



## Paralleles Sortierung

Björn Rathjen   Patrick Winterstein  
Freie Universität Berlin

Proseminar Algorithmen, SS14

## Motivation

## Grundlage des Sortierens

### Komparator

## Sortiernetzwerk

### Aufbau

### Korrektheit

### Odd-Even-Sort

## Laufzeit

### Herleitung

### Vergleich mit Software sortieren

## Zusammenfassung

## Ausblick

### Anhang

# Motivation

Grundlage des Sortierens

Sortiernetzwerk

Laufzeit

Zusammenfassung

Ausblick

Sortieren  
ist Grundlage für :

- ▶ Suche
- ▶ (Sortierung)
  - ▶ Listen
  - ▶ Wörterbücher
  - ▶ ...
- ▶ Ist dies auch effektiver / in Hardware möglich ?

Motivation

## Grundlage des Sortierens Komparator

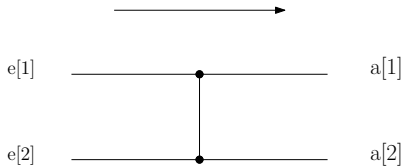
Sortiernetzwerk

Laufzeit

Zusammenfassung

Ausblick

- ▶ 2 Eingänge
- ▶ vergleichender Baustein
- ▶ 2 Ausgänge



```
1  void comp(chan in1, in2, out1, out2 Comparer{}){  
2      a := <- in1;  
3      b := <- in2;  
4  
5      if (a < b){  
6          out1 <- a;  
7          out2 <- b;  
8          return void;  
9      }  
10     out1 <- b;  
11     out2 <- a;  
12     return void;  
13 }
```

Motivation

Grundlage des Sortierens

**Sortiernetzwerk**

Aufbau

Korrektheit

Odd-Even-Sort

Laufzeit

Zusammenfassung

Ausblick



## Aufbau :

- ▶ mehrere Eingabeleitungen (gleiche Anzahl an Ausgabeleitungen)
- ▶ mehrere vergleichende Schritte

## Erweiterung : Aufbau , Aufgabe

### Aufbau :

- ▶ mehrere Eingabeleitungen (gleiche Anzahl an Ausgabeleitungen)
- ▶ mehrere vergleichende Schritte

### Aufgabe :

- ▶ Resultat soll sortierte Ausgabe sein

## Aufbau :

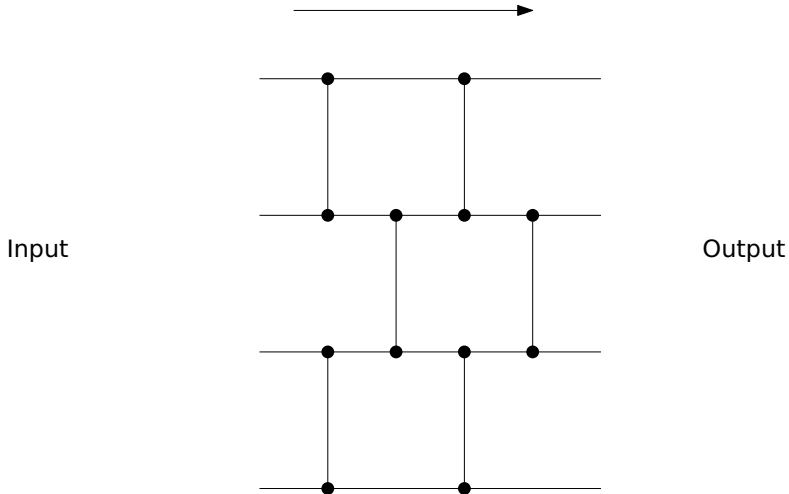
- ▶ mehrere Eingabeleitungen (gleiche Anzahl an Ausgabeleitungen)
- ▶ mehrere vergleichende Schritte

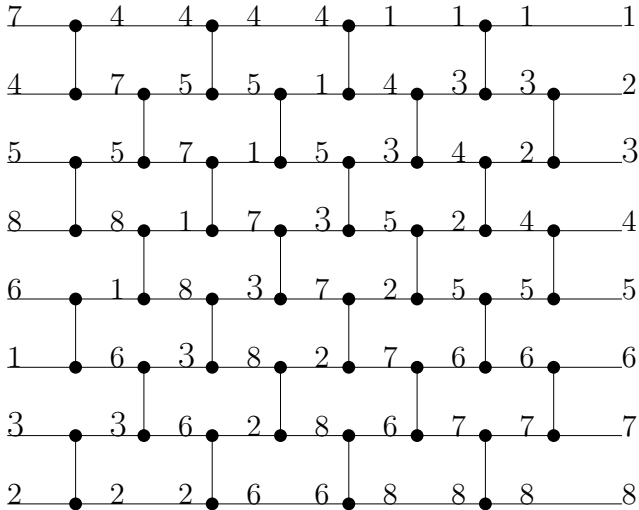
## Aufgabe :

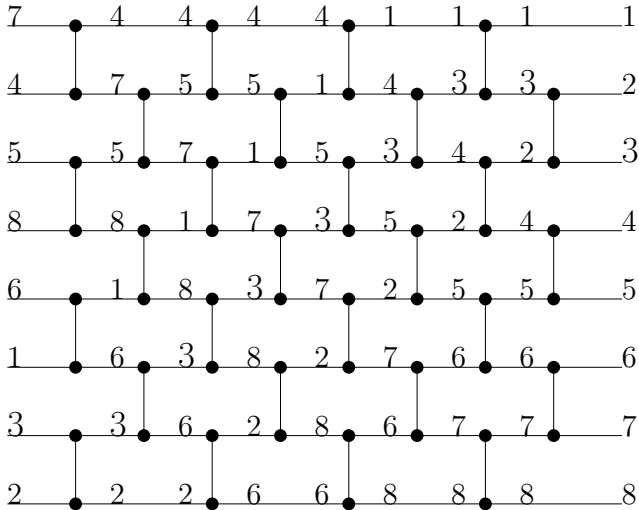
- ▶ Resultat soll sortierte Ausgabe sein

## grundlegendes Prinzip :

- ▶ intuitiver Einsatz von Vergleichen
- ▶ Schrittweises sortieren







Das Zahlenbeispiel funktioniert, aber funktioniert auch jedes andere ?

Lemma:

Wenn es eine Folge  $A$  gibt, die ein Sortiernetzwerk nicht sortiert, so existiert auch eine 0,1-Folge, die von diesem Netzwerk nicht sortiert wird.

Lemma:

Wenn es eine Folge  $A$  gibt, die ein Sortiernetzwerk nicht sortiert, so existiert auch eine 0,1-Folge, die von diesem Netzwerk nicht sortiert wird.

- Führen einen Beweis durch Widerspruch



Lemma:

Wenn es eine Folge  $A$  gibt, die ein Sortiernetzwerk nicht sortiert, so existiert auch eine 0,1-Folge, die von diesem Netzwerk nicht sortiert wird.

- ▶ Führen einen Beweis durch Widerspruch
- ▶ d.h : Wenn ein Algorithmus eine Eingabefolge nicht sortiert, so sortiert er trotzdem alle 0,1-Folgen.

Lemma:

Wenn es eine Folge  $A$  gibt, die ein Sortiernetzwerk nicht sortiert, so existiert auch eine 0,1-Folge, die von diesem Netzwerk nicht sortiert wird.

- ▶ Führen einen Beweis durch Widerspruch
- ▶ d.h : Wenn ein Algorithmus eine Eingabefolge nicht sortiert, so sortiert er trotzdem alle 0,1-Folgen.
- ▶ somit ist das Ziel eine 0,1-Folge zu finden, die nicht sortiert wird

Proof.

- ▶ benötigen folgendes



## Proof.

- ▶ benötigen folgendes
  - ▶ Eingabefolge  $E = e_0 \dots e_N$



## Proof.

- ▶ benötigen folgendes
  - ▶ Eingabefolge  $E = e_0 \dots e_N$
  - ▶ sortierte Folge  $S = s_0 \dots s_N$



## Proof.

- ▶ benötigen folgendes
  - ▶ Eingabefolge  $E = e_0 \dots e_N$
  - ▶ sortierte Folge  $S = s_0 \dots s_N$
  - ▶ unsortierte Ausgabefolge  $U = u_0 \dots u_N$



## Proof.

- ▶ benötigen folgendes
  - ▶ Eingabefolge  $E = e_0 \dots e_N$
  - ▶ sortierte Folge  $S = s_0 \dots s_N$
  - ▶ unsortierte Ausgabefolge  $U = u_0 \dots u_N$
  - ▶  $k$  : (kleinster) Index an dem  $u_k \neq s_k$



## Proof.

- ▶ benötigen folgendes
  - ▶ Eingabefolge  $E = e_0 \dots e_N$
  - ▶ sortierte Folge  $S = s_0 \dots s_N$
  - ▶ unsortierte Ausgabefolge  $U = u_0 \dots u_N$
  - ▶  $k$  : (kleinster) Index an dem  $u_k \neq s_k$
- ▶ dann gilt :
  - (1)  $u_i = s_i \quad \forall 0 \leq i < k$





## Proof.

- ▶ benötigen folgendes
  - ▶ Eingabefolge  $E = e_0 \dots e_N$
  - ▶ sortierte Folge  $S = s_0 \dots s_N$
  - ▶ unsortierte Ausgabefolge  $U = u_0 \dots u_N$
  - ▶  $k$  : (kleinster) Index an dem  $u_k \neq s_k$
- ▶ dann gilt :
  - (1)  $u_i = s_i \quad \forall 0 \leq i < k$
  - (2)  $u_r = s_k$  mit :  $r > k$



## Proof.

- ▶ man kann jede Zahlenfolge durch eine 0,1 Folge repräsentieren  
Konstante  $c$  (hier :  $c = s_k$ ) und Zahlenfolge  $E$  mit den Elementen  $e_i$

$$f(e) = \begin{cases} 0, & \text{if } e_i \leq s_k \\ 1, & \text{if } e_i \geq s_k \end{cases}$$



## Proof.

- man kann jede Zahlenfolge durch eine 0,1 Folge repräsentieren  
Konstante  $c$  (hier :  $c = s_k$ ) und Zahlenfolge  $E$  mit den Elementen  $e_i$

$$f(e) = \begin{cases} 0, & \text{if } e_i \leq s_k \\ 1, & \text{if } e_i \geq s_k \end{cases}$$

- Ausgabe muss also wie folgt aussehen

00.....01 ...0 ...



## Proof.

- man kann jede Zahlenfolge durch eine 0,1 Folge repräsentieren  
Konstante  $c$  (hier :  $c = s_k$ ) und Zahlenfolge  $E$  mit den Elementen  $e_i$

$$f(e) = \begin{cases} 0, & \text{if } e_i \leq s_k \\ 1, & \text{if } e_i \geq s_k \end{cases}$$

- Ausgabe muss also wie folgt aussehen

00.....01<sub>k</sub>...0<sub>r</sub>...



## Proof.

- man kann jede Zahlenfolge durch eine 0,1 Folge repräsentieren  
Konstante  $c$  (hier :  $c = s_k$ ) und Zahlenfolge  $E$  mit den Elementen  $e_i$

$$f(e) = \begin{cases} 0, & \text{if } e_i \leq s_k \\ 1, & \text{if } e_i \geq s_k \end{cases}$$

- Ausgabe muss also wie folgt aussehen

00.....01<sub>k</sub>...0<sub>r</sub>...

⇒ 0,1-Folge ist nicht sortiert



## Proof.

- man kann jede Zahlenfolge durch eine 0,1 Folge repräsentieren  
Konstante  $c$  (hier :  $c = s_k$ ) und Zahlenfolge  $E$  mit den Elementen  $e_i$

$$f(e) = \begin{cases} 0, & \text{if } e_i \leq s_k \\ 1, & \text{if } e_i \geq s_k \end{cases}$$

- Ausgabe muss also wie folgt aussehen

00.....01<sub>k</sub>...0<sub>r</sub>...

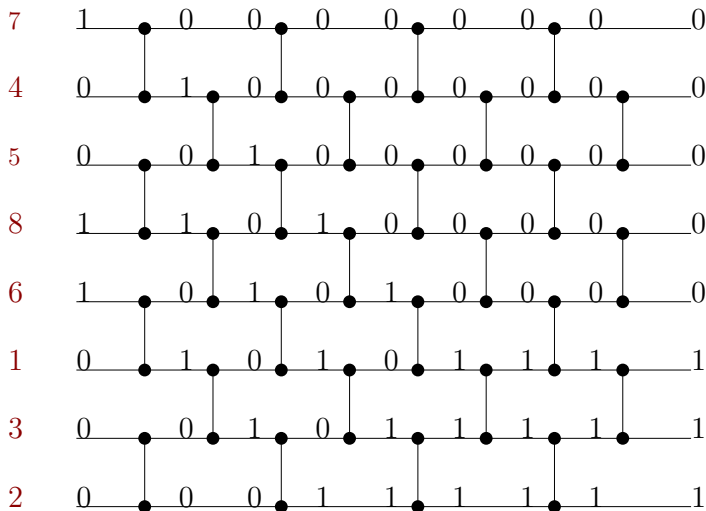
- ⇒ 0,1-Folge ist nicht sortiert
- ⇒ Widerspruch zur Annahme



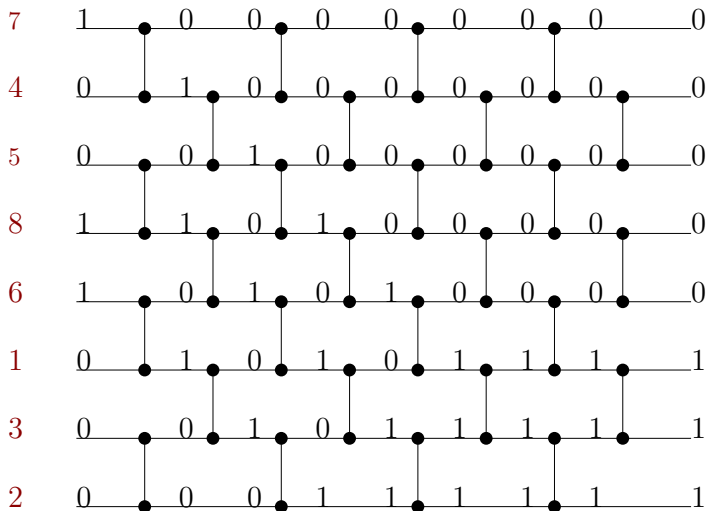
Vorteile des 0,1-Prinzips :

- ▶ einfach
  - ▶ sauberer Testcode
  - ▶ Sicherheit
- ▶ Anzahl der Testfälle sinkt

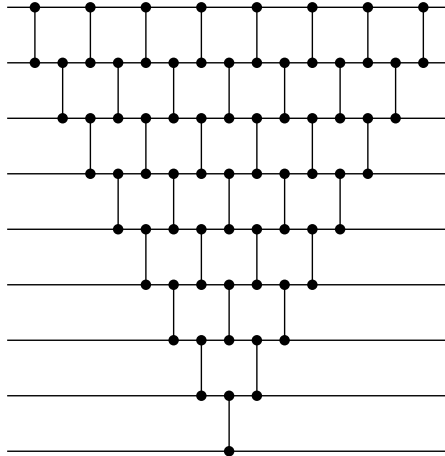
$$n^n \rightarrow 2^n$$



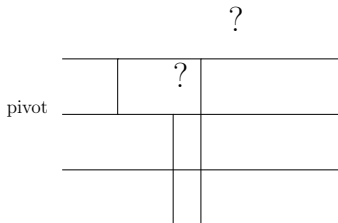




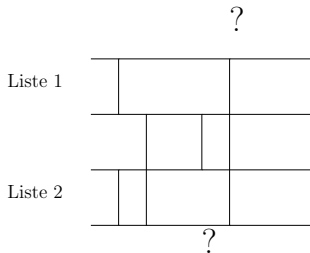
Beispiel an der Tafel ?



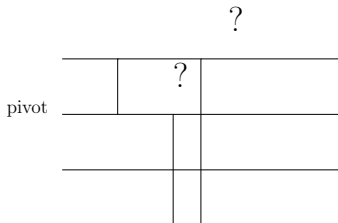
## Quicksort



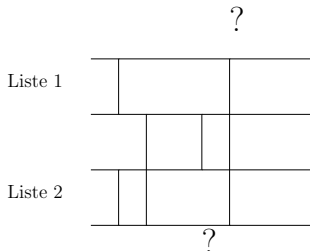
## Mergesort



## Quicksort



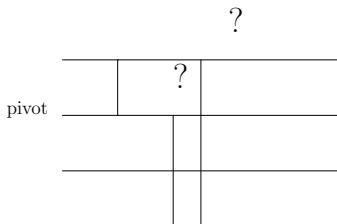
## Mergesort



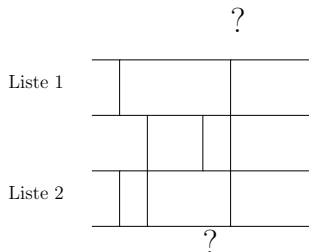
Quicksort : wo ist das Pivot Element ?

Mit welchem Element müssen wir nun vergleichen?

## Quicksort



## Mergesort



Quicksort : wo ist das Pivot Element ?

Mit welchem Element müssen wir nun vergleichen?

Mergesort : Wo ist nun das größte Element ?

welcher Vergleich kommt nun?)



## Aufgabe :

- ▶ Resultat soll sortierte Ausgabe sein
- ▶ **soll effizient sein**

Aufgabe :

- ▶ Resultat soll sortierte Ausgabe sein
- ▶ **soll effizient sein**

grundlegendes Prinzip :

- ▶ intuitiver Einsatz von Vergleichen  
+ **Einbezug von Teile und Herrsche**



Zur Entwicklung verwenden wir Odd-Even-Sort

Zur Entwicklung verwenden wir Odd-Even-Sort

Ablauf :

- ▶ Voraussetzung : bekommen zwei sortierte Listen

Zur Entwicklung verwenden wir Odd-Even-Sort

Ablauf :

- ▶ Voraussetzung : bekommen zwei sortierte Listen
- ▶ trennen in geraden und ungeraden Index

Zur Entwicklung verwenden wir Odd-Even-Sort

Ablauf :

- ▶ Voraussetzung : bekommen zwei sortierte Listen
- ▶ trennen in geraden und ungeraden Index
- ▶ fassen  $a(\text{even})$   $b(\text{odd}) = c$  und  $a(\text{odd})$   $b(\text{even}) = d$  zusammen  
(Resultat wird rekursiv sortiert)

Zur Entwicklung verwenden wir Odd-Even-Sort

Ablauf :

- ▶ Voraussetzung : bekommen zwei sortierte Listen
- ▶ trennen in geraden und ungeraden Index
- ▶ fassen  $a(\text{even})$   $b(\text{odd}) = c$  und  $a(\text{odd})$   $b(\text{even}) = d$  zusammen (Resultat wird rekursiv sortiert)
- ▶ sortierte  $c$  und  $d$  werden indexweise verschachtelt

Zur Entwicklung verwenden wir Odd-Even-Sort

Ablauf :

- ▶ Voraussetzung : bekommen zwei sortierte Listen
- ▶ trennen in geraden und ungeraden Index
- ▶ fassen  $a(\text{even})$   $b(\text{odd}) = c$  und  $a(\text{odd})$   $b(\text{even}) = d$  zusammen (Resultat wird rekursiv sortiert)
- ▶ sortierte  $c$  und  $d$  werden indexweise verschachtelt
- ▶ aufeinander folgende Paare werden verglichen und in richtige Reihenfolge gebracht

## Beispiel : Odd-Even-Mergesort

Beispiel : an der Tafel

$$A = 2, 3, 4, 8$$

$$B = 1, 5, 6, 7$$

## Beispiel : Odd-Even-Mergesort

Beispiel : an der Tafel

$$A = 2, 3, 4, 8$$

$$B = 1, 5, 6, 7$$

$$A_e = 2, 4 \quad B_o = 5, 7 \Rightarrow C = 2, 4, 5, 7$$



## Beispiel : Odd-Even-Mergesort

Beispiel : an der Tafel

$$A = 2, 3, 4, 8$$

$$B = 1, 5, 6, 7$$

$$A_e = 2, 4 \quad B_o = 5, 7 \Rightarrow C = 2, 4, 5, 7$$

$$A_o = 3, 8 \quad B_e = 1, 6 \Rightarrow D = 1, 3, 6, 8$$

## Beispiel : Odd-Even-Mergesort

Beispiel : an der Tafel

$$A = 2, 3, 4, 8$$

$$B = 1, 5, 6, 7$$

$$A_e = 2, 4 \quad B_o = 5, 7 \Rightarrow C = 2, 4, 5, 7$$

$$A_o = 3, 8 \quad B_e = 1, 6 \Rightarrow D = 1, 3, 6, 8$$

nach dem verschachteln

$$2, 1, 4, 3, 5, 6, 7, 8,$$

## Beispiel : Odd-Even-Mergesort

Beispiel : an der Tafel

$$A = 2, 3, 4, 8$$

$$B = 1, 5, 6, 7$$

$$A_e = 2, 4 \quad B_o = 5, 7 \Rightarrow C = 2, 4, 5, 7$$

$$A_o = 3, 8 \quad B_e = 1, 6 \Rightarrow D = 1, 3, 6, 8$$

nach dem verschachteln

$$2, 1, 4, 3, 5, 6, 7, 8,$$

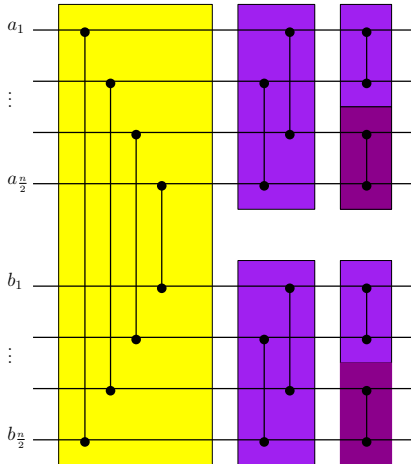
Resultat :

$$1, 2, 3, 4, 5, 6, 7, 8$$

- ▶ Veranschaulichung mit dem 0,1-Pinzip
  - ▶ gleichmäßiges Aufteilen von Nullen und Einsen
  - ▶ somit alle nah beieinander
- ▶ mit Bubblesort verwandt : Laufzeit

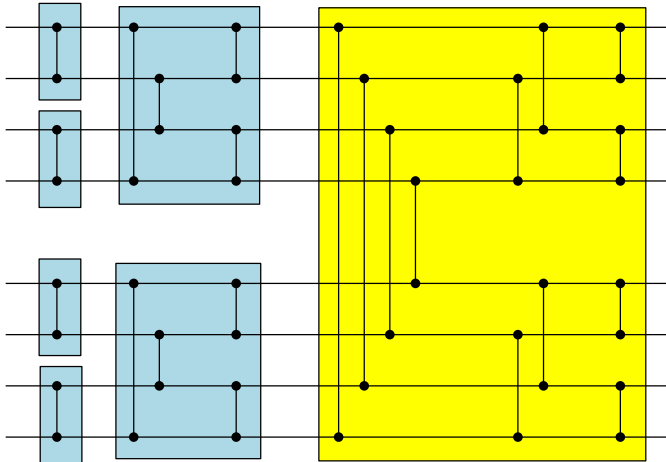
worst :  $O(n^2)$

best :  $O(n)$



Ablauf :

- ▶ sortiert / mischt Eingabelisten
  - ▶ untere Hälfte alle größer als in oberer
- ▶ rekursiv die kleineren Listen
- ▶ Resultat eine Sortierte Liste



Beispiel an der Tafel



Motivation

Grundlage des Sortierens

Sortiernetzwerk

**Laufzeit**

Herleitung

Vergleich mit Software sortieren

Zusammenfassung

Ausblick

N	Anzahl der Schritte
$2^1$	

N	Anzahl der Schritte
$2^1$	1

N	Anzahl der Schritte
$2^1$	1
$2^2$	

N	Anzahl der Schritte
$2^1$	1
$2^2$	$1 + 2$

N	Anzahl der Schritte
$2^1$	1
$2^2$	$1 + 2$
$2^k$	

N	Anzahl der Schritte
$2^1$	1
$2^2$	$1 + 2$
$2^k$	$1 + 2 + 3 + \dots + k - 1 + k = \sum_{i=1}^k i$

N	Anzahl der Schritte
$2^1$	1
$2^2$	$1 + 2$
$2^k$ (kleiner Gauss)	$1 + 2 + 3 + \dots + k - 1 + k = \sum_{i=1}^k i$



N	Anzahl der Schritte
$2^1$	1
$2^2$	1 + 2
$2^k$	$1 + 2 + 3 + \dots + k - 1 + k = \sum_{i=1}^k i$
(kleiner Gauss)	$= \frac{k \cdot (k+1)}{2}$

N	Anzahl der Schritte
$2^1$	1
$2^2$	1 + 2
$2^k$	$1 + 2 + 3 + \dots + k - 1 + k = \sum_{i=1}^k i$
(kleiner Gauss)	$= \frac{k \cdot (k+1)}{2}$
$(k = \log_2 n)$	

N	Anzahl der Schritte
$2^1$	1
$2^2$	1 + 2
$2^k$	$1 + 2 + 3 + \dots + k - 1 + k = \sum_{i=1}^k i$
(kleiner Gauss)	$= \frac{k \cdot (k+1)}{2}$
$(k = \log_2 n)$	$\Rightarrow \frac{1}{2} \cdot \log_2 n (\log_2 n + 1)$

## Vergleich zu bekannten Softwareansätzen

- ▶ Schritte gegen Vergleiche
  - ▶ in jedem Schritt  $\frac{n}{2}$  Komparatoren benötigt
  - $\frac{n}{2} \left( \frac{1}{2} \cdot \log_2 n (\log_2 n + 1) \right)$
  - ▶ 1 Vergleiche bei Software benötigt
  - ▶ Laufzeit konstant
- ▶ Abhängigkeit von der Eingabe
- ▶ Bezug zum vorherigen Vergleich
- ▶ Andere Laufzeiten

Algorithmus	Laufzeit		
	best	worst	
Bubblesort	$O(n)$	$O(n^2)$	
Mergosort	$O(n \log n)$	$O(n \log n)$	
Quicksort	$O(n \log n)$	$O(n^2)$	
Netzwerk	$\frac{1}{2} \cdot \log_2 n (\log_2 n + 1)$		

## Vergleich zu bekannten Softwareansätzen

- ▶ Schritte gegen Vergleiche
  - ▶ in jedem Schritt  $\frac{n}{2}$  Komparatoren benötigt
  - $\frac{n}{2} \left( \frac{1}{2} \cdot \log_2 n (\log_2 n + 1) \right)$
  - ▶ 1 Vergleiche bei Software benötigt
  - ▶ Laufzeit konstant
- ▶ Abhängigkeit von der Eingabe
- ▶ Bezug zum vorherigen Vergleich
- ▶ Andere Laufzeiten

Algorithmus	Laufzeit		
	best	worst	average / normiert
Bubblesort	$O(n^2)$	$O(n^2)$	$\frac{1}{2} (n^2 - n \cdot \ln n - (\gamma + \ln(2) - 1) \cdot n) + O(\sqrt{n})$
Mergosort	$O(n \log n)$	$O(n \log n)$	$n \log n$
Quicksort	$O(n \log n)$	$O(n^2)$	$n \log n$
Netzwerk	$\frac{1}{2} \cdot \log_2 n (\log_2 n + 1)$		$\frac{n}{2} \left( \frac{1}{2} \cdot \log_2 n (\log_2 n + 1) \right)$

Motivation

Grundlage des Sortierens

Sortiernetzwerk

Laufzeit

**Zusammenfassung**

Ausblick

- ▶ paralleles sortieren ist schnell und effizient
- ▶ problemabhängige Lösung

Motivation

Grundlage des Sortierens

Sortiernetzwerk

Laufzeit

Zusammenfassung

**Ausblick**

**Anhang**



- ▶ 2D Netzwerke
- ▶ Hypercubes
- ▶ Paketrouting
- ▶ Simulation von Maschinenmodellen
- ▶ ...

## For Further Reading I



*Taschenbuch der Algorithmen.*  
Springer Verlag , 2008.



Tom Leighton.  
*Einführung in Parallele Algorithmen und Architekturen*  
Gitter, Bäume und Hypercubes.  
Thomson Publishing , 1997.  
3-8266-0248-X



Laufzeiten Sortieralgorithmen  
[www.wikipedia.de](http://www.wikipedia.de)



Alternativer Ansatz zum Beweis des 0,1-Prinzip  
<http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/networks/nulleins.htm>

Ende

Fragen, Anregungen?