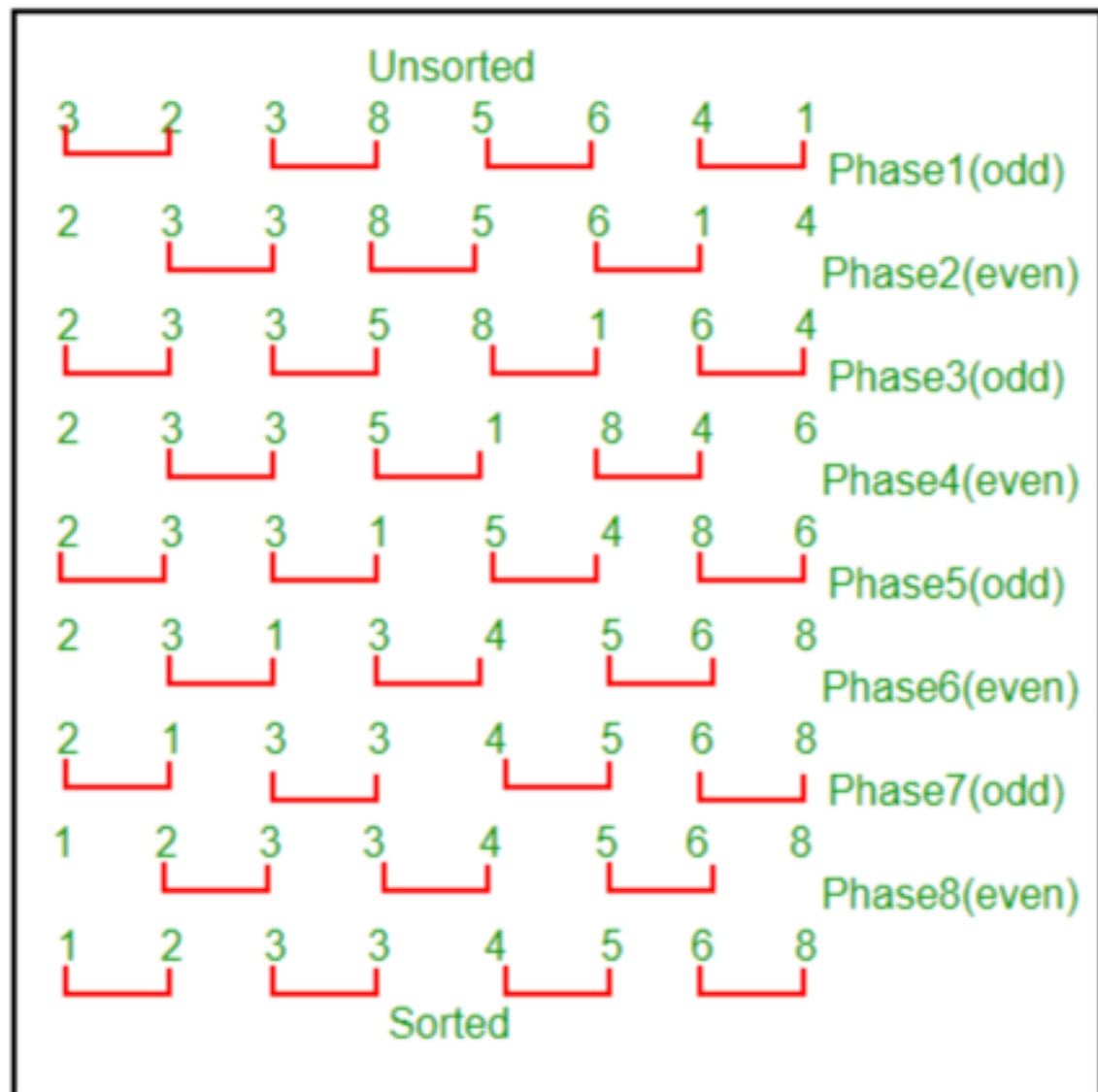




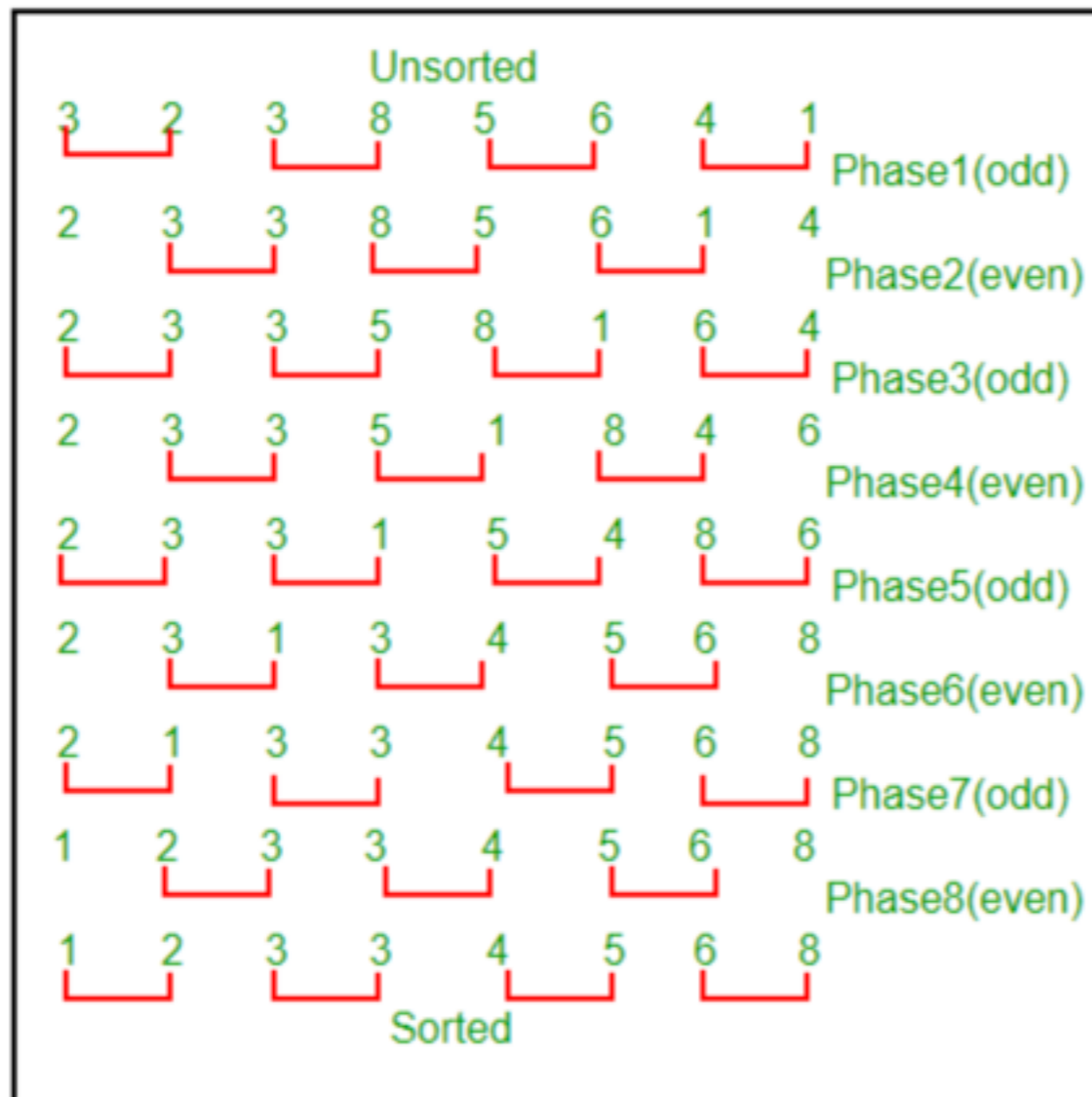




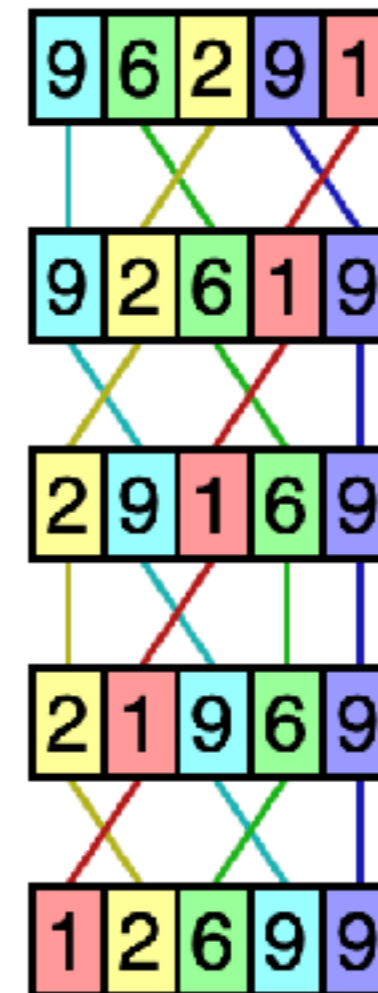




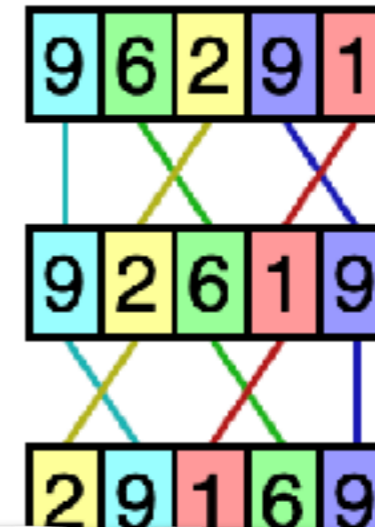
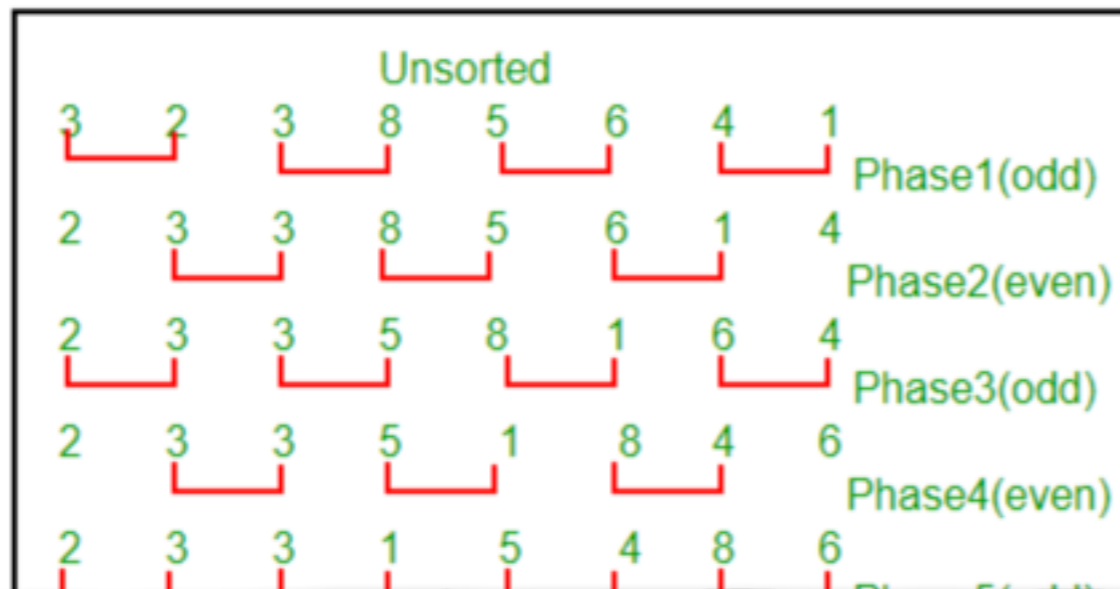
geeksforgeeks



geeksforgeeks



growingwiththeweb

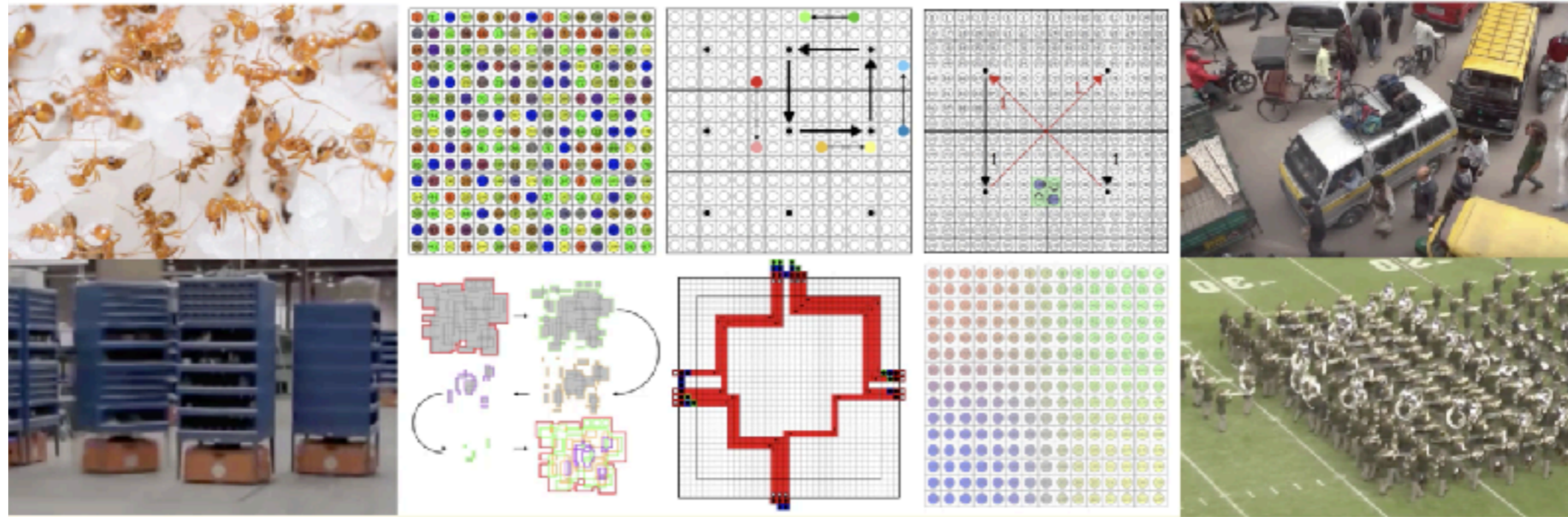


Satz 5.25

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**

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	...	1	2	3		
$i$				$R$	1	2	3	



*Kapitel 5.10:*  
*Analoge Verfahren*  
*Algorithmen und Datenstrukturen*  
*WS 2020/21*

**Prof. Dr. Sándor Fekete**





## 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 extenuating myself from a vast heap of wood boards, rubber bands, strings, balls of polystyrene, fish tanks, lead weights, canisters, tubing and stop-cocks. 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 *s.g.*, 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 *s.g.*, 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:

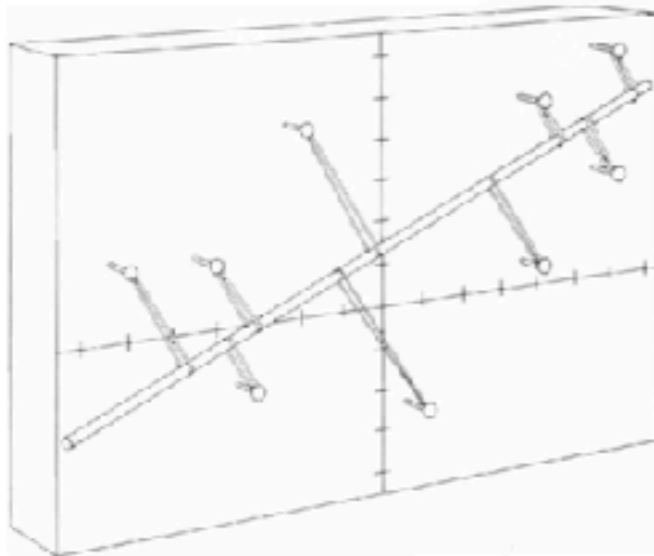
Plot the data points on a wood surface and drive a nail partway into the wood at each point. Next, slip a number of uniform rubber bands onto the rod, one band for each nail. Fit the rod approximately into place and pull each band over one of the nails. When the rod is released, it wiggles and shivers 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 Wassenmaker 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 Wassenmaker 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 27].

When I initially encountered Wassenmaker'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-

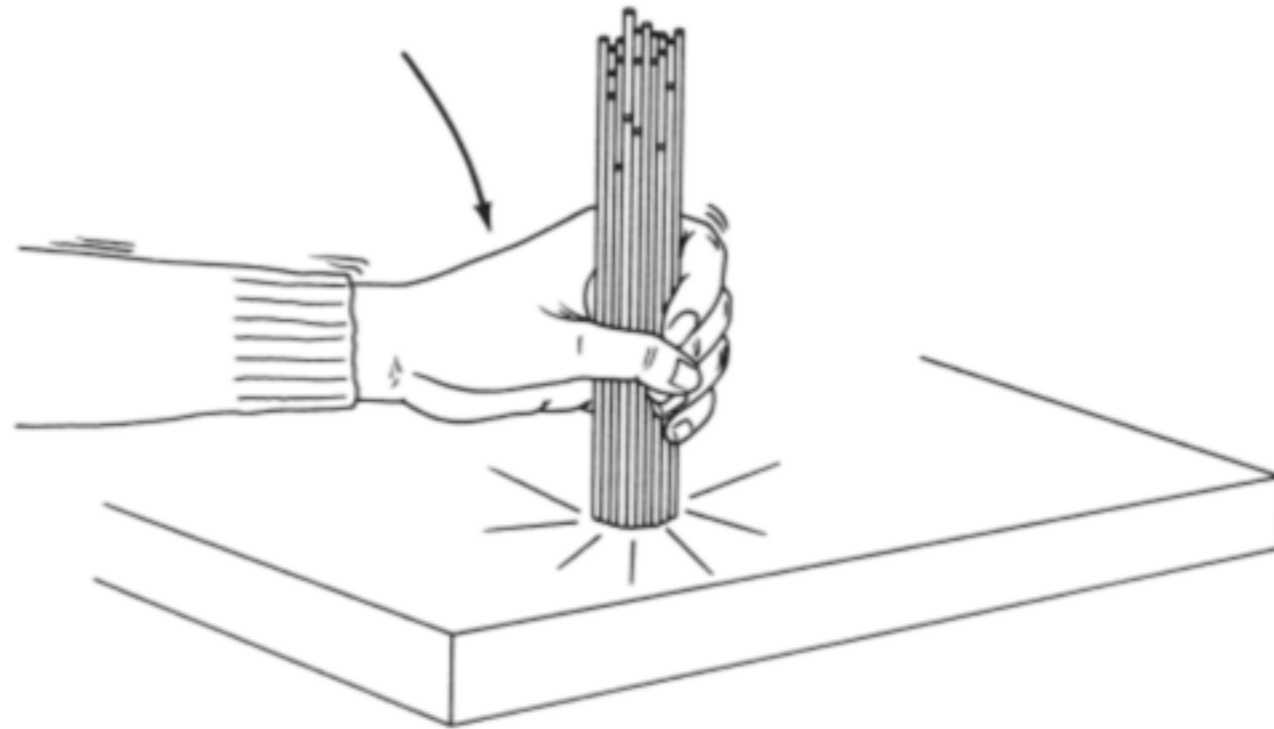


A gadget for finding the line that best fits a series of data points

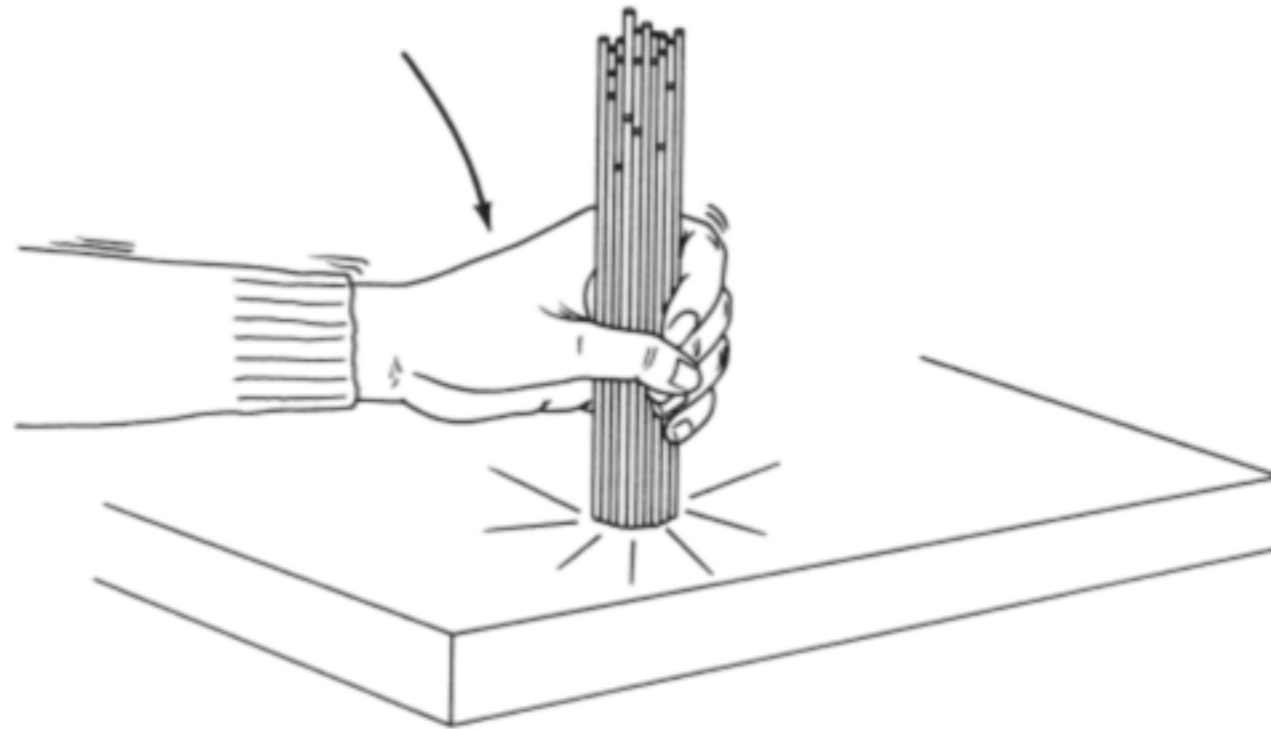


1. For each number, break off a piece of spaghetti, whose length equals the number. This takes linear time.

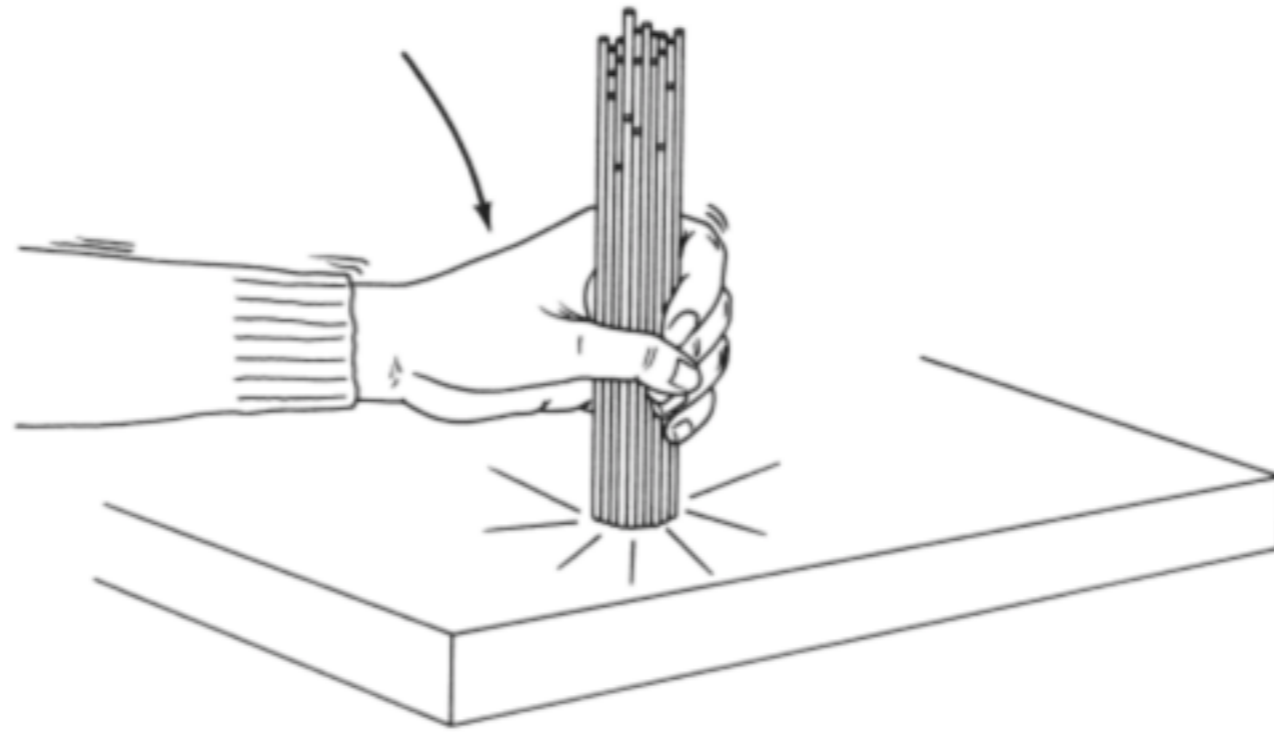
1. For each number, break off a piece of spaghetti, whose length equals the number. This takes linear time.
2. Take all spaghetti in your fist, and slam their lower sides on the table, so they all point up starting at the table. This takes constant time, thanks to the multi-processing properties of our universe.



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.



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.

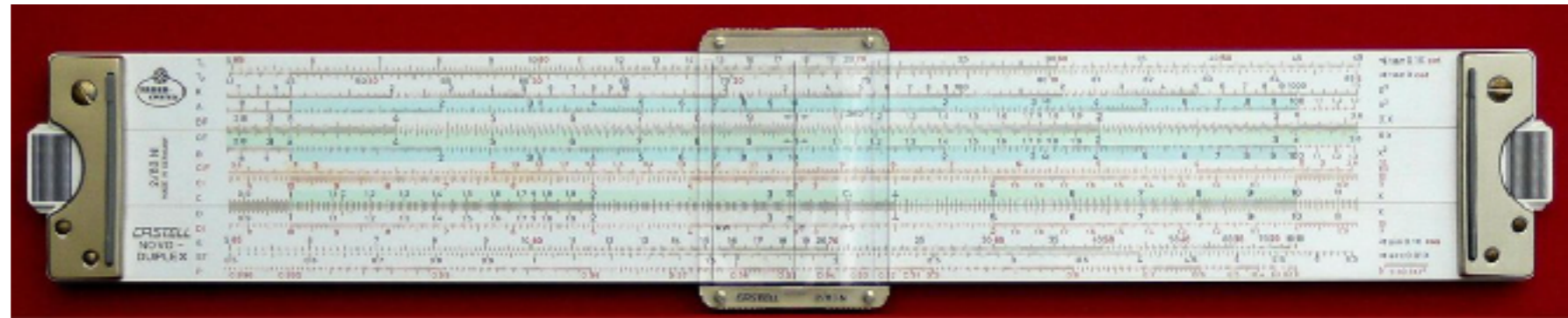


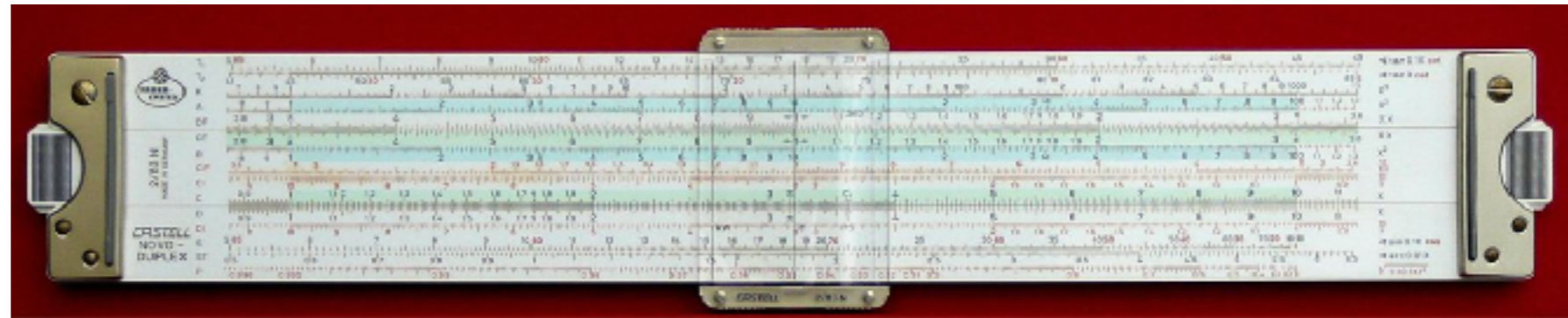
1. For each number, break off a piece of spaghetti, whose length equals the number. This takes linear time.
2. Take all spaghetti in your fist, and slam their lower sides on the table, so they all point up starting at the table. This takes constant time, thanks to the multi-processing properties of our 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!

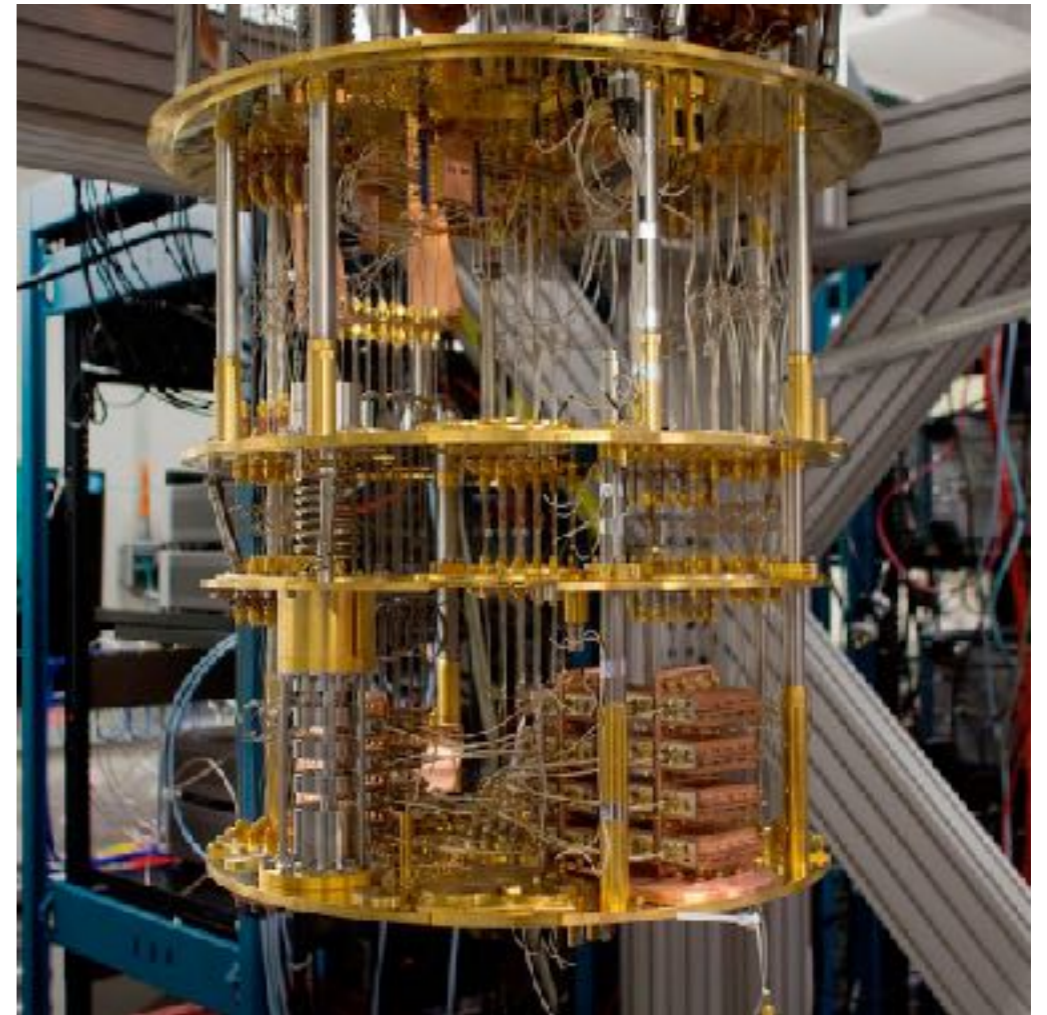
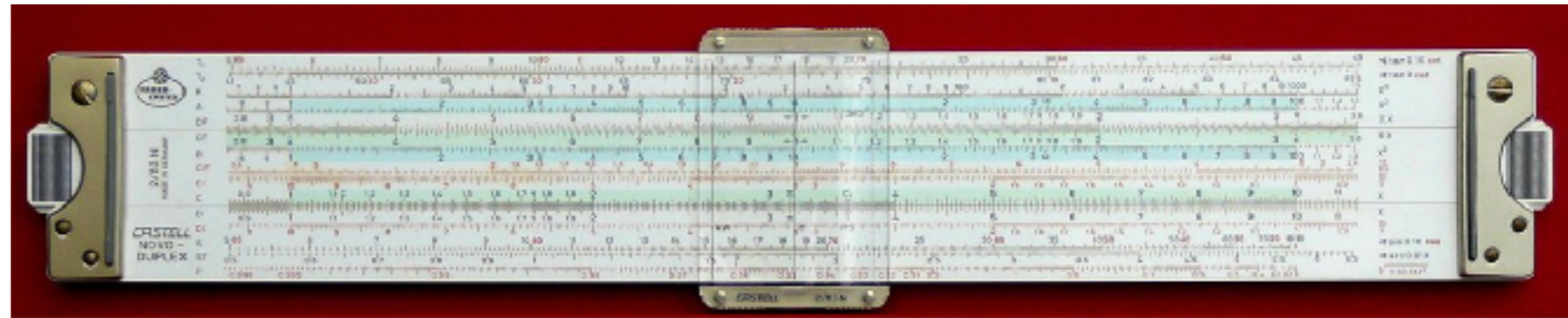


















6,022 140 76 · 10<sup>23</sup> Teilchen

# Schnelles und genaues Zählen?



6,022 140 76 · 10<sup>23</sup> Teilchen







# Schnelles und genaues Zählen?



6,022 140 76 · 10<sup>23</sup> Teilchen



1.000.000.000 Teilchen pro Sekunde  
1.000 Zähler

---



# Schnelles und genaues Zählen?



6,022 140 76 · 10<sup>23</sup> Teilchen



1.000.000.000 Teilchen pro Sekunde

---

1.000 Zähler

über 19.000 Jahre

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



*Kapitel 5.11:*  
*Unernste Verfahren*  
*Algorithmen und Datenstrukturen*  
*WS 2020/21*

**Prof. Dr. Sándor Fekete**



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



## BOGO SÖRT

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

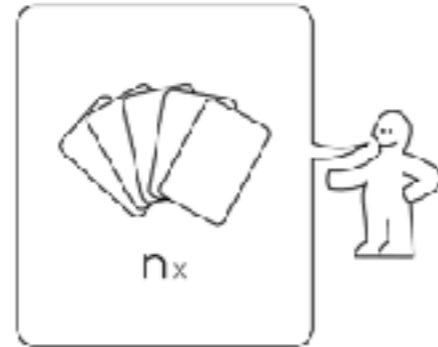
**IDEA**



## BOGO SÖRT

[idea-instructions.com/bogo-sort/](http://idea-instructions.com/bogo-sort/)  
v1.2, CC by-nc-sa 4.0

**IDEA**

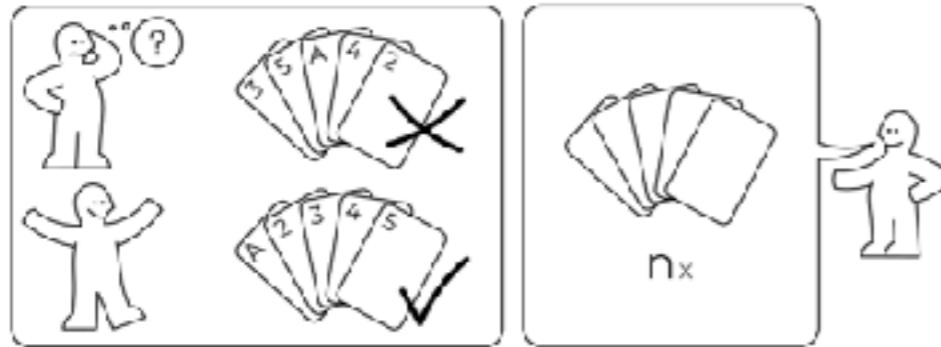




## BOGO SÖRT

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

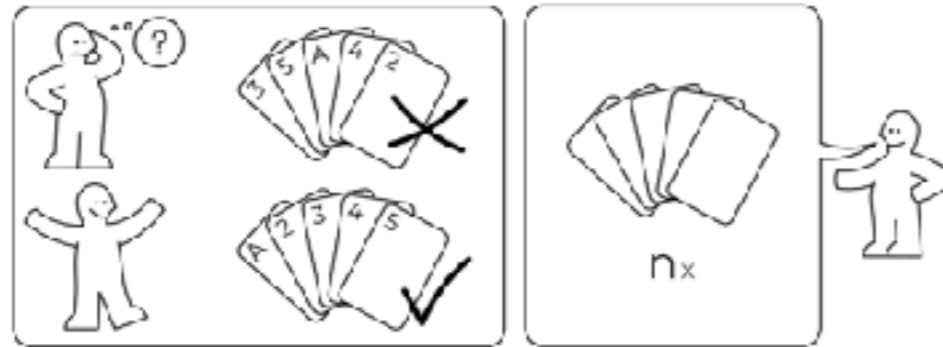
**IDEA**



## BOGO SÖRT

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

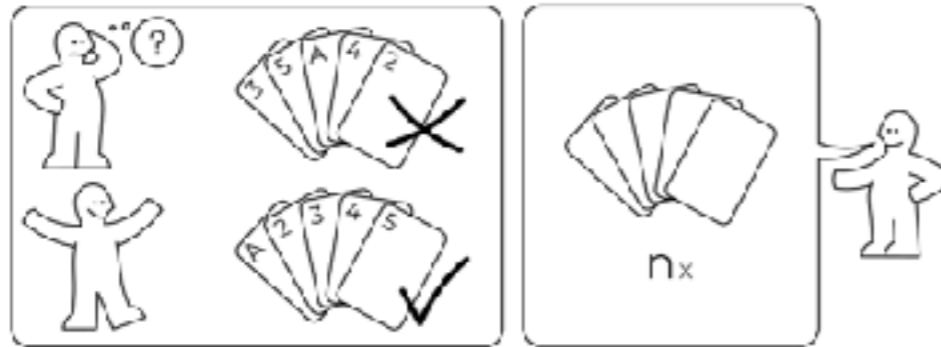
**IDEA**



## BOGO SÖRT

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

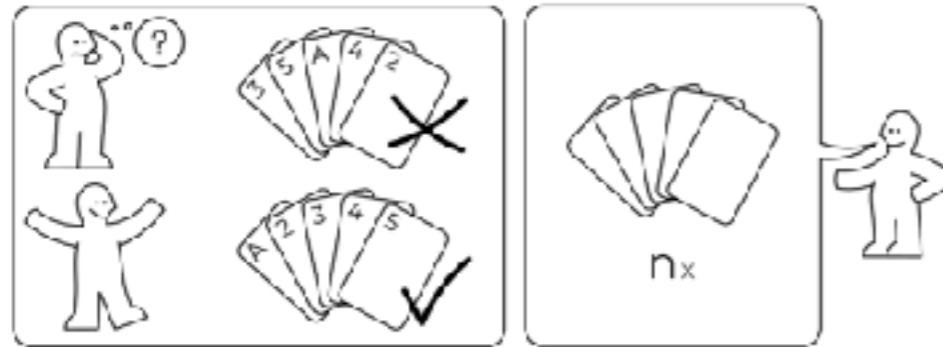
**IDEA**



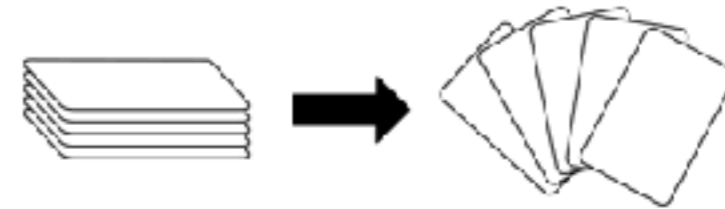
## BOGO SÖRT

[idea-instructions.com/bogo-sort/](http://idea-instructions.com/bogo-sort/)  
v1.2, CC by-nc-sa 4.0

**IDEA**



**3**



**1**

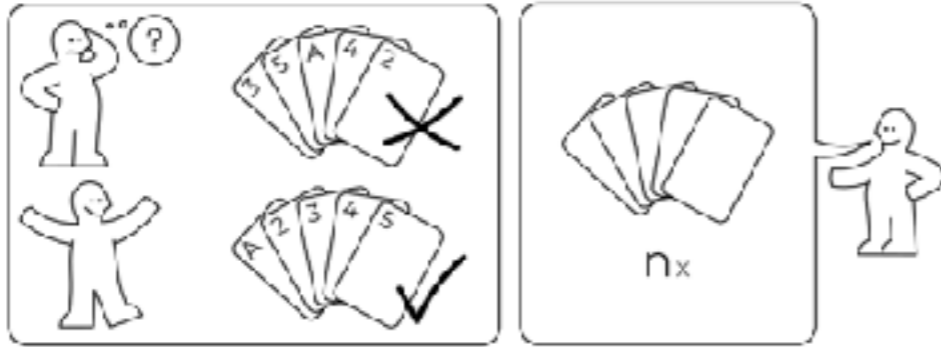


**2**



# BOGO SÖRT

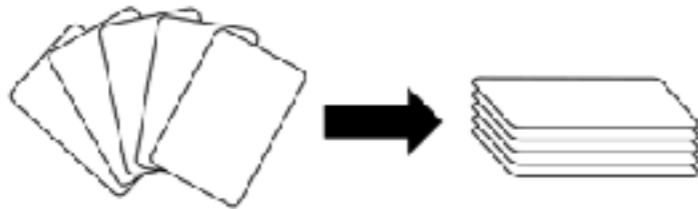
idea-instructions.com/bogo-sort/  
v1.2, CC by-nc-sa 4.0 **IDEA**



**3**



**1**



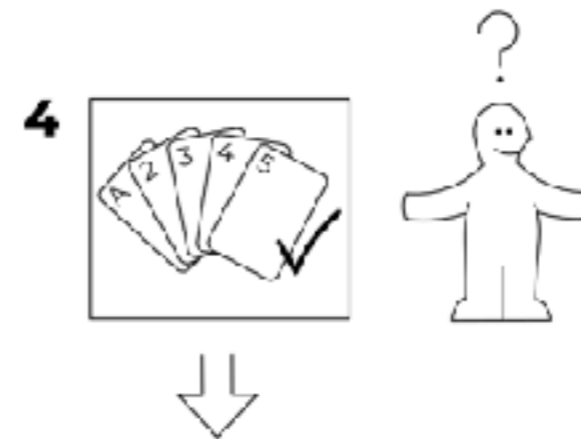
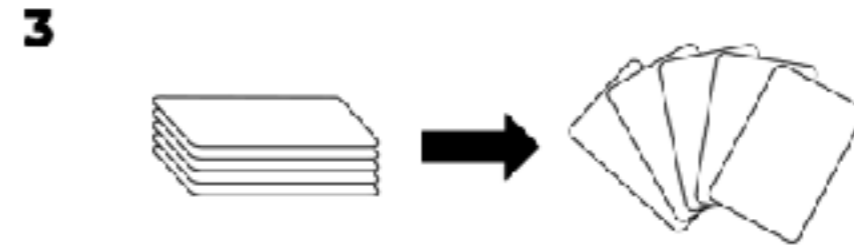
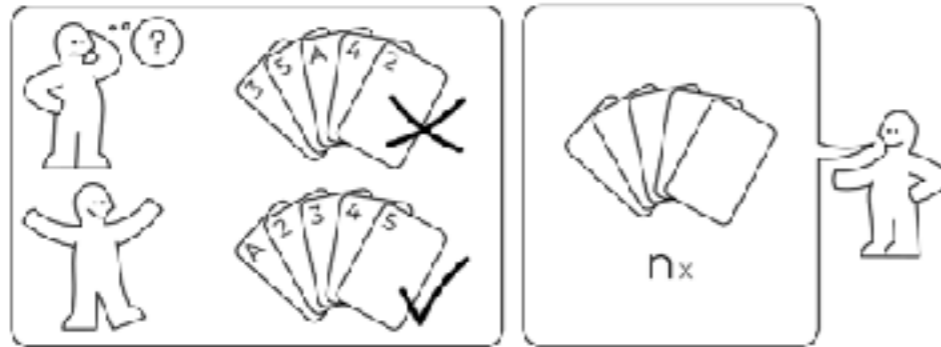
**2**



## BOGO SÖRT

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

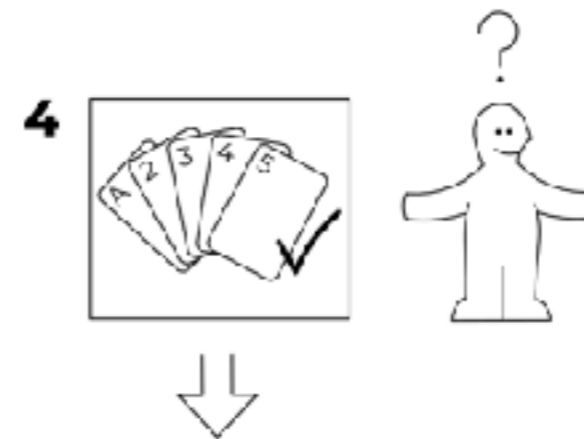
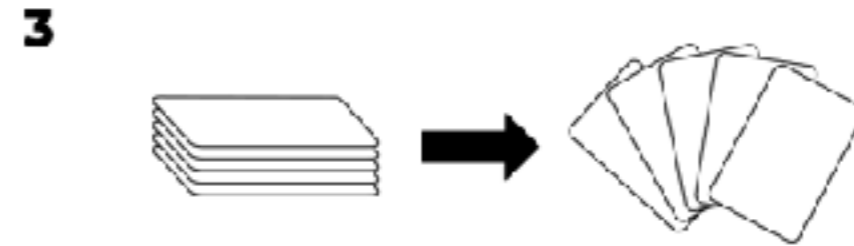
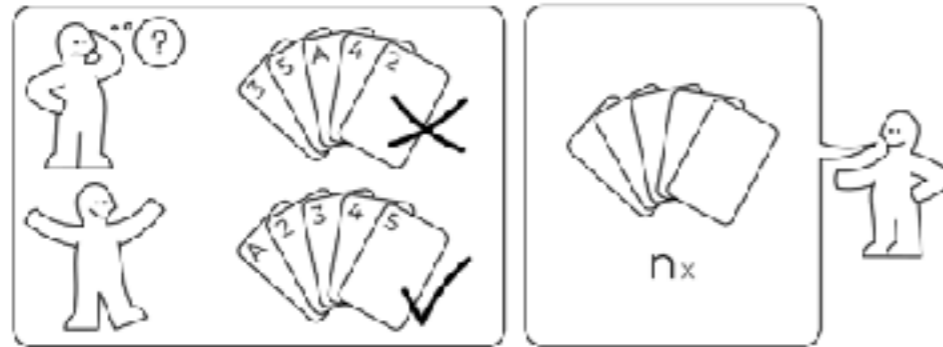
**IDEA**



## BOGO SÖRT

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

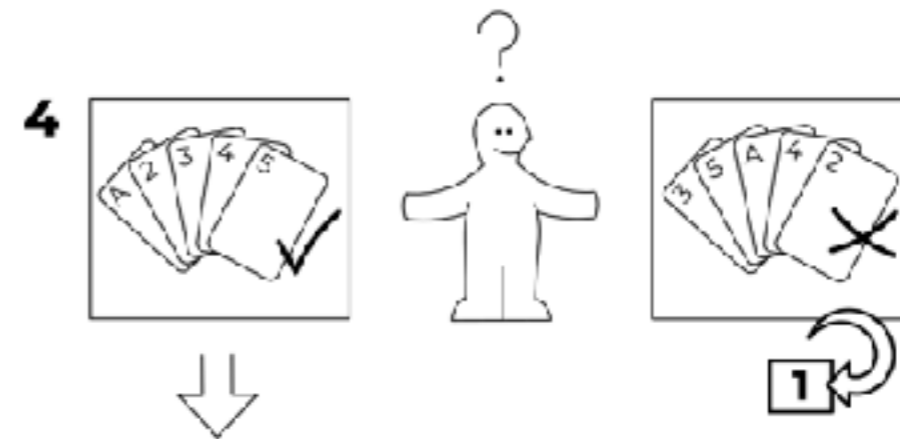
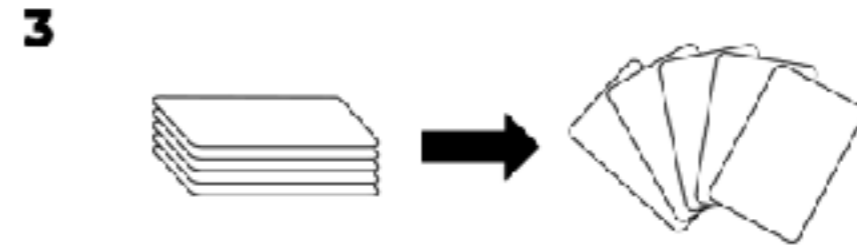
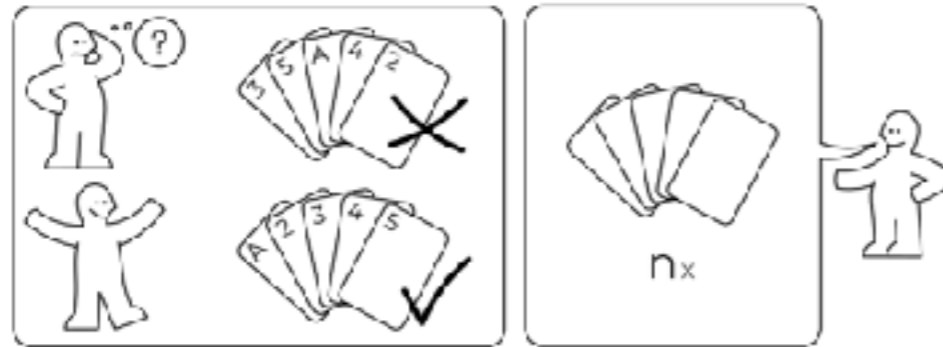
**IDEA**



## BOGO SÖRT

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

**IDEA**

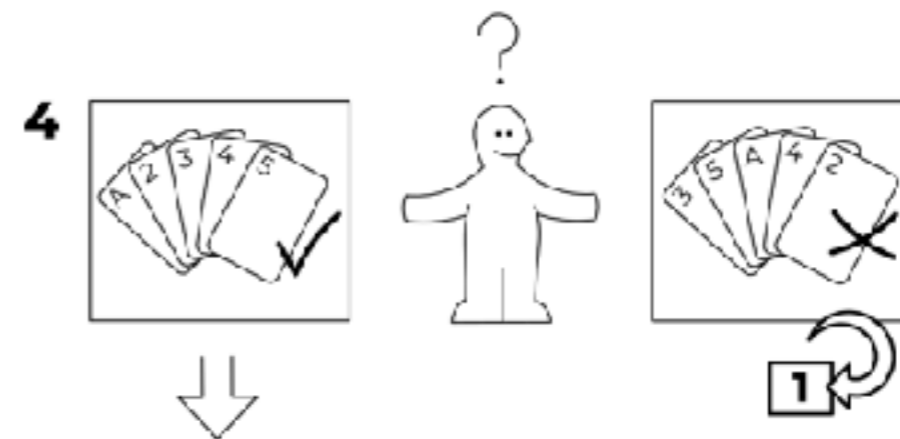
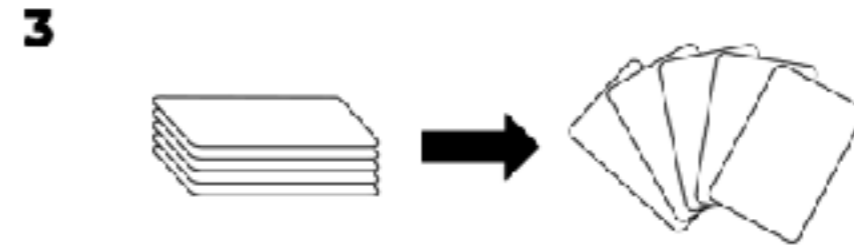
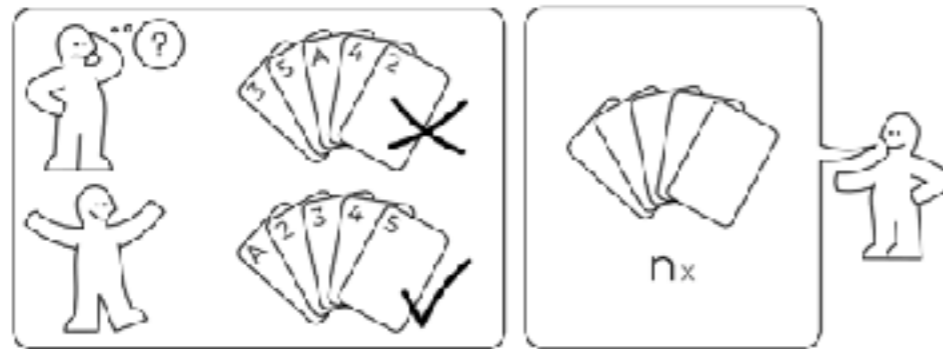




## BOGO SÖRT

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

ALGORITHMUS S.28 (Bogobogosort)

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

OUTPUT: Sortierter Array

1.  $i = 2$

ALGORITHMUS 5.28 (Bogobogosort)INPUT:  $n$  Zahlen  $A[1], \dots, A[n]$ 

OUTPUT: Sortierter Array

1.  $i := 2$ 2. WHILE ( $i \leq n$ ) DO2.1 wähle Zufallspermutation der Elemente  $A[1], \dots, A[i]$

ALGORITHMUS 5.28 (Bogobogosort)INPUT:  $n$  Zahlen  $A[1], \dots, A[n]$ 

OUTPUT: Sortierter Array

1.  $i := 2$ 2. WHILE ( $i \leq n$ ) DO2.1 Wähle Zufallspermutation der Elemente  $A[1], \dots, A[i]$ 

2.2 IF (Permutation ist sortiert)

2.2.1  $i := i + 1$

ALGORITHMUS 5.28 (Bogobogosort)INPUT:  $n$  Zahlen  $A[1], \dots, A[n]$ 

OUTPUT: Sortierter Array

1.  $i := 2$ 2. WHILE ( $i \leq n$ ) DO2.1 Wähle Zufallspermutation der Elemente  $A[1], \dots, A[i]$ 

2.2 IF (Permutation ist sortiert)

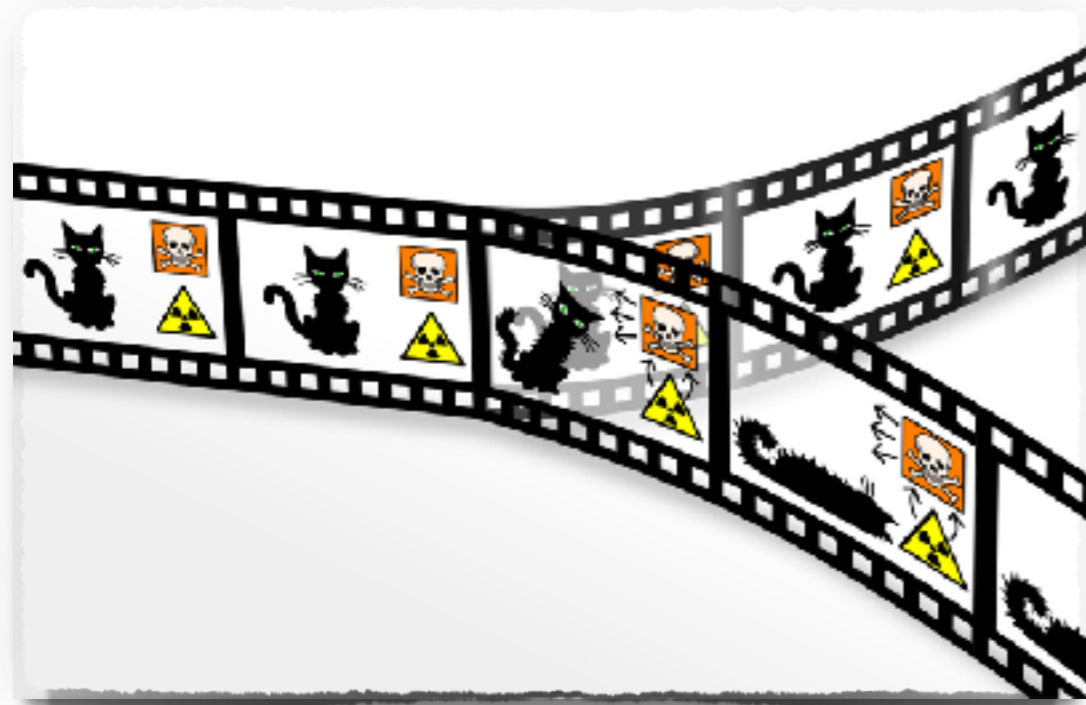
2.2.1  $i := i + 1$ 

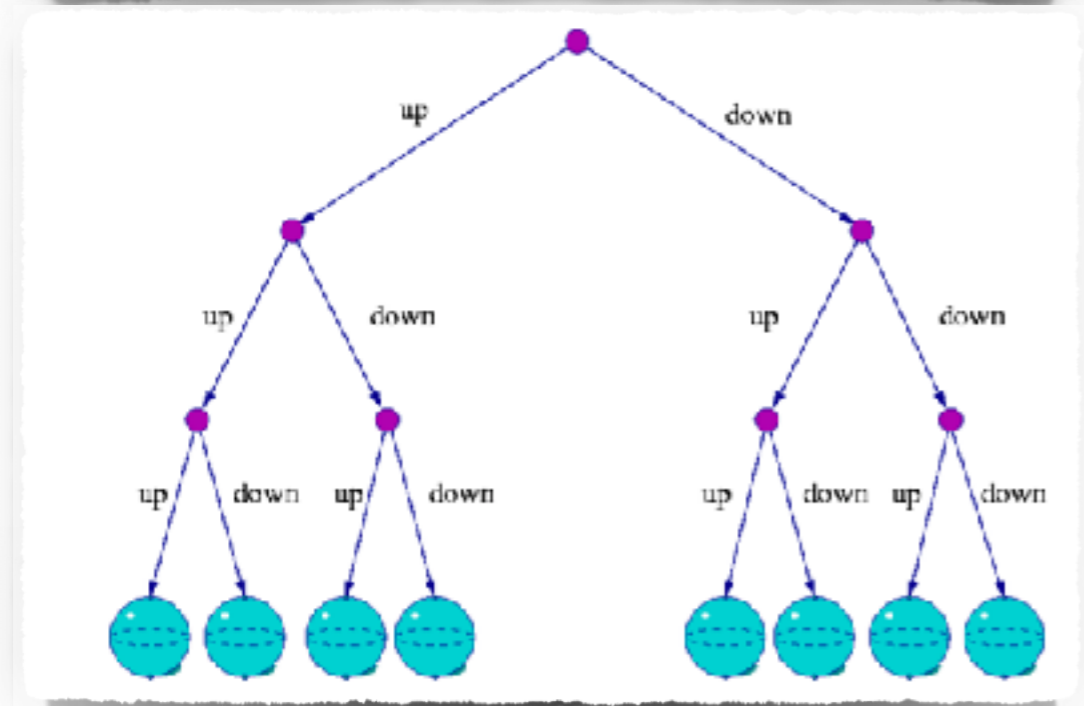
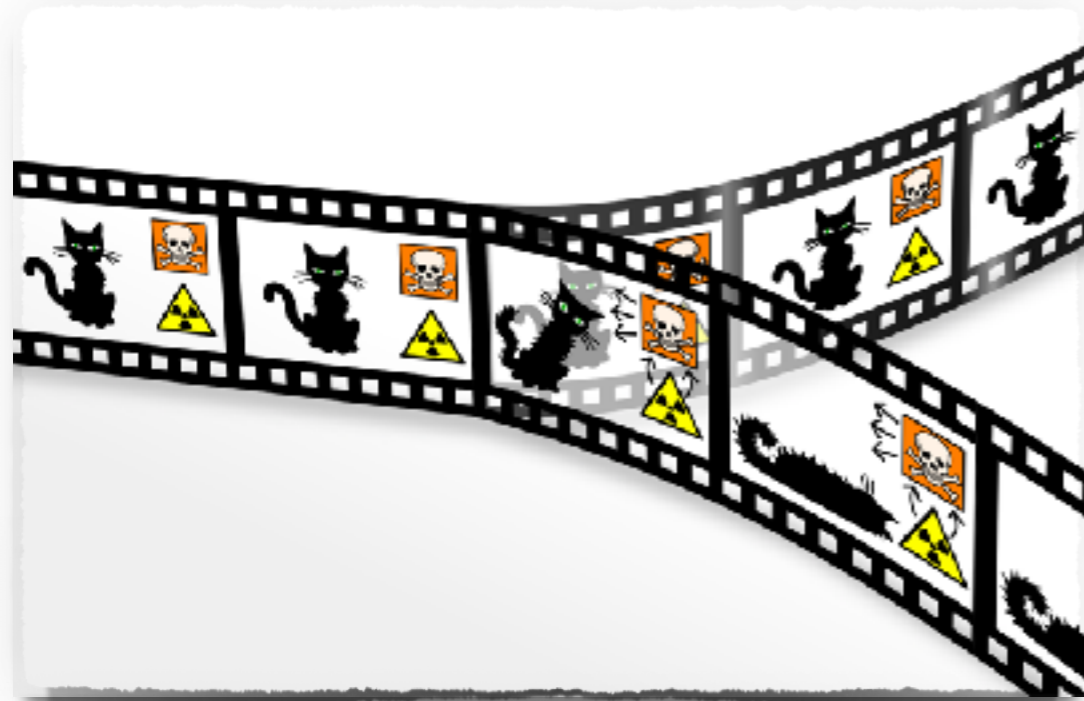
2.3 ELSE

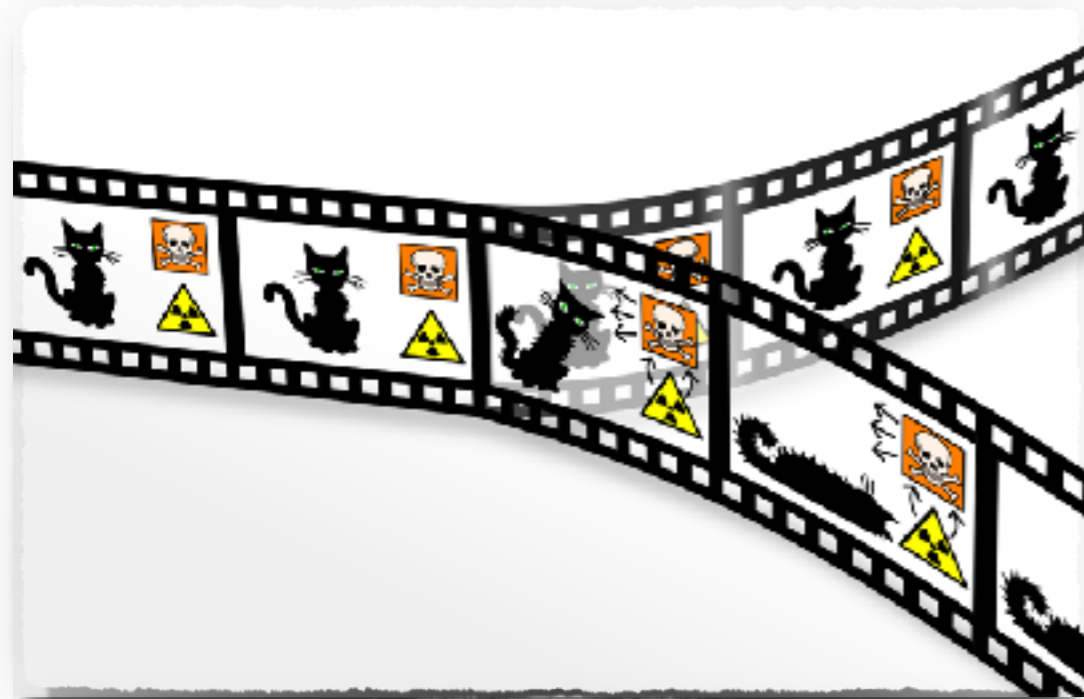
2.3.1  $i := 2$





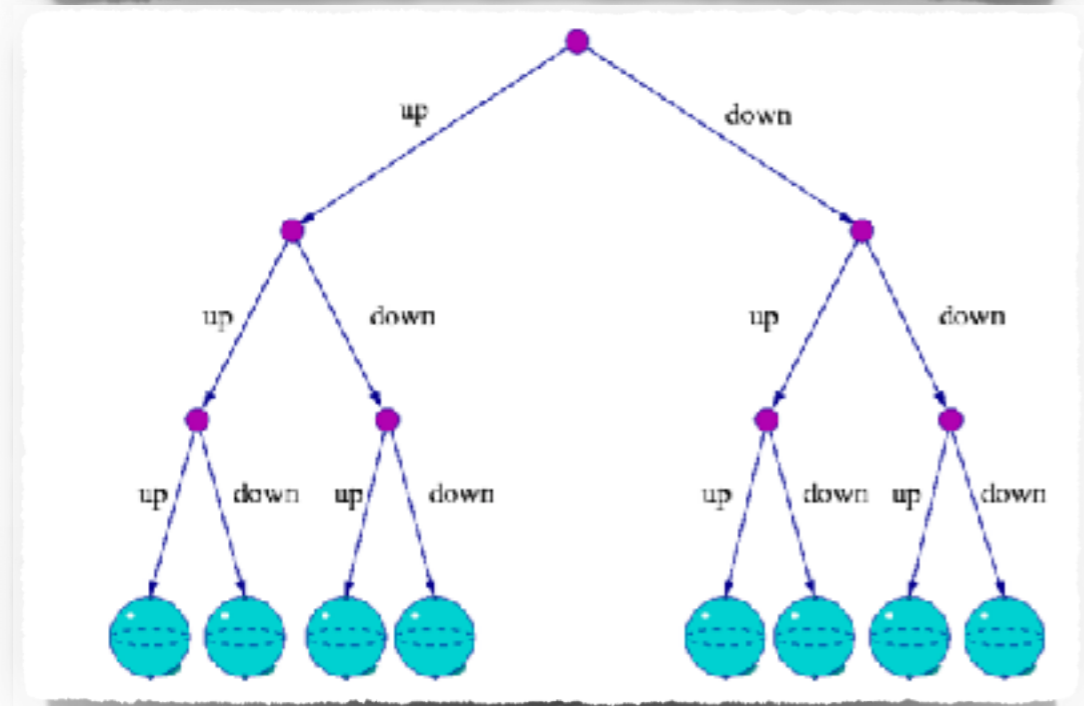


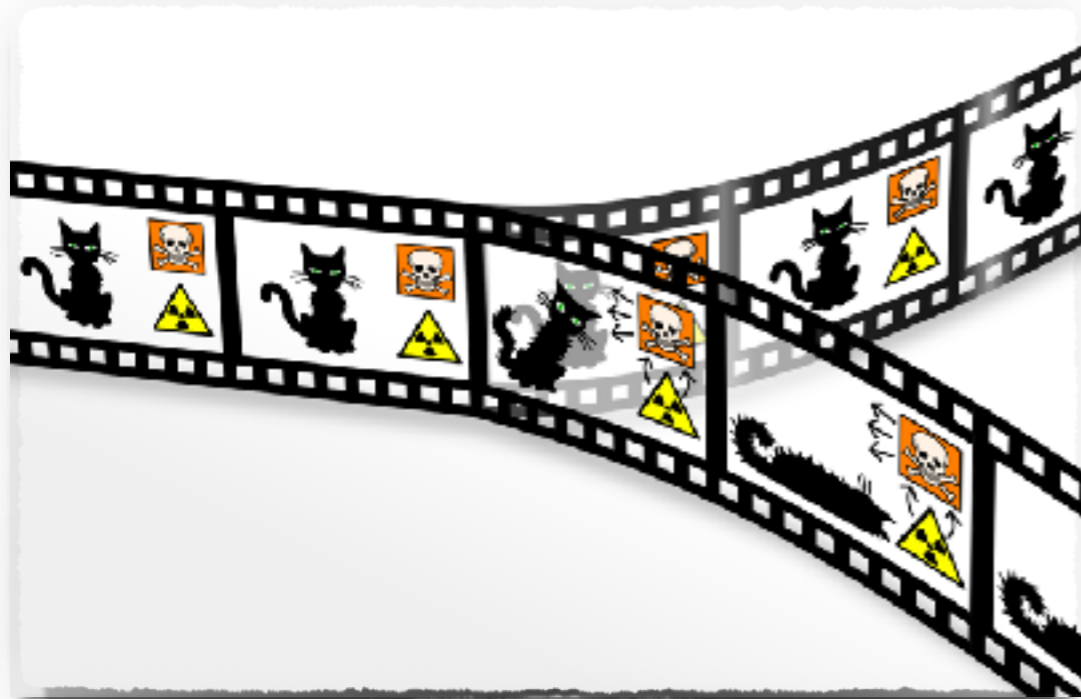




ALGORITHMUS 5.30 (Quantum Bogosort)

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

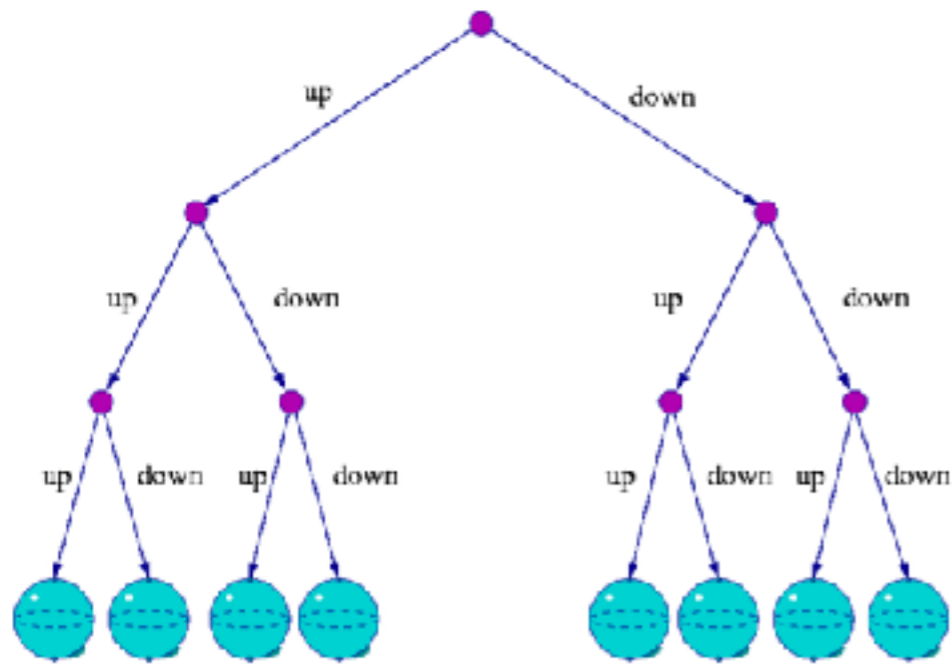


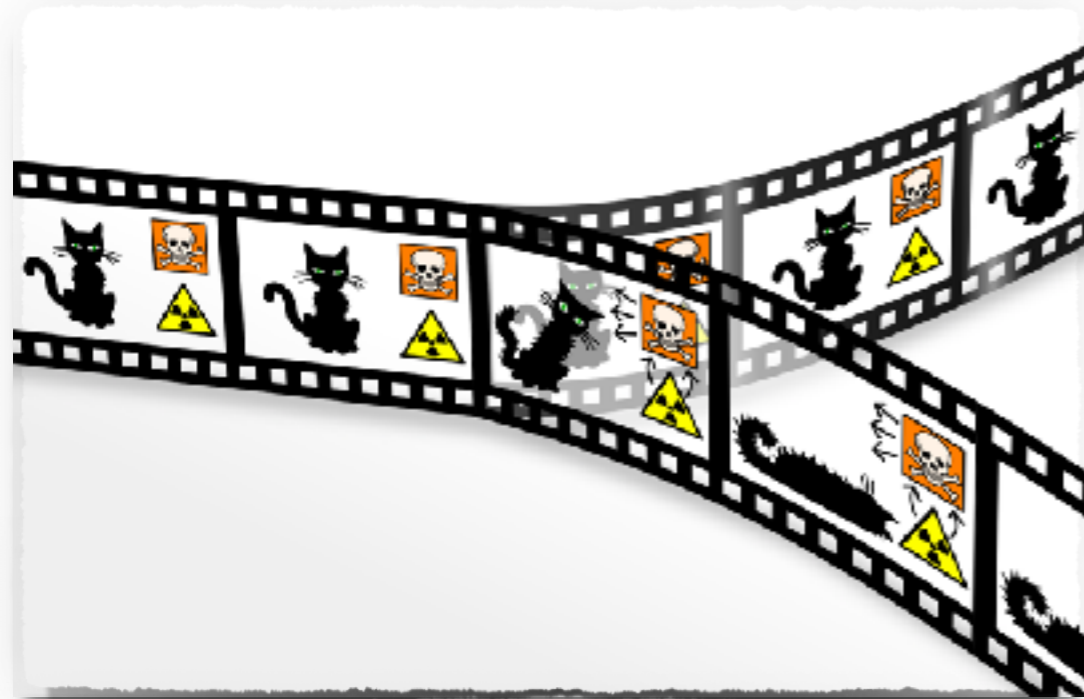


ALGORITHMUS 5.30 (Quantum Bogosort)

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

1. Permutiere zufällig

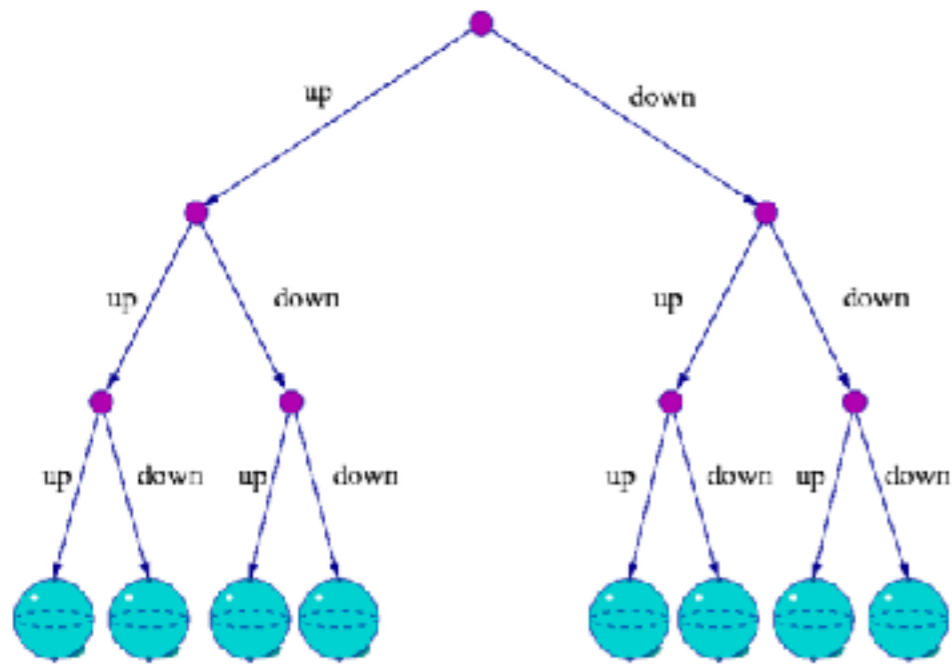


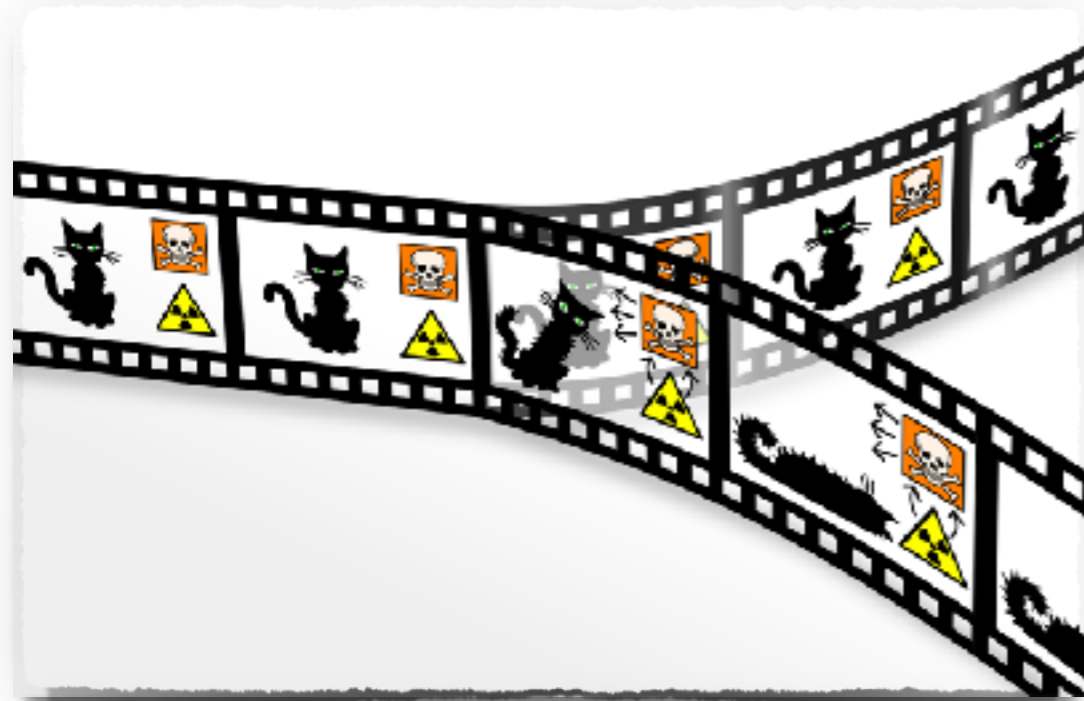


ALGORITHMUS 5.30 (Quantum Bogosort)

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

1. Permutiere zufällig
2. IF (Permutation unsortiert)



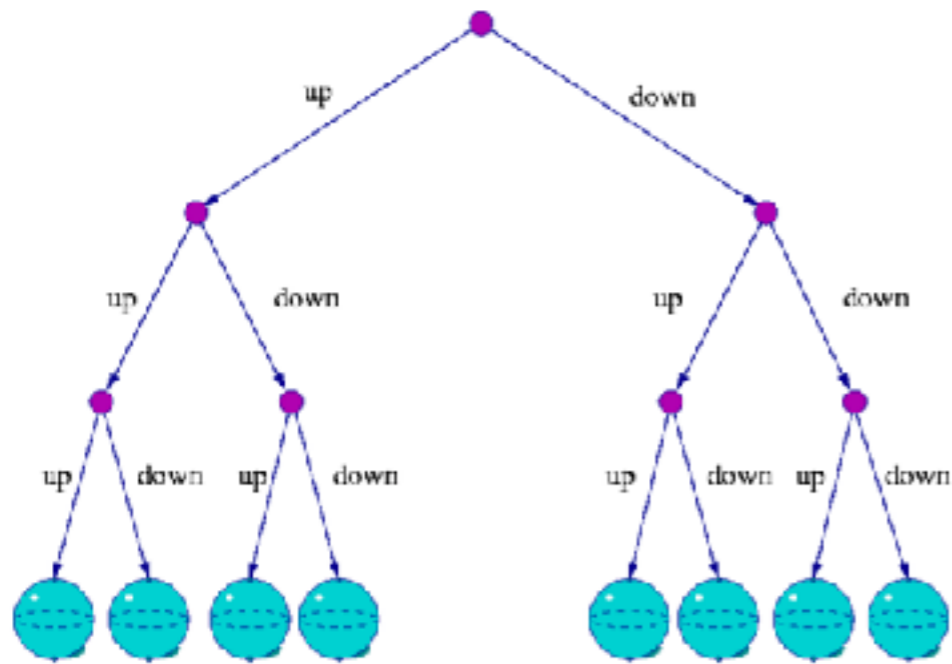


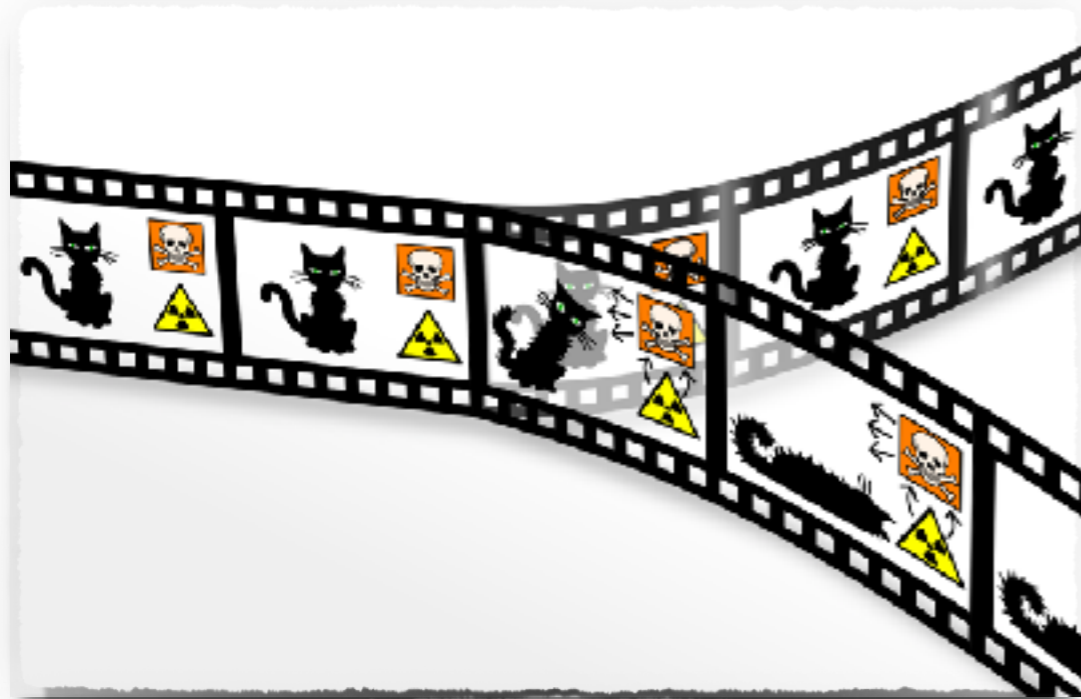
## ALGORITHMUS 5.30 (Quantum Bogosort)

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

OUTPUT :

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

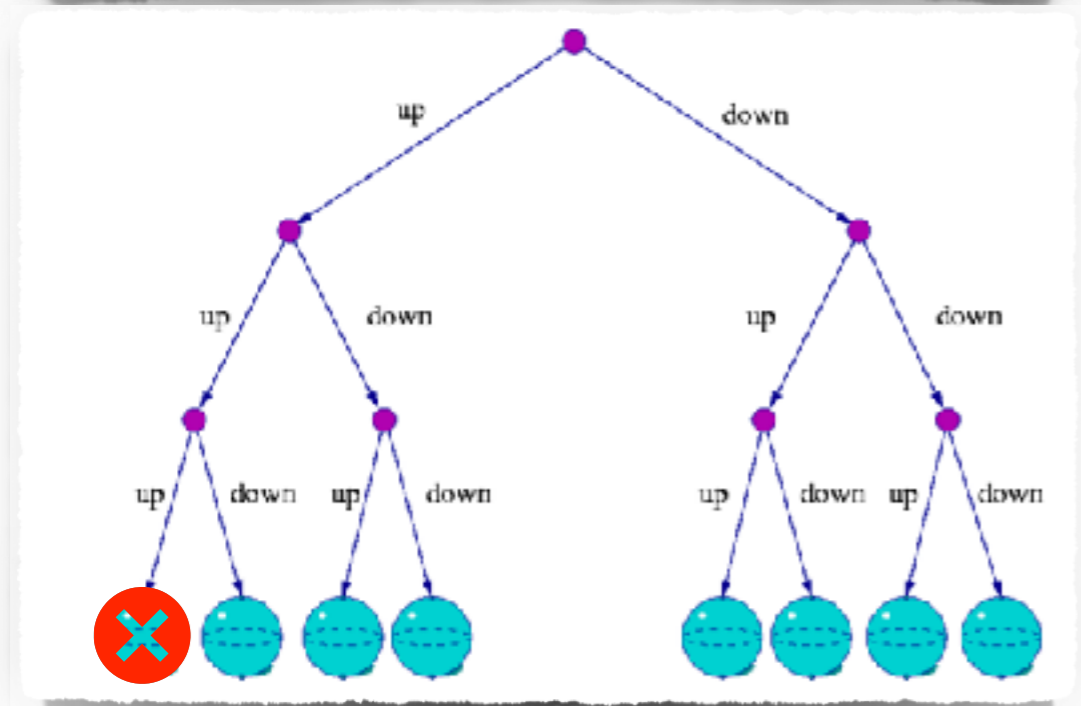


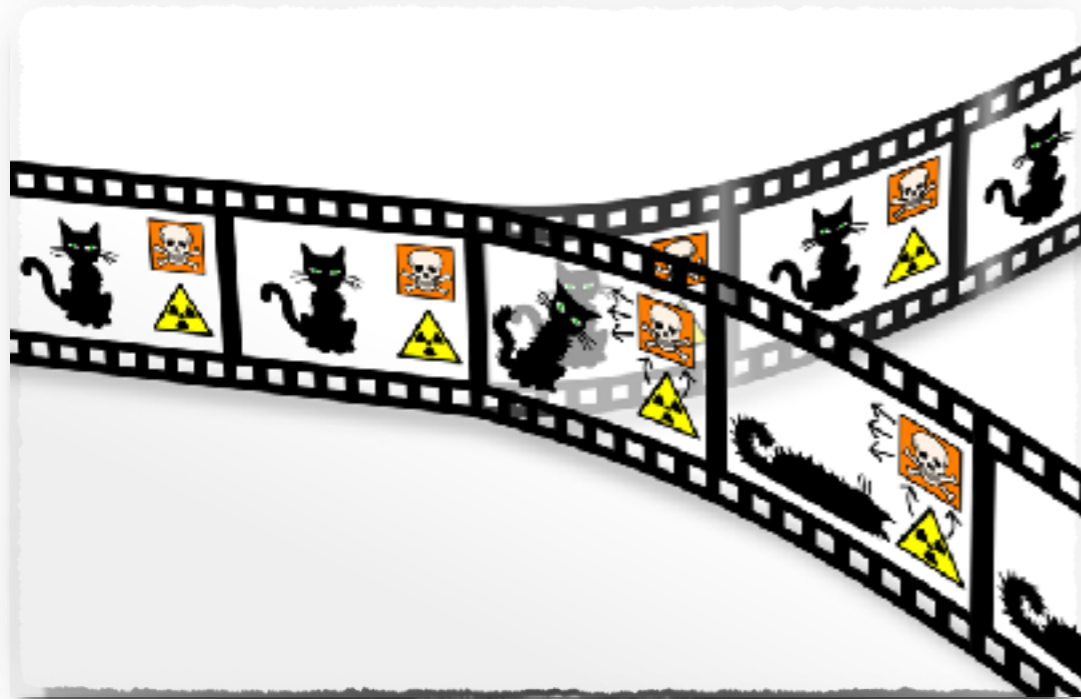


ALGORITHMUS 5.30 (Quantum Bogosort)

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



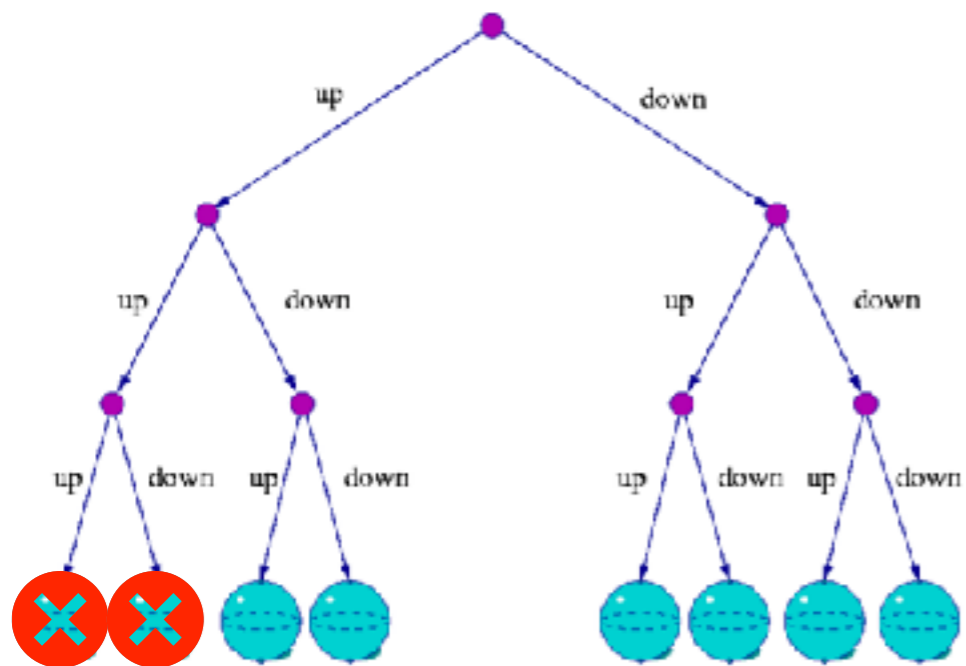


## ALGORITHMUS 5.30 (Quantum Bogosort)

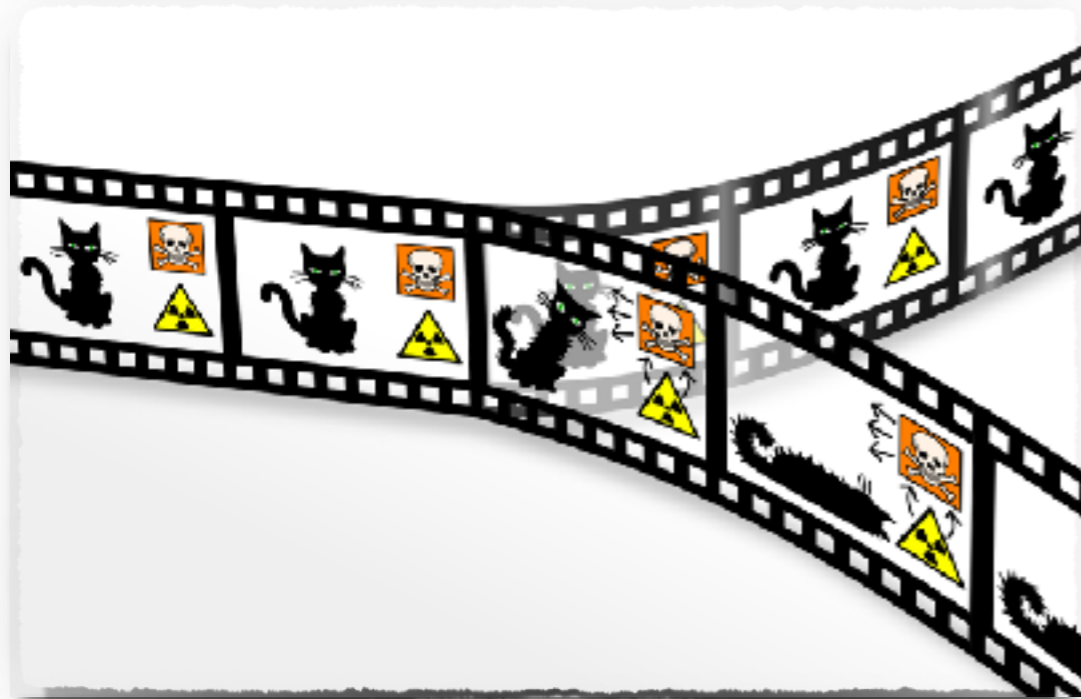
INPUT :  $n$  Zahlen  $A[1], \dots, A[n]$   
SORTIERTES Array in geeignetem Universum

OUTPUT :

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



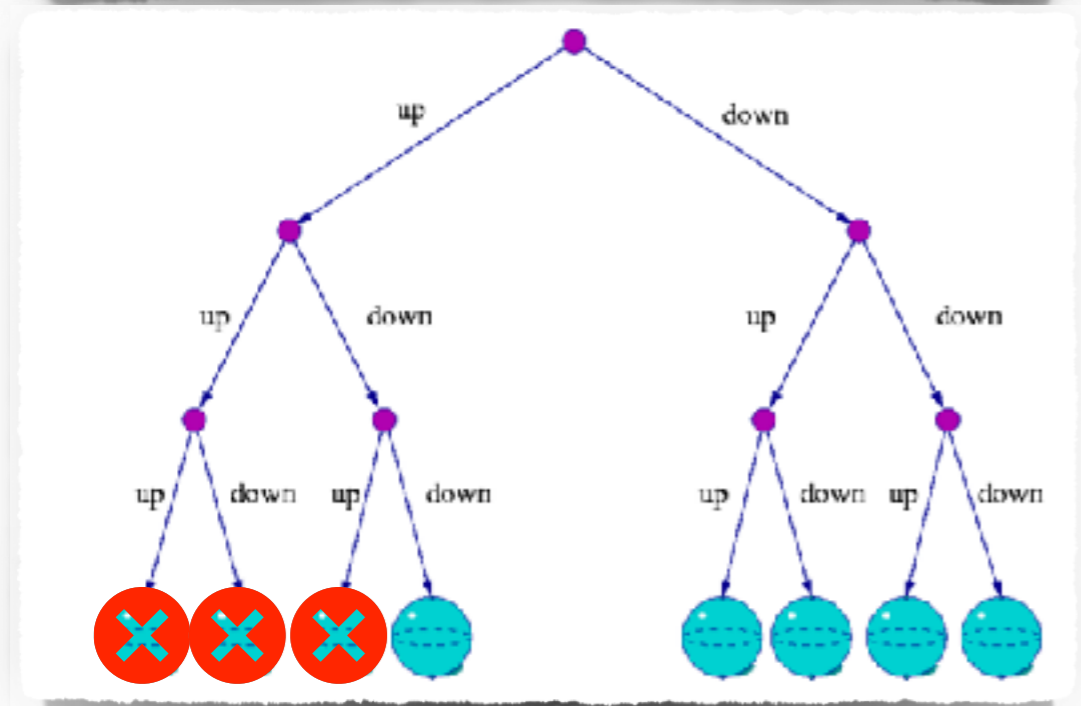


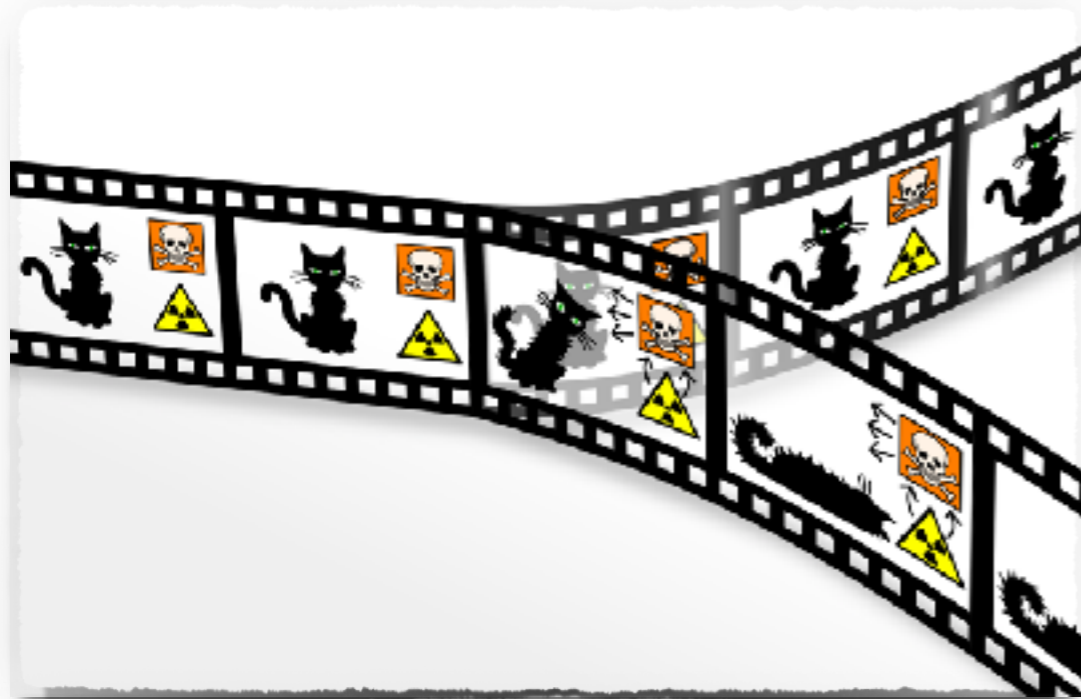


ALGORITHMUS 5.30 (Quantum Bogosort)

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

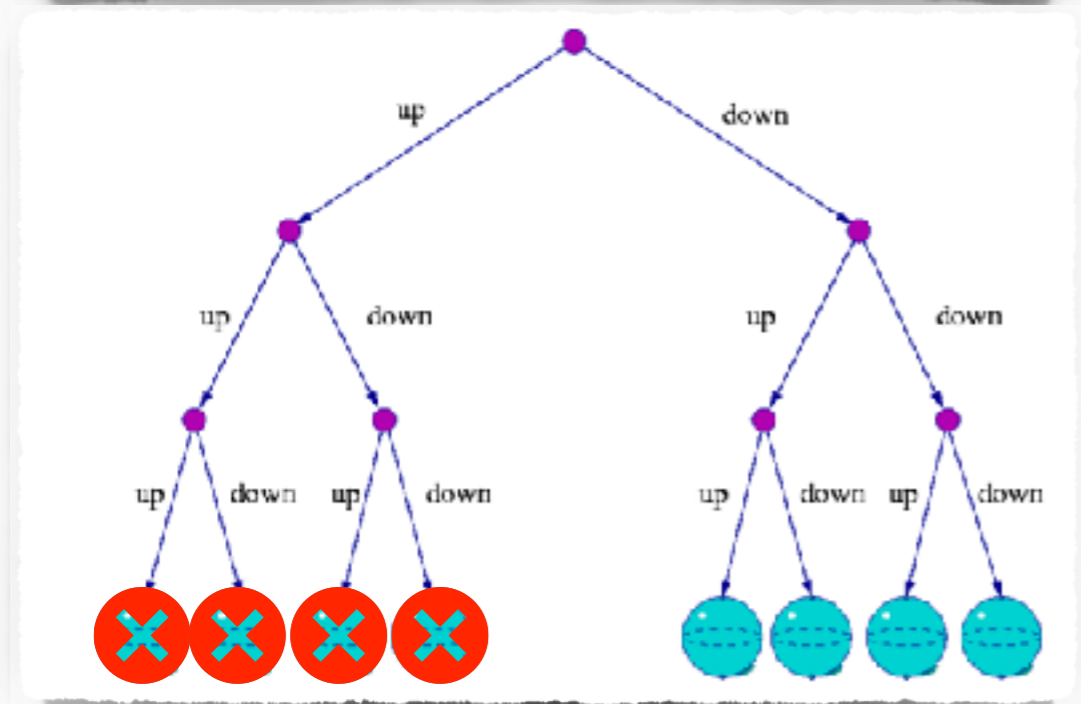


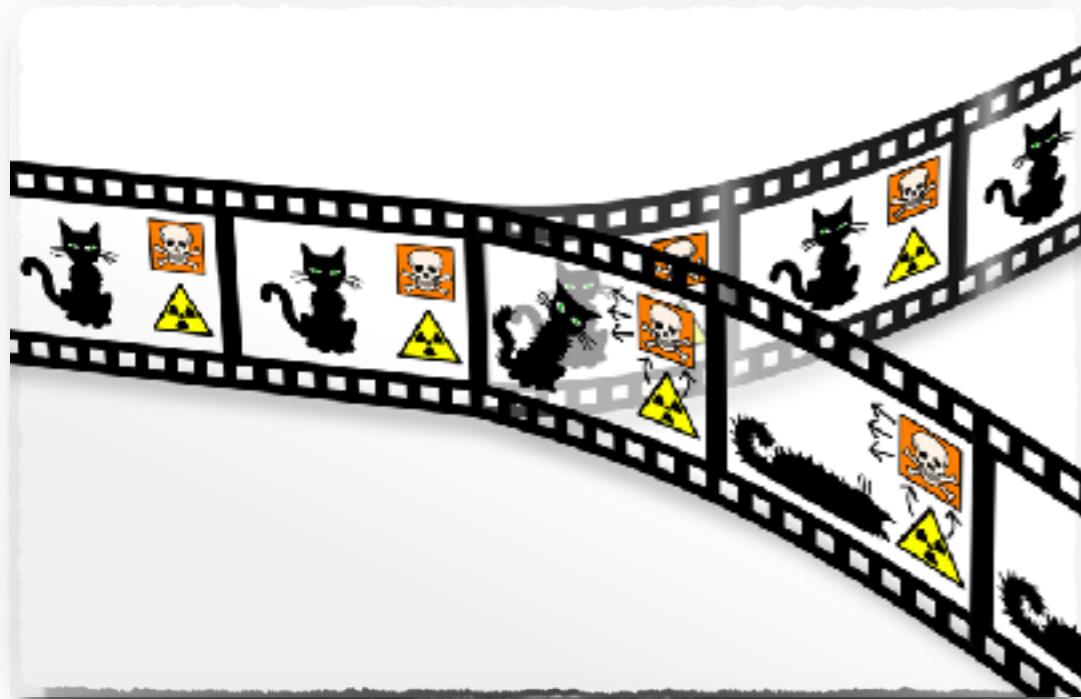


ALGORITHMUS 5.30 (Quantum Bogosort)

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

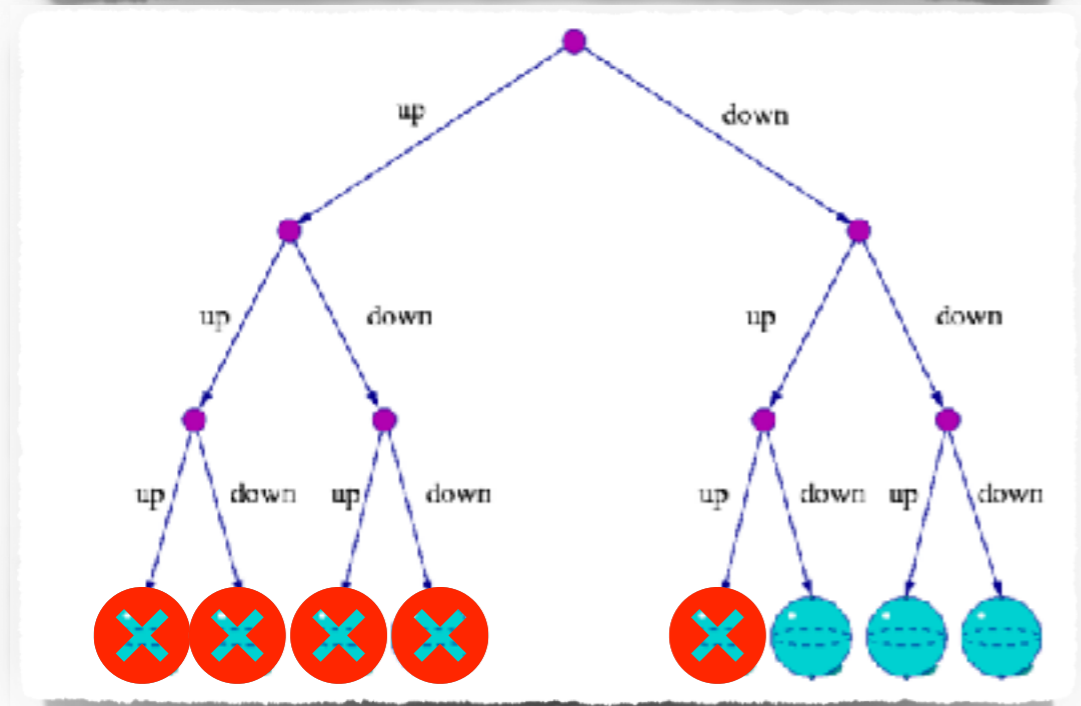


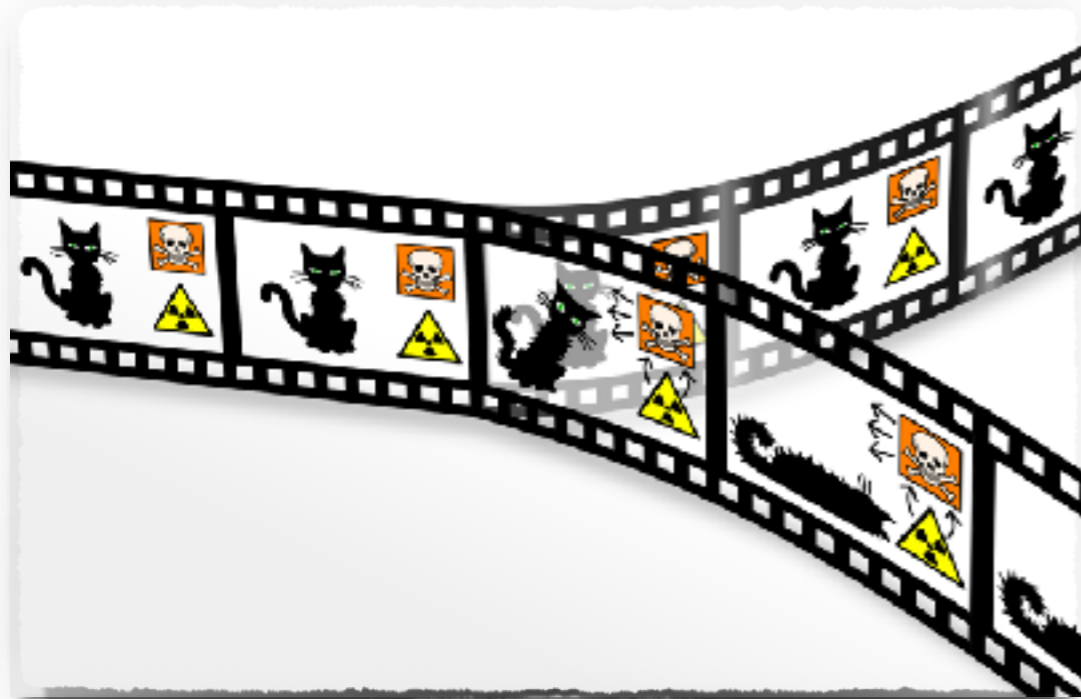


ALGORITHMUS 5.30 (Quantum Bogosort)

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

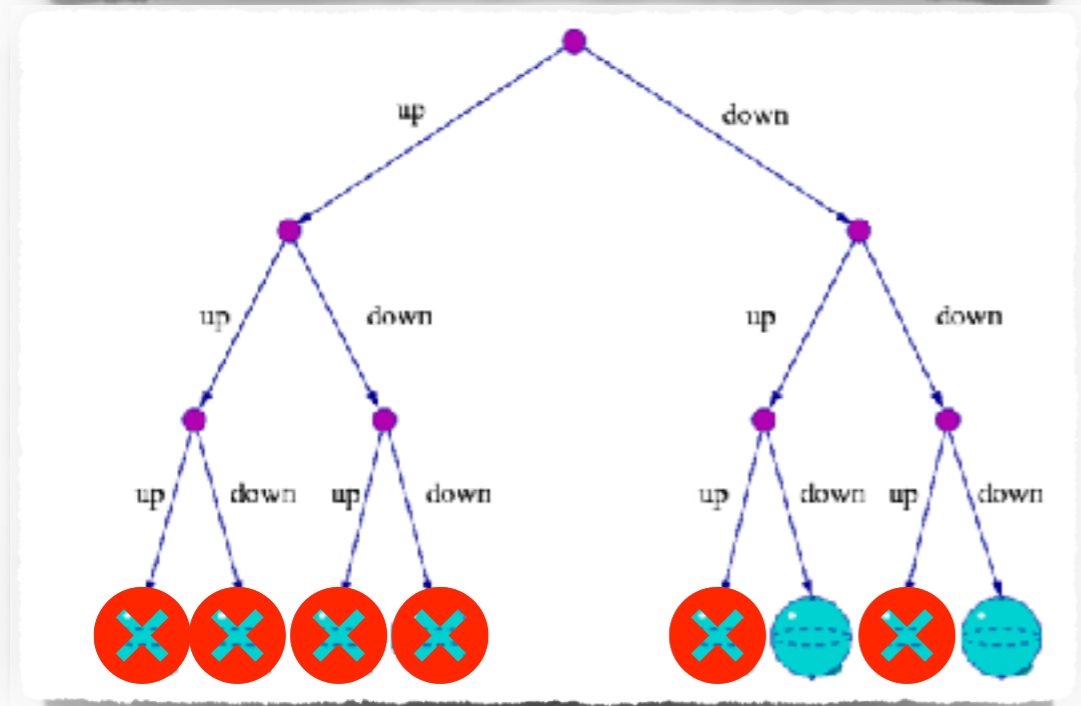


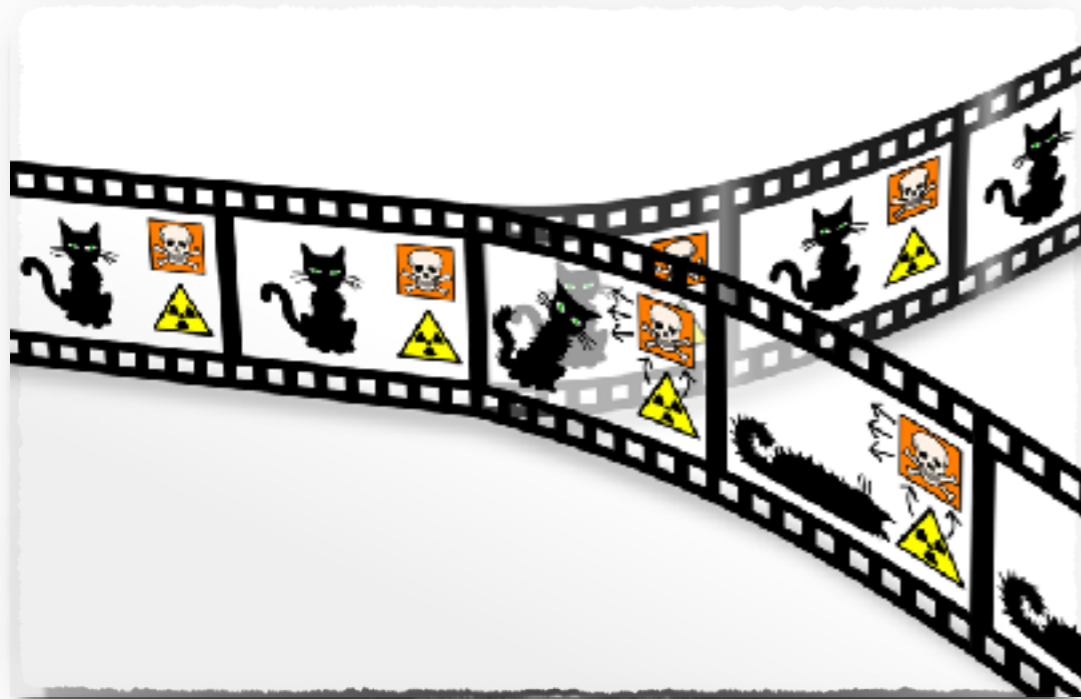


ALGORITHMUS 5.30 (Quantum Bogosort)

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

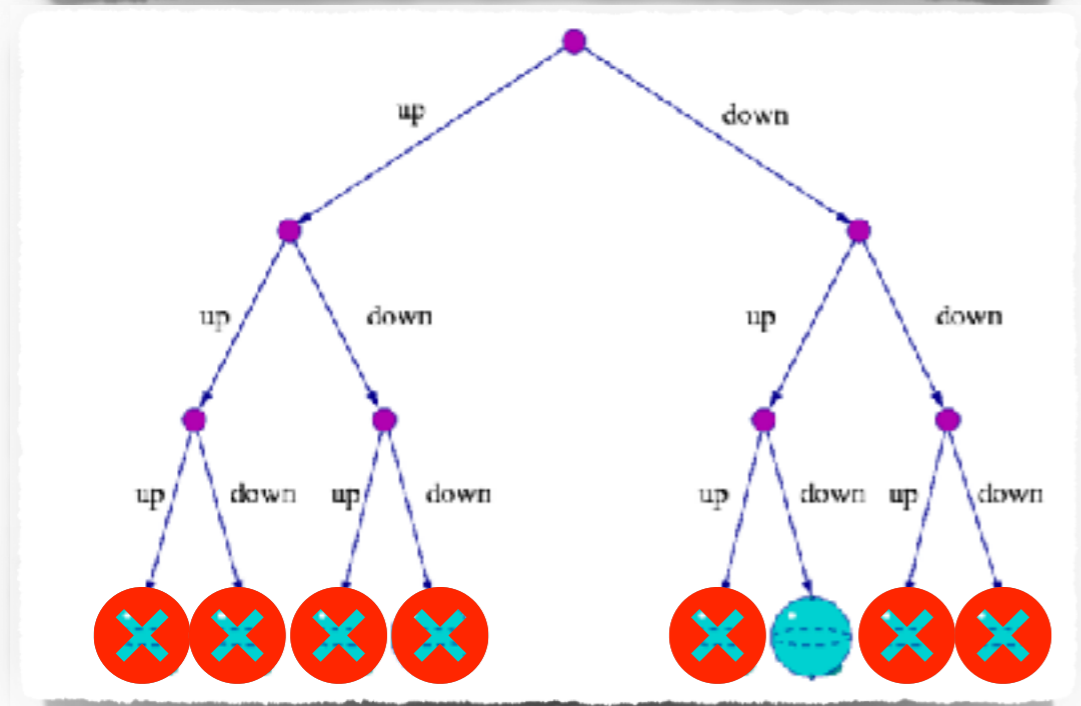


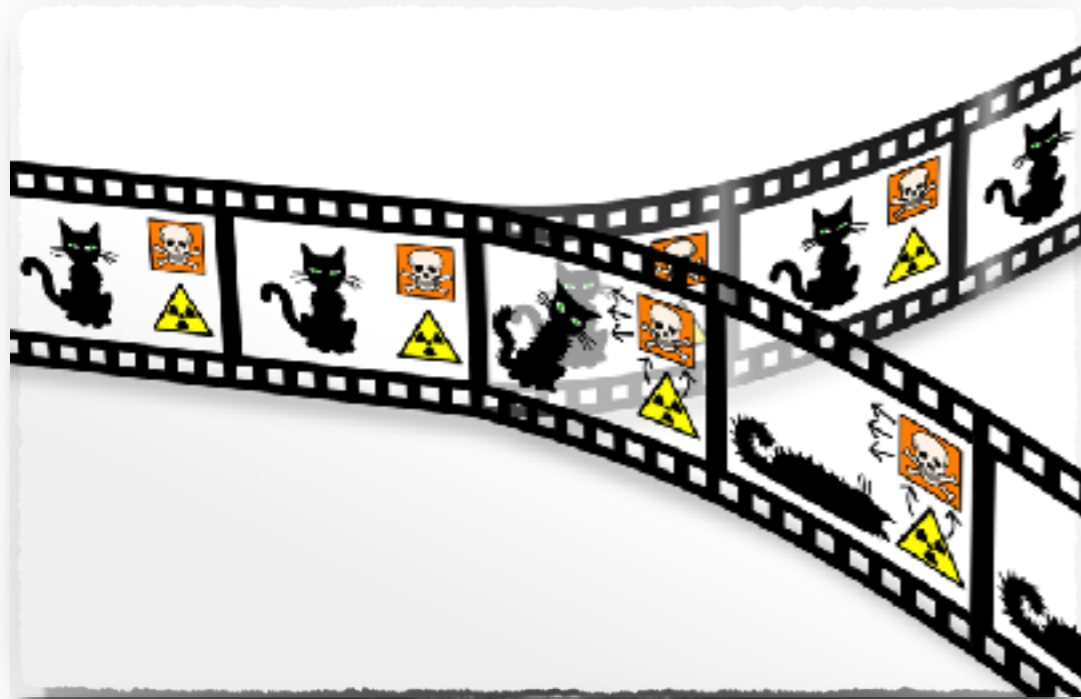


ALGORITHMUS 5.30 (Quantum Bogosort)

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

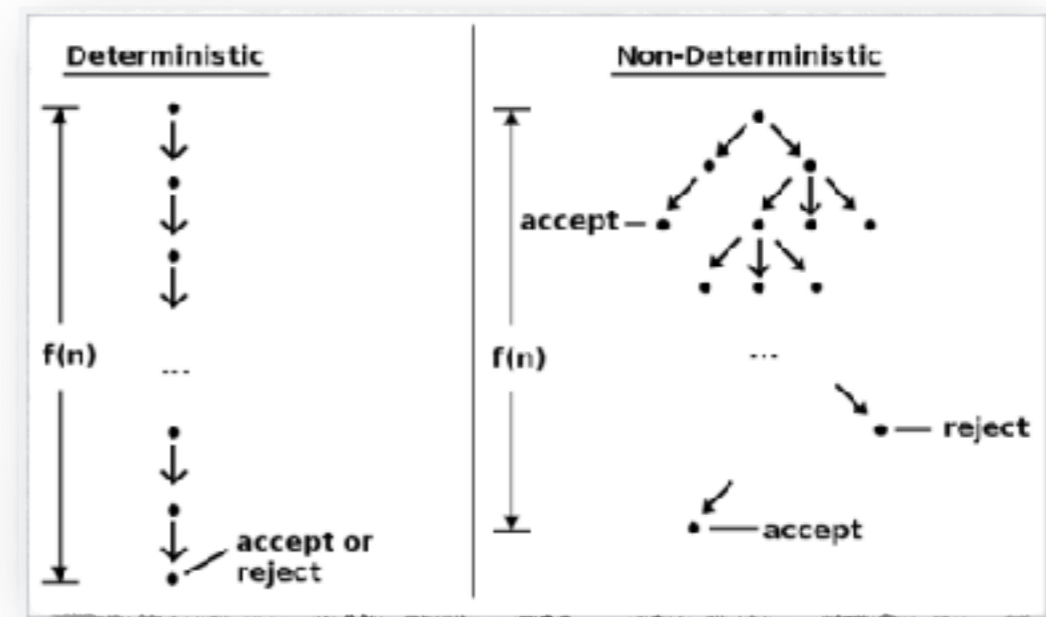
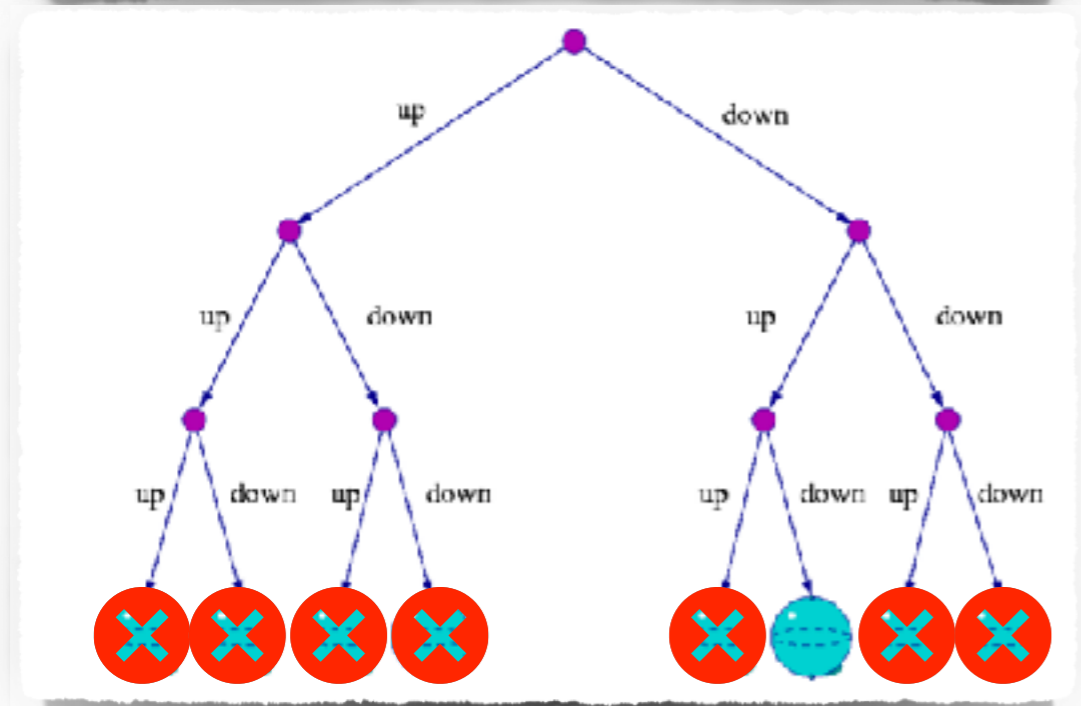


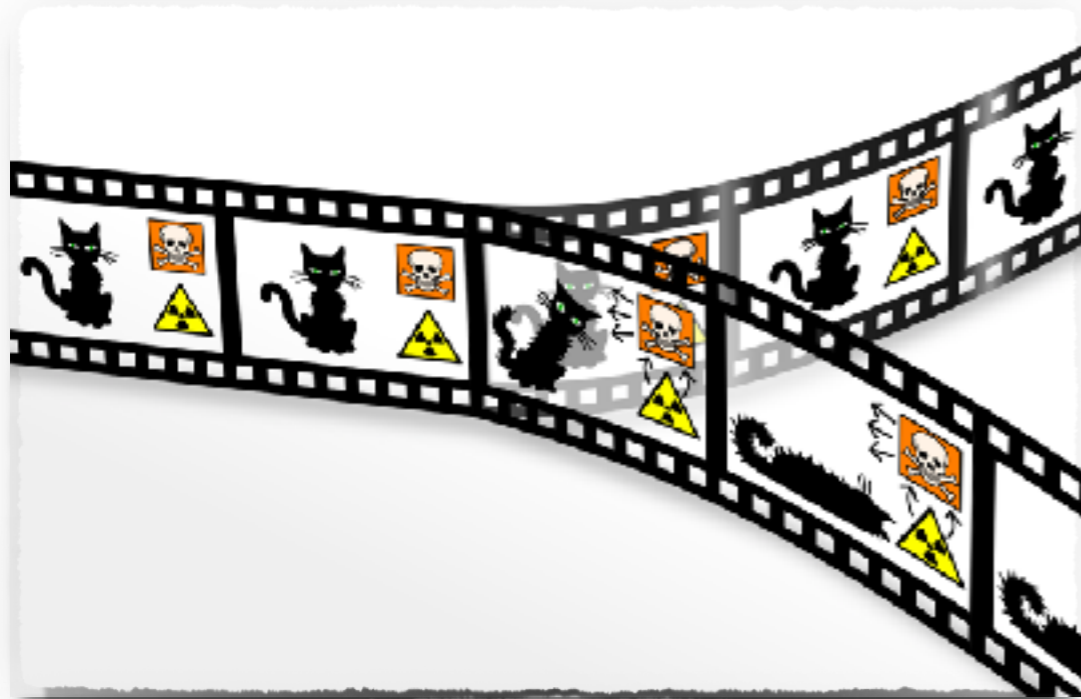


ALGORITHMUS 5.30 (Quantum Bogosort)

INPUT :  $n$  Zahlen  $A[1], \dots, A[n]$   
 SORTIERTER ARRAY in geeignetem Universum

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

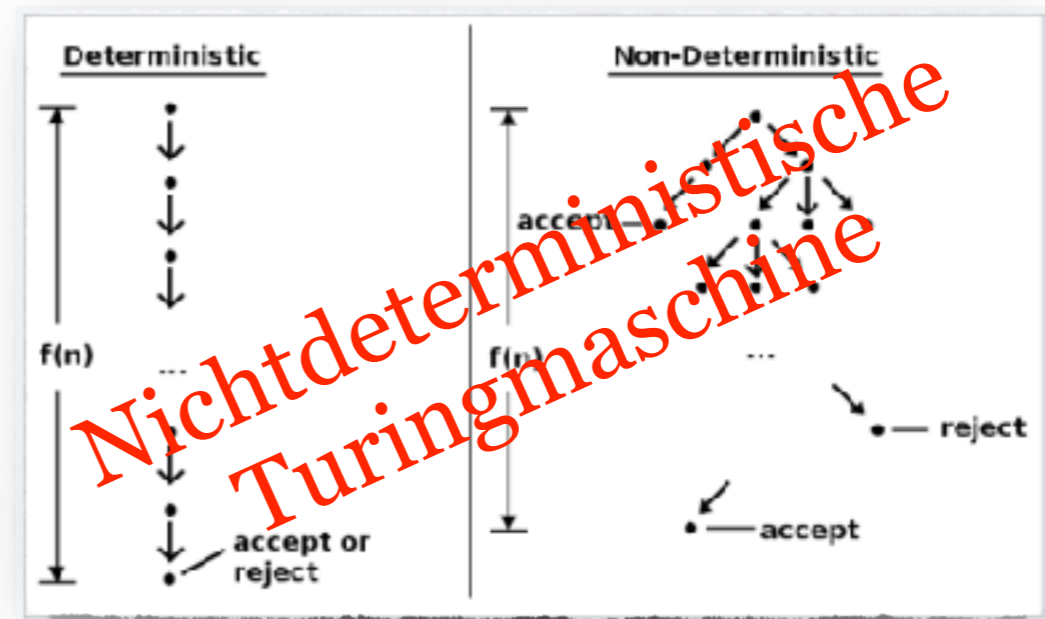
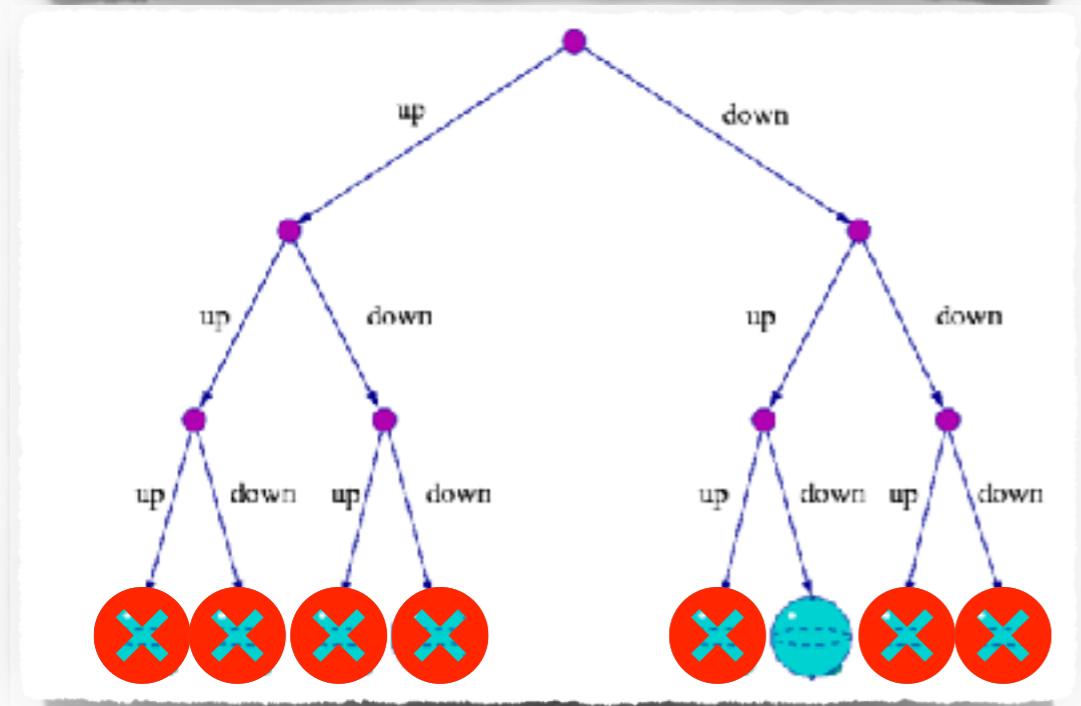




ALGORITHMUS 5.30 (Quantum Bogosort)

INPUT :  $n$  Zahlen  $A[1], \dots, A[n]$   
 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					1	2	3
4	5	7	...					1	2	3
$i$				$R$						

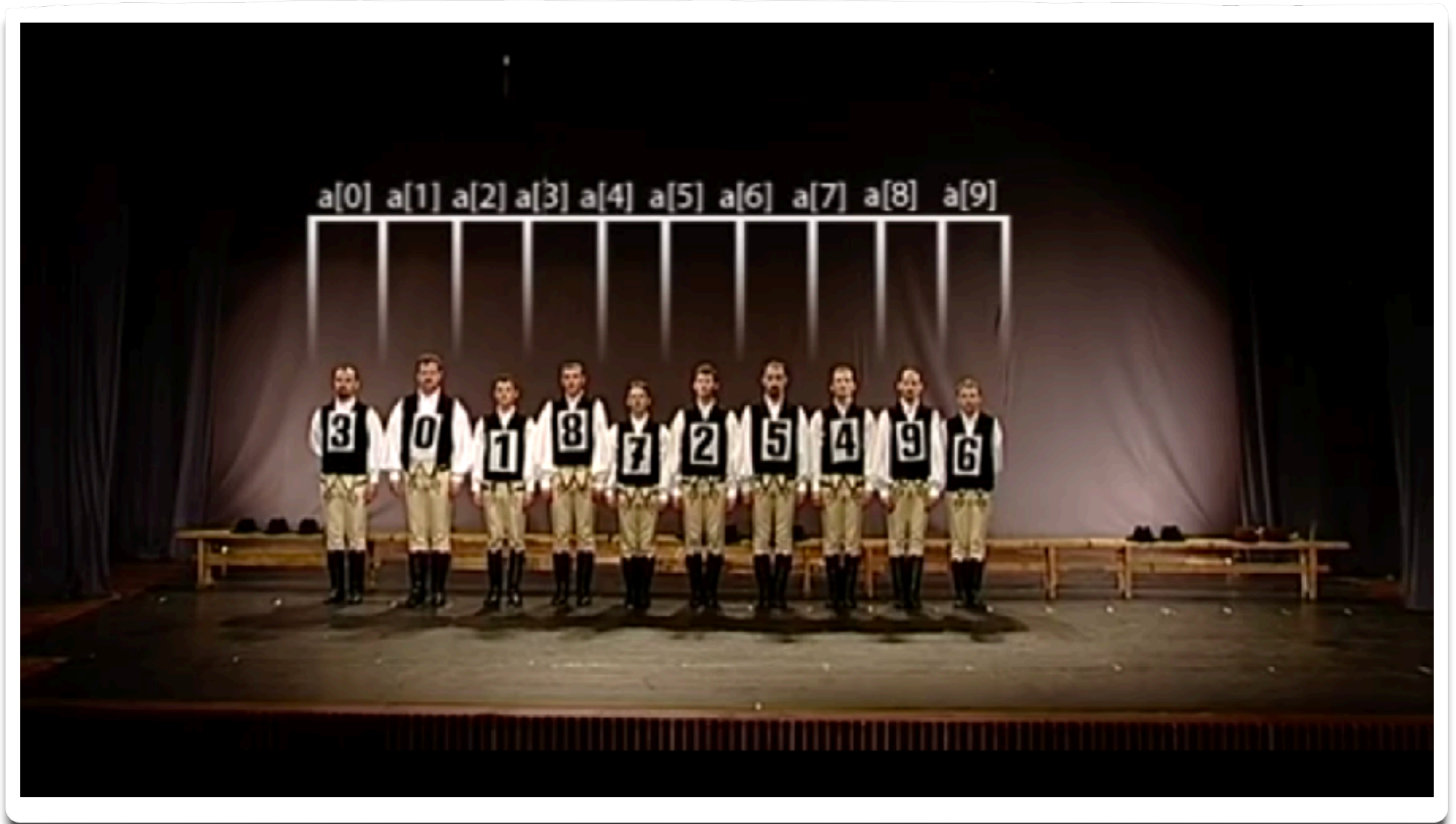


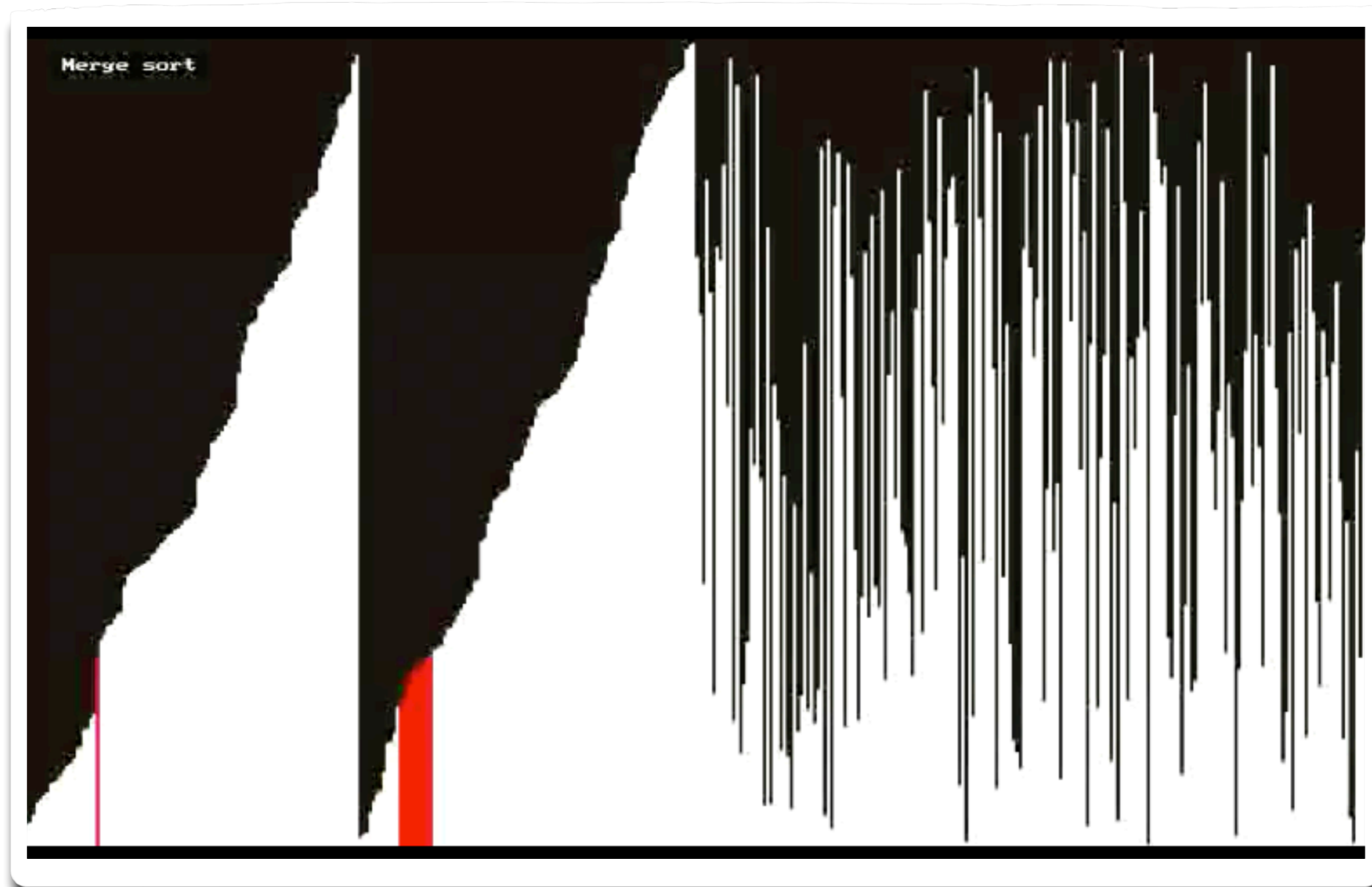
# *Kapitel 5.12: Tanz und Musik!*

*Algorithmen und Datenstrukturen  
WS 2020/21*

**Prof. Dr. Sándor Fekete**











Liebe/r Student/in,

vielen Dank, dass Sie an einer Teilnahme interessiert sind!

Der Arbeitsbereich Empirische Bildungsforschung (Prof. Dr. Lysann Zander, Leibniz Universität Hannover) möchte in Ihrem Studiengang eine **anonyme** Befragung durchführen, bei der es um Ihre Eindrücke im Studiengang sowie Ihre Einstellung zu verschiedenen Themen Ihr Studium betreffend geht. Einige Studierende haben bereits zu Beginn des Semesters an dieser Befragung teilgenommen. Zum Ende des Semesters möchten wir Sie nun erneut befragen.

Selbstverständlich können Sie aber auch an der Befragung teilnehmen, wenn Sie zuvor noch nicht teilgenommen haben.

**Wichtig: Wenn Sie vor Ende der Befragung abbrechen, werden Ihre Daten nicht gespeichert und können nicht ausgewertet werden.** In bisherigen Befragungen hat das Ausfüllen des Fragebogens etwa 15-20 Minuten in Anspruch genommen.

Nach Abschluss der Befragung und der Ergebnisauswertung erhalten Sie - wenn Sie das wünschen - eine Rückmeldung über die Gesamtergebnisse. Dazu können Sie Ihre Anfrage an [emp.bildungsforschung@iew.uni-hannover.de](mailto:emp.bildungsforschung@iew.uni-hannover.de) mit dem Betreff "Ergebnisse Befragung Informatik" senden.

Das Ausfüllen des Fragebogens wird etwa 15-20 Minuten in Anspruch nehmen.

Liebe/r Student/in,

vielen Dank, dass Sie an einer Teilnahme interessiert sind!

Der Arbeitsbereich Empirische Bildungsforschung (Prof. Dr. Lysann Zander, Leibniz Universität Hannover) möchte in Ihrem Studiengang eine **anonyme** Befragung durchführen, bei der es um Ihre Eindrücke im Studiengang sowie Ihre Einstellung zu verschiedenen Themen Ihr Studium betreffend geht. Einige Studierende haben bereits zu Beginn des Semesters an dieser Befragung teilgenommen. Zum Ende des Semesters möchten wir Sie nun erneut befragen. Selbstverständlich können Sie aber auch an der Befragung teilnehmen, wenn Sie zuvor noch nicht teilgenommen haben.

**Wichtig:** Wenn Sie vor Ende der Befragung abbrechen, werden Ihre Daten nicht gespeichert und können nicht ausgewertet werden. In bisherigen Befragungen hat das Ausfüllen des Fragebogens etwa 15-20 Minuten in Anspruch genommen.

Nach Abschluss der Befragung und der Ergebnisauswertung erhalten Sie - wenn Sie das wünschen - eine Rückmeldung über die Gesamtergebnisse. Dazu können Sie Ihre Anfrage an [emp.bildungsforschung@iew.uni-hannover.de](mailto:emp.bildungsforschung@iew.uni-hannover.de) mit dem Betreff "Ergebnisse Befragung Informatik" senden.

Das Ausfüllen des Fragebogens wird etwa 15-20 Minuten in Anspruch nehmen.

Seit 27.01.!

*Herzlichen Dank!*

*s.fekete@tu-bs.de*