Paralleles Sortierung

Björn Rathjen   Patrick Winterstein
Freie Universität Berlin

Proseminar Algorithmen, SS14

Inhalt

Motivation

Grundlage des Sortierens

Sortiernetzwerk

Laufzeit

Zusammenfassung

Ausblick

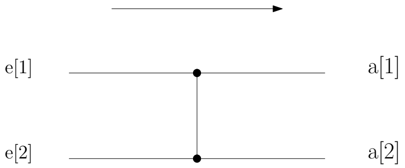
Motivation : Allgemein

Sortieren
ist Grundlage für :

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

Aufbau

- 2 Eingänge
- vergleichender Baustein
- 2 Ausgänge



Vergleichender Baustein (ii)

```
void comp(chan in1, in2, out1, out2 Comparer{}){
    a := <− in1;
    b := <− in2;

    if (a < b){
        out1 <− a;
        out2 <− b;
        return void;
    }
    out1 <− b;
    out2 <− a;
    return void;
}
```

Erweiterung : Aufbau , Aufgabe

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

Erweiterung : Aufbau , Aufgabe

Aufbau :

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

Aufgabe :

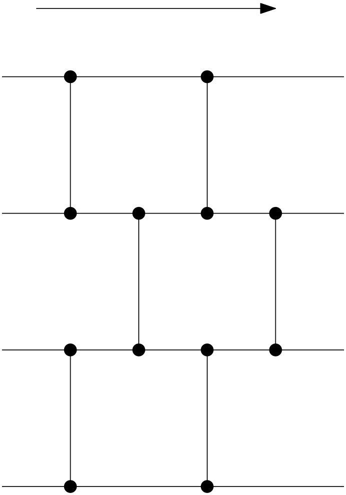
- Resultat soll sortierte Ausgabe sein

grundlegendes Prinzip :

- intuitiver Einsatz von Vergleichen
- Schrittweises sortieren

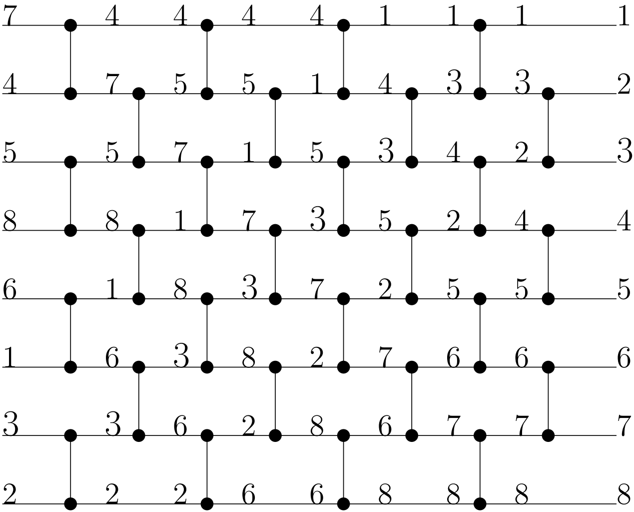
naiv : grundlegendes Prinzip

Input

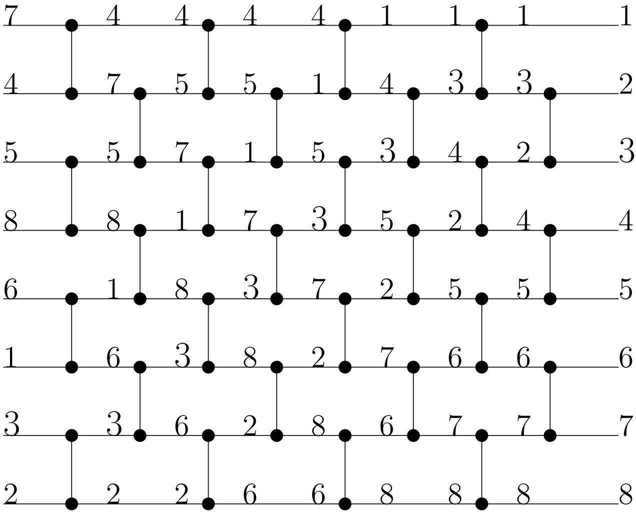


Output

Demonstration



Demonstration



Das Zahlenbeispiel funktioniert, aber funktioniert auch jedes andere ?

0,1-Prinzip

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.

0,1-Prinzip

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

0,1-Prinzip

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.

0,1-Prinzip

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

Beweis : 0,1-Prinzip

Proof.

- benötigen folgendes

□

Beweis : 0,1-Prinzip

Proof.

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

$\square$

Beweis : 0,1-Prinzip

Proof.

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

$\square$

Beweis : 0,1-Prinzip

Proof.

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

$\square$

Beweis : 0,1-Prinzip

Proof.

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

$\square$

Freie Universität Berlin

Proof.

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

(1) $u_i = s_i \; \forall \, 0 \leq i < k$

□

Freie Universität Berlin

Proof.

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

$\square$

Proof.

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

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

□

Proof.

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

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

- Ausgabe muss also wie folgt aussehen

$$00 \ldots\ldots 01 \;\; \ldots 0 \;\; \ldots$$

□

Proof.

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

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

- Ausgabe muss also wie folgt aussehen

$$00\ldots\ldots01_k\ldots0_r\ldots$$

□

Proof.

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

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

- Ausgabe muss also wie folgt aussehen

$$00 \ldots\ldots 01_k \ldots 0_r \ldots$$

$\Rightarrow$ 0,1-Folge ist nicht sortiert

$\square$

Proof.

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

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

- Ausgabe muss also wie folgt aussehen

$$00\ldots\ldots 01_k \ldots 0_r \ldots$$

⇒ 0,1-Folge ist nicht sortiert

⇒ Widerspruch zur Annahme

□

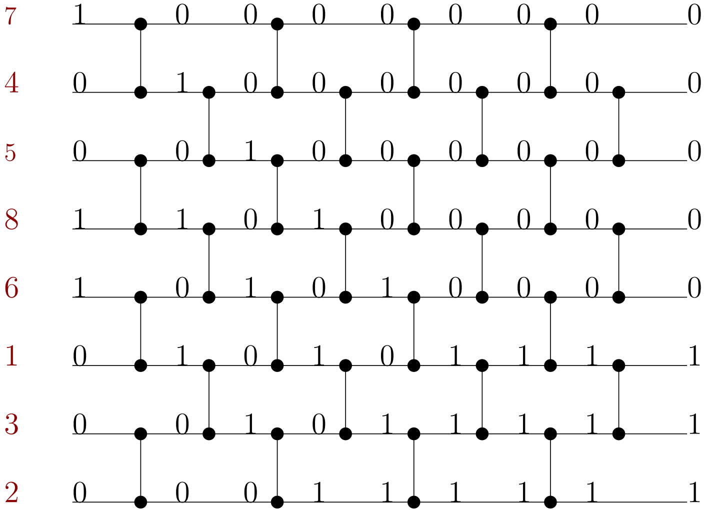
0,1-Prinzip

Vorteile des 0,1-Prinzips :

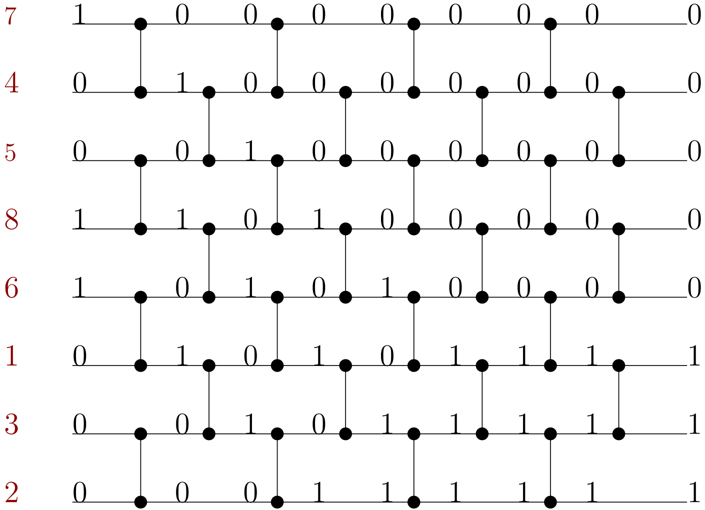
- einfach
  - sauberer Testcode
  - Sicherheit
- Anzahl der Testfälle sinkt

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

0,1- Beispiel

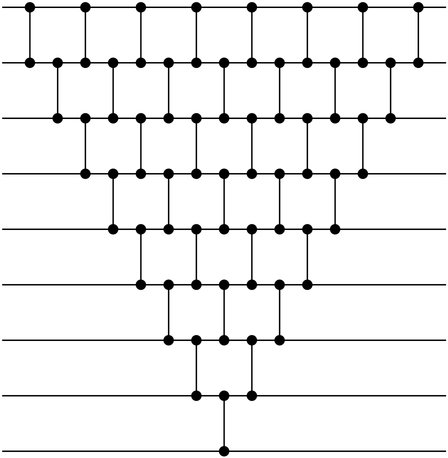

7 1 0 0 0 0 0 0 0 0
4 0 1 0 0 0 0 0 0 0
5 0 0 1 0 0 0 0 0 0
8 1 1 0 1 0 0 0 0 0
6 1 0 1 0 1 0 0 0 0
1 0 1 0 1 0 1 1 1 1
3 0 0 1 0 1 1 1 1 1
2 0 0 0 1 1 1 1 1 1

0,1- Beispiel

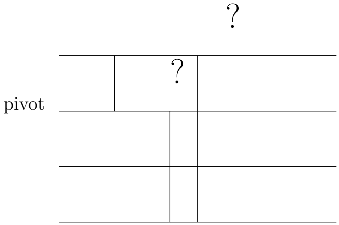


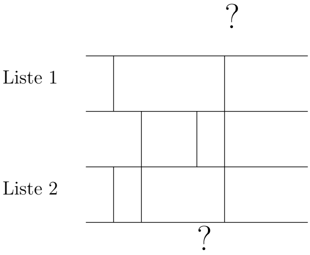
Beispiel an der Tafel ?

Bubblesort im Komparatornetz

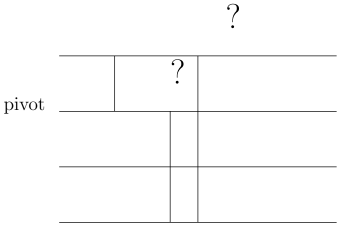


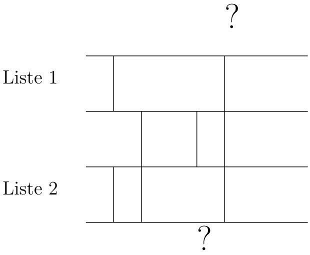
Mergesort Quicksort





Mergesort Quicksort





Quicksort : wo ist das Pivot Element ?
Mit welchem Element müssen wir nun vergleichen?

Mergesort Quicksort

Quicksort : wo ist das Pivot Element ?
Mit welchem Element müssen wir nun vergleichen?
Mergesort : Wo ist nun das größte Element ?
welcher Vergleich kommt nun?)

effektiveres Netzwerk

effektiveres Netzwerk

Aufgabe :

- Resultat soll sortierte Ausgabe sein
- soll effizient sein

effektiveres Netzwerk

Aufgabe :

- Resultat soll sortierte Ausgabe sein
- soll effizient sein

grundlegendes Prinzip :

- intuitiver Einsatz von Vergleichen

  + Einbezug von Teile und Herrscher

Zur Entwicklung verwenden wir Odd-Even-Sort

Zur Entwicklung verwenden wir Odd-Even-Sort

Ablauf :

- Voraussetzung : bekommen zwei sortierte Listen

odd even sort

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(even) b(odd) = c und a(odd) b(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(even) b(odd) = c und a(odd) b(even) = d zusammen (Resultat wird rekursiv sortiert)
- sortierte c und d werden indexweise verschachtelt

odd even sort

Zur Entwicklung verwenden wir Odd-Even-Sort

Ablauf :

- Voraussetzung : bekommen zwei sortierte Listen
- trennen in geraden und ungeraden Index
- fassen a(even) b(odd) = c und a(odd) b(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-sort

Beipiel : an der Tafel

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

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

Beispiel : odd-even-sort

Beipiel : an der Tafel

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

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

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

Beispiel : odd-even-sort

Beipiel : an der Tafel

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

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

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

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

Beispiel : odd-even-sort

Beipiel : an der Tafel

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

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

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

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

nach dem verschachteln

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

Beispiel : odd-even-sort

Beipiel : an der Tafel

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

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

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

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

nach dem verschachteln

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

Resultat :

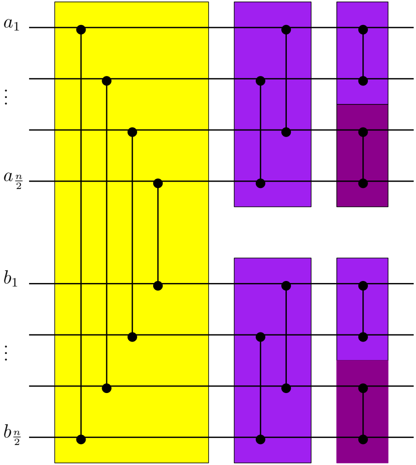
$$1, 2, 3, 4, 5, 6, 7, 8$$

Wieso funktioniert das?

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

$$\begin{aligned} \text{worst}: &\quad O(n^2) \\ \text{best}: &\quad O(n) \end{aligned}$$

Übertragung auf Komparatornetzwerk



Bitonmischer

Ablauf :

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

Biton -Sortierer : Aufbau

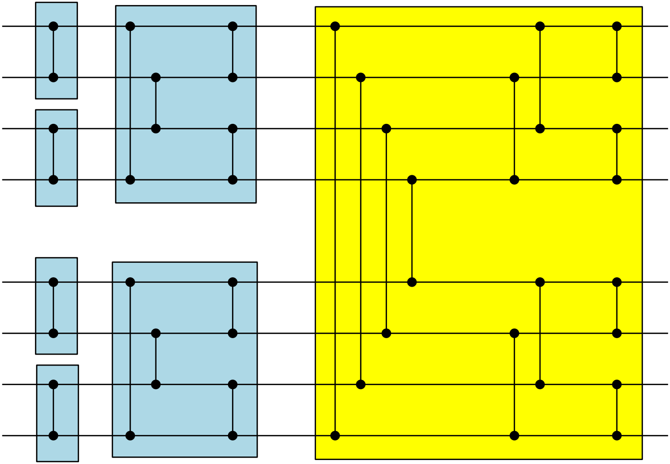


Bild kleiner Zahlenfolge 4-8-16
Beispiel

Herleitung der Laufzeit

N	Anzahl der Schritte
2^1	

Herleitung der Laufzeit

N	Anzahl der Schritte
2^1	1

Herleitung der Laufzeit

N	Anzahl der Schritte
2^1	1
2^2	

Herleitung der Laufzeit

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

Herleitung der Laufzeit

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

Herleitung der Laufzeit

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

Herleitung der Laufzeit

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

Herleitung der Laufzeit

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

Herleitung der Laufzeit

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

Herleitung der Laufzeit

N	Anzahl der Schritte
2^1	1
2^2	$1+2$
2^k	$1+2+3+...+k-1+k=\sum_{i=1}^{k} i$
(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
    $\frac{n}{2}\left(\cdot\frac{1}{2}\cdot\log_2 n\ (\log_2 n+1)\right)$
  - 1 Vergleicher 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
    $\frac{n}{2}\left(\cdot \frac{1}{2} \cdot \log_2 n \ (\log_2 n + 1)\right)$
  - 1 Vergleicher bei Software benötigt
  - Laufzeit konstant
- Abhängigkeit von der Eingabe
- Bezug zum vorherigen Vergleich
- Andere Laufzeiten

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

Motivation

Grundlage des Sortierens

Sortiernetzwerk

Laufzeit

Zusammenfassung

Ausblick

Zusammenfassung

- paralleles sortieren ist schnell und effizient
- problemabhängige Lösung

Ausblick

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

- *Taschenbuch der Algorithmen*.
  Springer Verlag , 2008.

- Tom Leighton.
  Einführung in Parallele Algorithmen und Architekturen
  Gitter, Bäume und Hypercubes.
  Thomsom Publisching , 1997.
  3-8266-0248-X

- Laufzeiten Sortieralgorithmen
  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?