The Perfect Shuffle –

Sortieren auf Multiprozessorrechnern

von Diego Semmler

Inhaltsverzeichnis

Was sind Sortiernetze?

Es gibt verschiedene Sortierverfahren, z.B.

- Heapsort
- Quicksort
- Mergesort
- Bitonic Sort

Alle Sortierverfahren basieren auf dem Vergleich zweier Elemente.

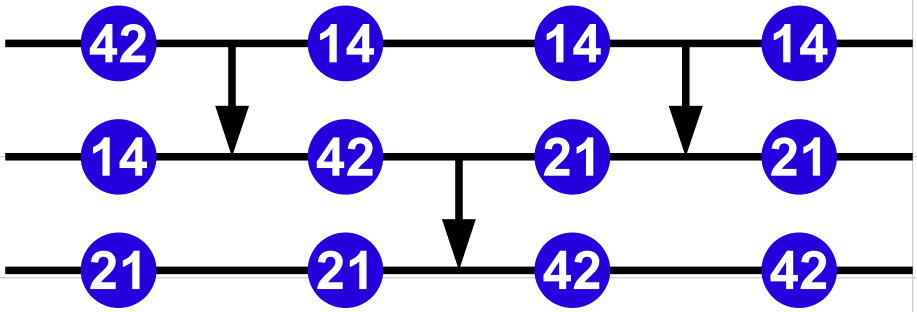
Durch mehrere Vergleiche ist es auch möglich Mengen mit mehr als zwei Elementen zu sortieren.

Wenn die Reihenfolge der Vergleiche nicht von der Eingabemenge abhängt, sprechen wir von einem **Sortiernetz**.

Bitonic Sort ist ein solches Sortiernetz. Quicksort und Heapsort nicht.

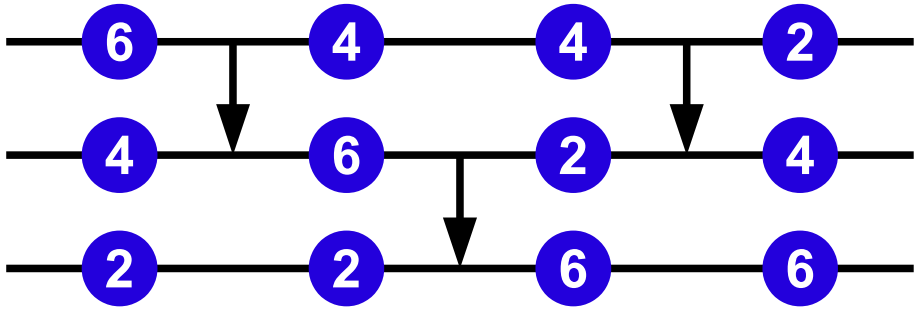
Was sind Sortiernetze?

Sortiernetzwerken benötigen nur eine Aktion:
Zwei Elemente werde miteinander verglichen und ggf. ausgetauscht.
Wir stellen dies mit einem Pfeil dar, wobei nach der Aktion das größere Element an der Pfeilspitze steht.



Beispiel für ein intuitives Sortiernetzwerk mit 3 Elementen. Die Anzahl der Elemente wird im folgenden mit **n** bezeichnet.

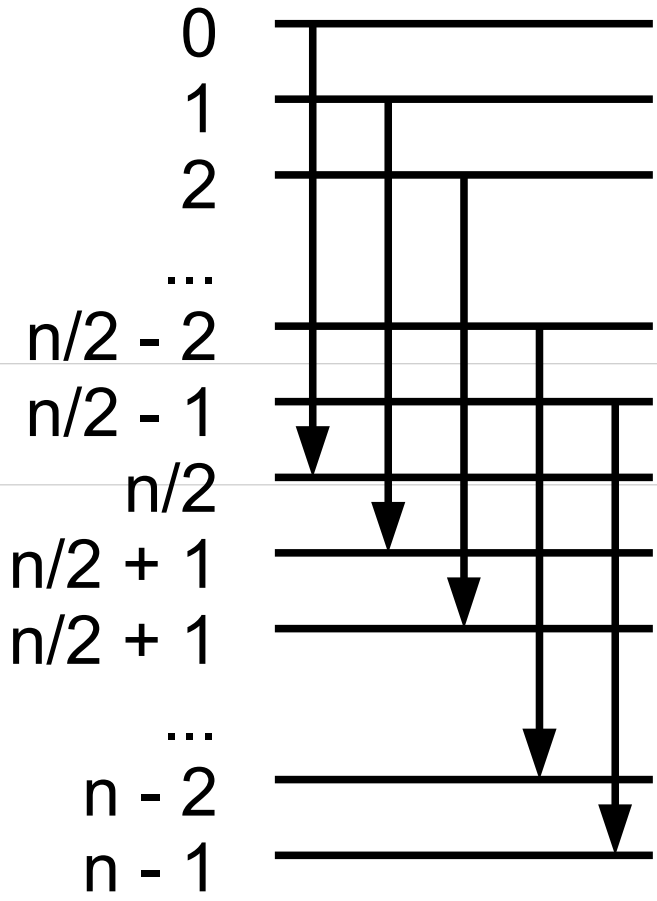
Wie bei jedem Sortiernetz hängt die Reihenfolge der Vergleiche nicht von der Eingabemenge selbst ab.



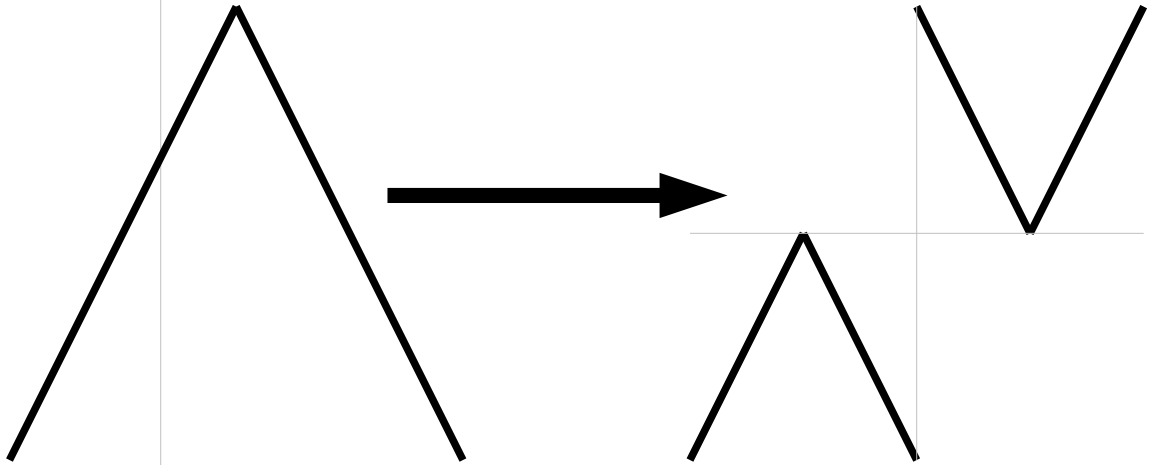
Egal welche Zahlen am Anfang stehen, am Ende sind sie immer aufsteigend sortiert.

Die Idee von Bitonic Sort

Betrachten wir einmal folgendes Sortiernetz B_n:

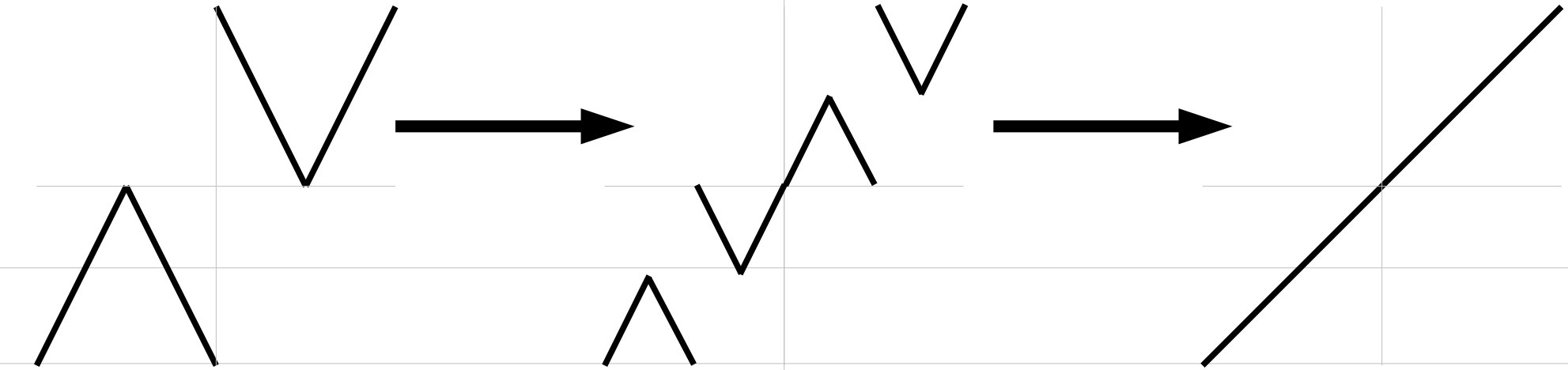


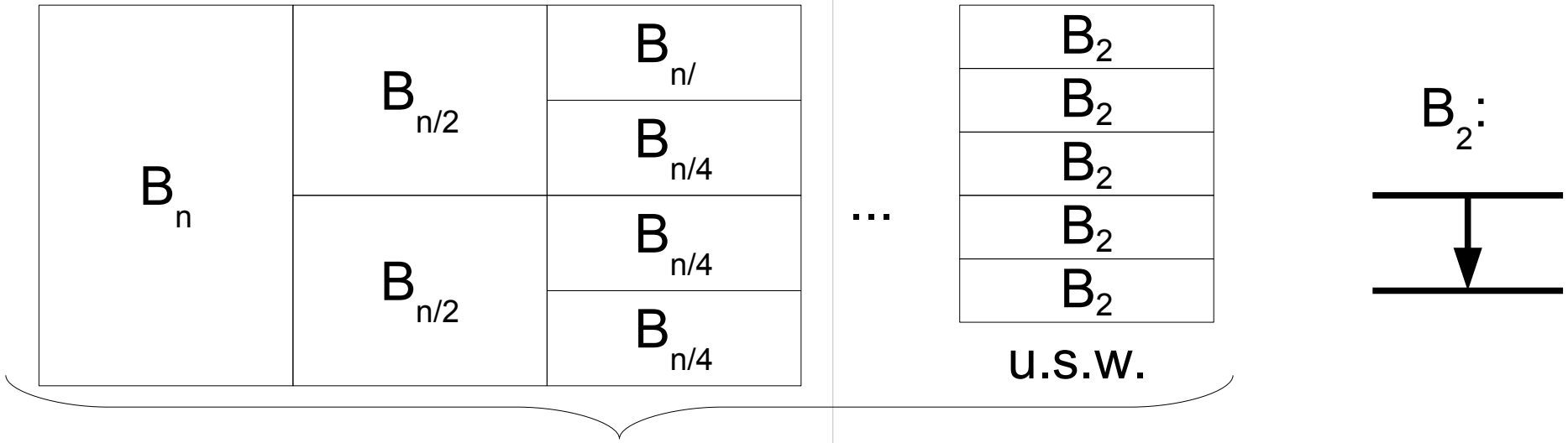
Wenn wir eine **bitonische Folge** haben, das heißt eine Folge, die zunächst aufsteigend und dann absteigend sortiert ist, dann erhalten wir nach Anwendung zwei bitonische Folgen, wobei das größte Element der ersten Folge kleiner gleich dem kleinsten Element der zweiten Folge ist.



Die Idee von Bitonic Sort

Wenden wir das Schema rekursiv an, erhalten wir eine monotone Folge, d.h. unsere Folge ist sortiert.





B_2:

Die Idee von Bitonic Sort

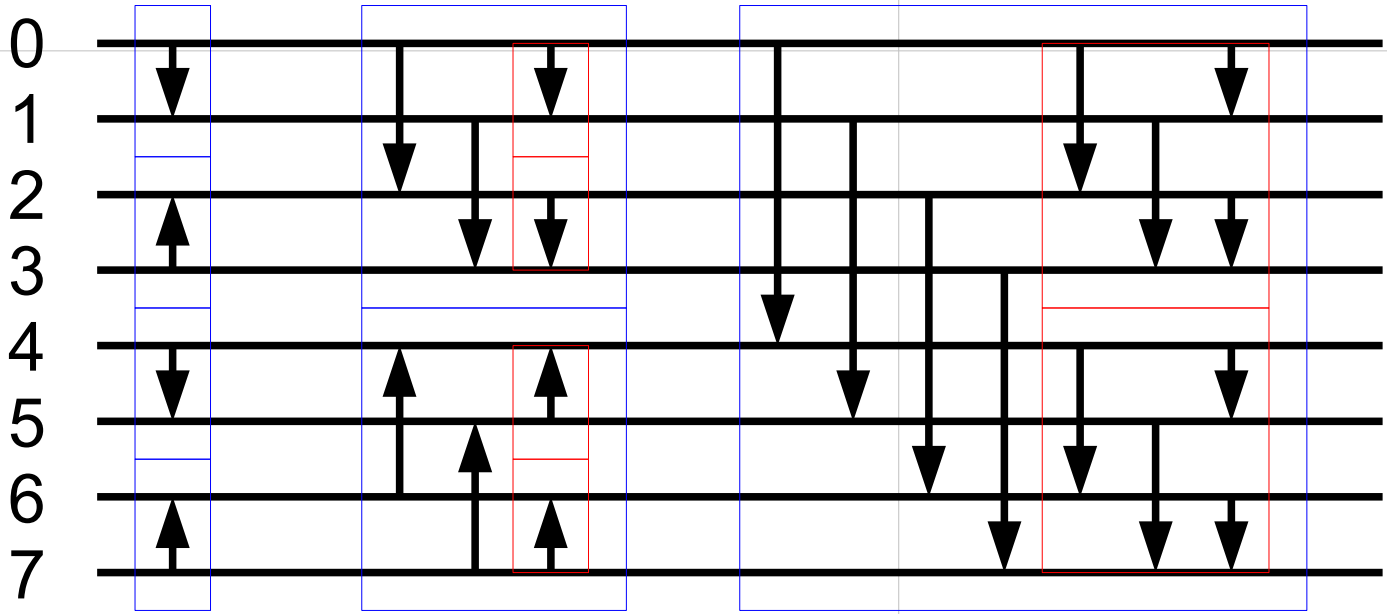
Frage: Woher bekommen wir eine bitonische Folge?

Lösung: Wir sortieren vor.

Hier ist ein Beispiel für n = 8:

<table>
<tr><td>M_2</td><td rowspan="2">M_4</td><td rowspan="4">M_8</td></tr>
<tr><td>M_2</td></tr>
<tr><td>M_2</td><td rowspan="2">M_4</td></tr>
<tr><td>M_2</td></tr>
</table>

Jetzt können wir das bitonische Sortiernetz für n = 8 aufbauen:



Phase: 1 2 3

Komplexität von Bitonic Sort

Für die Komplexität der M_n-Schritte gilt folgendes:

$$M_n = \frac{n}{2} + 2M_{n/2} = \frac{n}{2} + 2\frac{n}{4} + 4\frac{n}{8} + \ldots = lb(n) \cdot \frac{n}{2}$$

Damit kann die Anzahl der Vergleiche V(n) errechnet werden:

$$\begin{aligned} V(n) &= \frac{n}{2}M_2 + \frac{n}{4}M_4 + \frac{n}{8}M_8 + \ldots + M_n \\ &= \frac{n}{\cancel{2}} \cdot 1 \cdot \frac{\cancel{2}}{2} + \frac{n}{\cancel{4}} \cdot 2 \cdot \frac{\cancel{4}}{2} + \frac{n}{\cancel{8}} \cdot 3 \cdot \frac{\cancel{8}}{2} + \ldots + lb(n) \cdot M_n = 1 \cdot \frac{n}{2} + 2 \cdot \frac{n}{2} + 3 + \ldots + lb(n) \cdot \frac{n}{2} \\ &= \frac{n}{2}(1 + 2 + 3 + \ldots + lb(n)) = \frac{n}{2}\frac{lb(n)[lb(n)+1]}{2} = \underline{\underline{\frac{n}{4}lb(n)[lb(n)+1]}} \end{aligned}$$

$$\Rightarrow V(n) \in O(n \cdot \log(n)^2)$$

Also Bitonic Sort ist kein optimales Sortiernetz, denn dazu müsste es eine komplexität von n log(n) haben.

Beispiel für Bitonic Sort

Der Diego Musicfinder sortiert seine Playlisten mit Bitonic Sort.

Wichtige Größen

Die maßgebende Größe für Algorithmen auf Singelprozessorrechnern ist die Laufzeit $t_s(n)$.

Wenn wir uns über **Multiprozessorrechner** unterhalten, spielen folgende Größen eine wichtige Rolle:

- Laufzeit: $t(n)$
- Prozessoranzahl: $p(n)$
- Kosten: $c(n) \equiv t(n) \bullet p(n)$
- Effizienz: $e(n) = t_s(n) / c(n)$

$t_s(n) \leq c(n)$

$e(n) \leq 1$

Hierbei gilt, dass die Laufzeit des effizientesten Singleprozessoralgorithmus nicht größer sein kann als die Kosten eines paralelen Algorithmus, denn sonst könnte man mit einem Prozessor mehrere simulieren und hätte so einen besseren Singleprozessoralgorithmus.

Das Perfect Shuffle

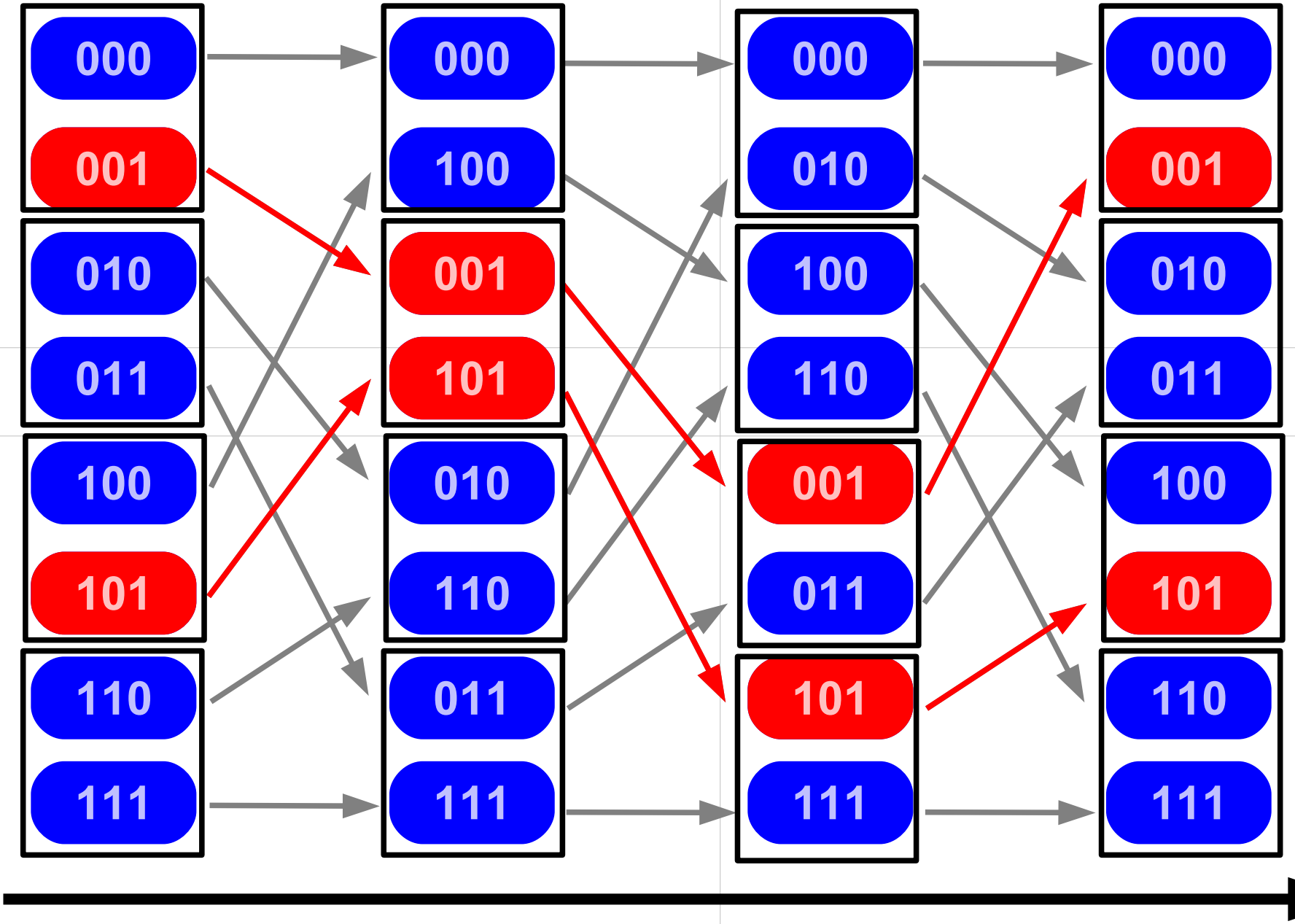
Stellen wir uns ein Feld mit $n = 2^m$ Elementen vor und geben jedem Feld eine binäre Nummer.
Jetzt verbinden jedes Feld mit einem anderen: Indem wir das erste Bit einer Zahl abhängen und hinten anfügen.
Beispiel mit m = 6:

42 = 101010 -> 1-01010 -> 01010-1 -> 010101 = 21

35 = 100011 -> 1-00011 -> 00011-1 -> 000111 = 7

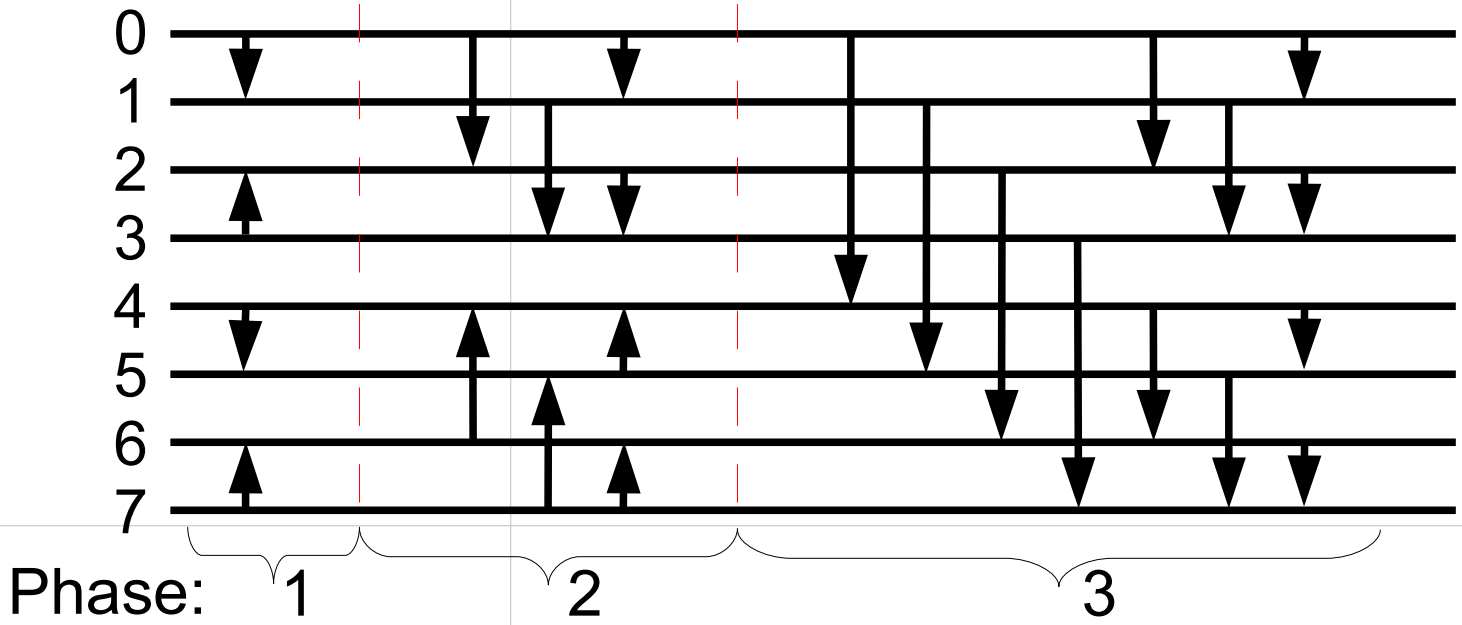
Übrigens:
Jedes Element ist nach m Schritten wieder an seinem Platz, weil die Zahlenfolge ein mal „durchrotiert“ ist.

Das Perfect Shuffle

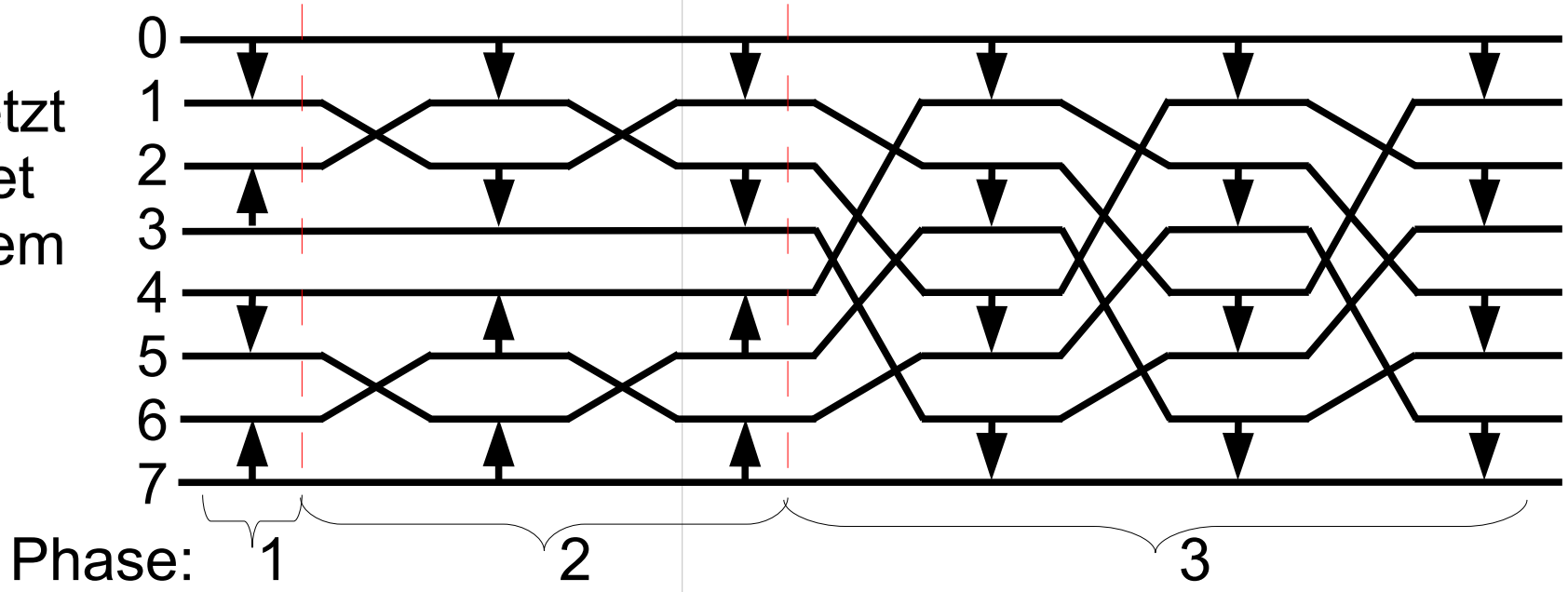


Sortieren mit Perfect Shuffle

Noch ein mal zur Erinnerung das bitonische Sortiernetzwerk für n = 8:



Und jetzt verknotet mit dem perfect Shuffle:



Ein Pseudoalgorithmus

Wir benötigen n/2 durchnummerierte Prozessoren mit jeweils folgende Variablen:

m wird vor dem sortieren an alle Rechner verteilt, wobei 2^m die Problemgröße ist.

s als Zählervariable für die Phase

i als Zählervariable für den Schritt innerhalb der Phase.

```
For s = 1 To m
        For i = 1 To s
            Shuffle(s)
            Sort(b_s)
            Next
        Next
```

Wobei Shuffle(s) bedeutet, dass die Daten mithilfe des Shuffels der Größe 2^s weitergegeben werden.

Sort(0) die beiden Elemente in einem Prozessor aufwärs, Sort(1) abwärts sortiert.

b_s ist das s. Bit der Prozessornummer.

Analyse

```
For s = 1 To m
    For i = 1 To s
        Shuffle(s)
        Sort(b_s)
        Next
    Next
```

Uns interessiert wieder die Anzahl der Vergleiche, vor allem um sie mit der Singleprozessorvariante zu vergleichen.

m = lb(n) => Die innere Schleife wird lb(n) mal aufgerufen.
Damit gilt für die Anzahl der Vergleiche pro Prozessor folgendes:

$$V(n) = (1+2+3+...+lb(n)) = \frac{\ln(n)[lb(n)+1]}{2} = \frac{1}{2}lb(n)[lb(n)+1]$$

Wir verwenden n/2 Prozessoren, daraus ergibt sich die gesamte Anzahl der Vergleiche, sowie alle weiteren interesanten Daten:

$$V_{ges}(n) = \frac{n}{4}lb(n)[lb(n)+1] = V_{Singleprozessor}(n)$$

=> Die Umsetzung ist optimal.

Analyse

$p(n) \in O(n)$
$t(n) \in O(\log(n)^2)$
$c(n) \in O(n) \bullet O(\log(n)^2) = O(n \log(n)^2)$

Bei t(n) muss man bedenken, dass t(n) € =(n) ist, falls man eine serielle Eingabe vornimmt, d.h. die Elemente nacheinander einliest.

Außerdem berücksichtigt die Analsys die Längen der Leitungen nicht. Das ist in der Praxis meistens auch nicht relevant, kann aber bei **sehr großen** n durchaus eine Rolle spielen.

Vor- & Nachteile des Verfahrens

Vorteile:

- Die Steigerung der Zeit ist niedrig. $O(\log(n)^2)$
- Selbst große Problemgrößen können in akzeptebler Zeit geköst werden.

Nachteile:

- Das Verfahren ist nicht kostenoptimal.
- Es werden sehr viele Prozessoren benötigt. (n/2)
- Die Prozessoranzahl ist nicht variabel, sondern allein von der Problemgröße abhängig.
- Wenn ein Prozessor ausfällt, scheitert das ganze Verfahren.

Obwohl diese Variante von Bitonic Sort sehr schnell ist, ist sie für die Praxis zu teuer und nicht flexibel genug.

Halbbitonisches Sortieren

Angenommen, wir haben jetzt statt n nur noch p/2 Prozessoren zur Verfügung, wobei n und p jeweils eine Potenz von 2 sind und p < n ist. Jeder Prozessor soll eine Sequenz der Länge n/p in der Zeit O(n log(n)) sortieren können, sowie mit einen Merge-Agorithmus zwei Sequenzen der Länge n/p zusammenführen können.

Der ursprüngliche Algorithmus muss so abgeändert werden, dass zunächst sortierte Folgen der Länge n/p geschaffen werden. Sort(a; dir) soll n/p Elemente, beginnend mit a in Richtung dir sortieren.
Shuffle(s) gibt statt zwei Elementen zwei Sequenzen der Länge n/p weiter und Merge(b_s) führt zwei Sequenzen der Länge n so zusammen, dass sie in Richtung b_s sortiert sind.

```
Sort(0; 0)
Sort(n/p; 1)
For s = 1 To lb(p)
        For i = 1 To s
            Shuffle(s)
            Merge(b_s)
        Next
    Next
```

Analyse

Sort(a; dir) besitzt eine Komplexität von O(n/p log(n/p)).
Shuffle(s) hat die Komplexität O(n/p) und
Merge(b_s) habe eine Komplexität von O(n/p).

Daraus folgt direkt die Komplexität des gesamten Algorithmus:
t(n) € O(2 • n/p log(n/p)) + O(log(p)²) • O(n/p)
€ O(n/p log(n/p) + n/p log(p)²)

c(n) = t(n) • p € O(n log(n/p) + n log(p)²)

Also ist dieser Algorithmus kostenoptimal, wenn

$$\log(p)^2 \leq \log(n) \Leftrightarrow lb(p) \leq lb(n)^{1/2} \Leftrightarrow$$
$$p \leq 2^{lb(n)^{1/2}}$$

p	Minimales n
1	1
2	4
3	27
4	256
5	3125
6	46.656
7	823.543
8	16.777.216
9	387.420.489
10	10.000.000.000

Vor- & Nachteile dieser Methode

Vorteile:

- Die Steigerung der Zeit ist niedrig.
- Selbst große Problemgrößen können in akzeptabler Zeit gelöst werden.
- Die Prozessoranzahl ist variabel.
- Es müssen nicht n/2 Prozessoren verwendet werden.
- Das Verfahren kann kostenoptimal durchgeführt werden.

Nachteile:

- Das Verfahren ist nicht für beliebig viele Prozessoren kostenoptimal.
- Wenn ein Prozessor ausfällt, kann mit der Hälfte der Kapazität weiter sortiert werden.

Diese Variante ist deutlich flexibler und billiger in der Anschaffung. Sie funktioniert auch auf wenigen Prozessorkernen und könnte schon heute auf Heimcomputern mit mehreren Prozessorkernen implementiert werden.

Quellen

Informationsquellen:

- Selim G. Atel - Parallel Sorting Algorithms
- http://iti.fh-flensburg.de/lang/algorithmen/sortieren/networks/sortier.htm
- http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonic.htm

Die Präsentation kann auf meiner Homepage **www.dsemmler.de** heruntergeladen werden.