Algorithmen in Zellularautomaten

5. Sortieren in eindimensionalen Zellularautomaten

Thomas Worsch

Fakultät für Informatik
Institut für Theoretische Informatik

Sommersemester 2018

Ziele

- einfaches Sortieren mit einem Datum je Zelle
- Knuths 0-1-Sortierlemma

Ziele

- einfaches Sortieren mit einem Datum je Zelle
- Knuths 0-1-Sortierlemma

Problemstellung

Vereinbarung

für $x \in A$, $w \in A^*$ bezeichne

$N_x(w)$ die Anzahl der Vorkommen des Symboles x in dem Wort w.

Aufgabe: Sortieren von Bits

Gegeben: $A = \{0, 1\}$ $R = \mathbb{Z}$ $N = H_1$

Gesucht: Zellularautomat mit Q und δ, so dass jedes Muster $w \in A^+$ in das Muster $0^{N_0(w)}1^{N_1(w)}$ überführt wird.

Algorithmus

$Q = A \cup \{\square\}$

$\delta =$???

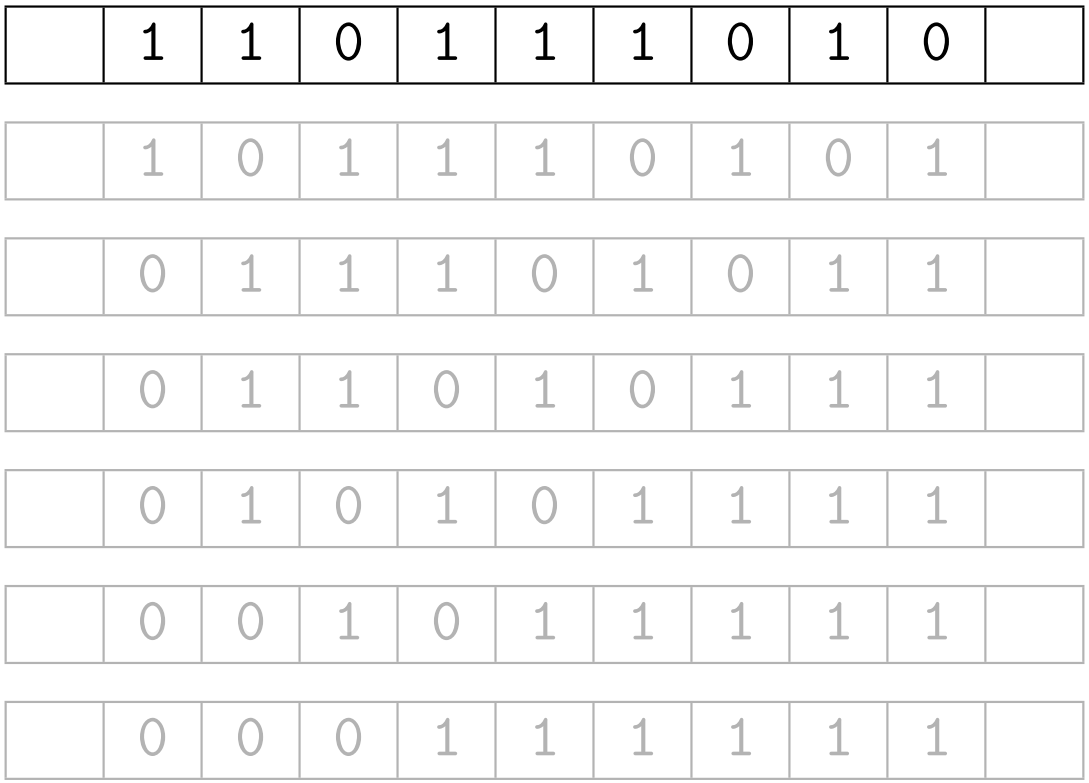
Algorithmus

$Q = A \cup \{\square\}$
δ gegeben durch die folgende Tabelle:

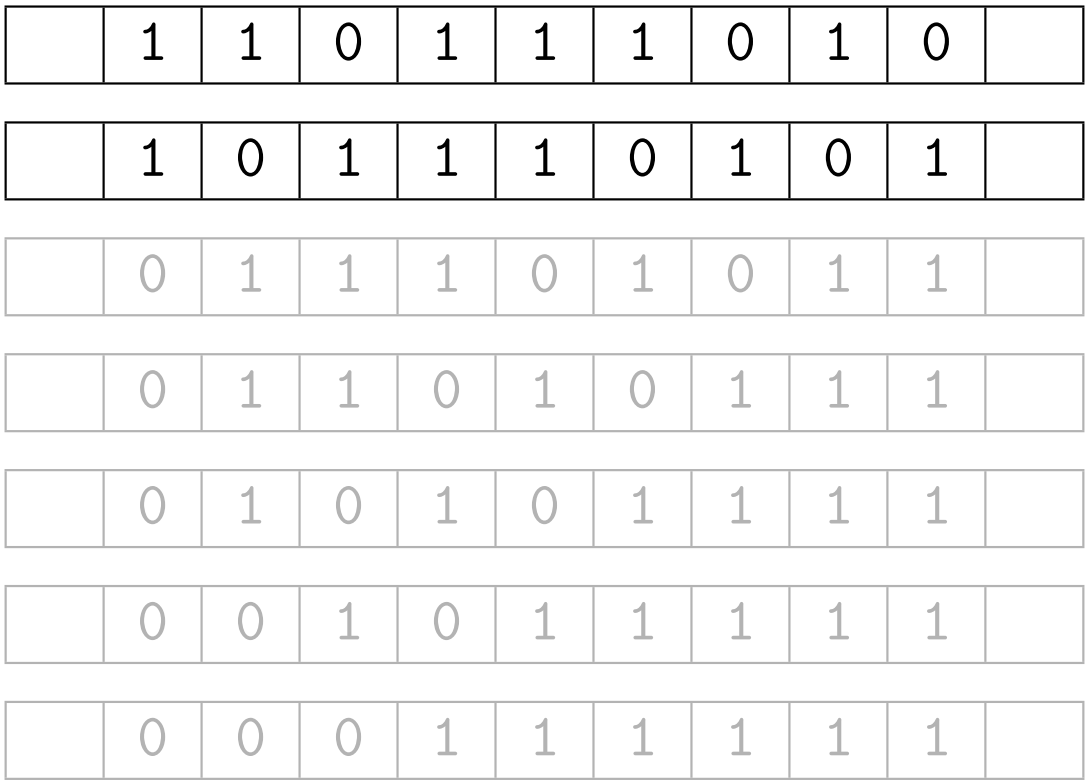
$\ell(-1)$	$\ell(0)$	$\ell(1)$	$\delta(\ell)$
1	0	$\times$	1
$\times$	1	0	0
und in allen anderen Fällen			
$\times$	s	$\times$	s

Beachte: δ ist wohldefiniert.

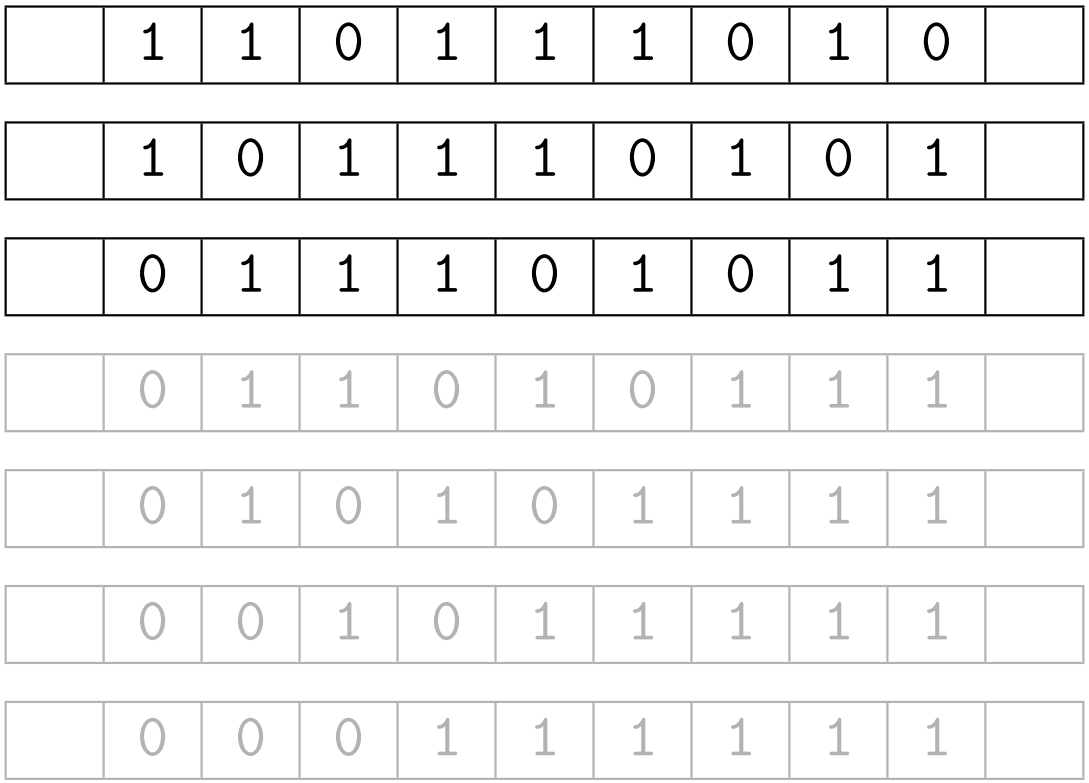
Beispiel


1 1 0 1 1 1 0 1 0
1 0 1 1 1 0 1 0 1
0 1 1 1 0 1 0 1 1
0 1 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

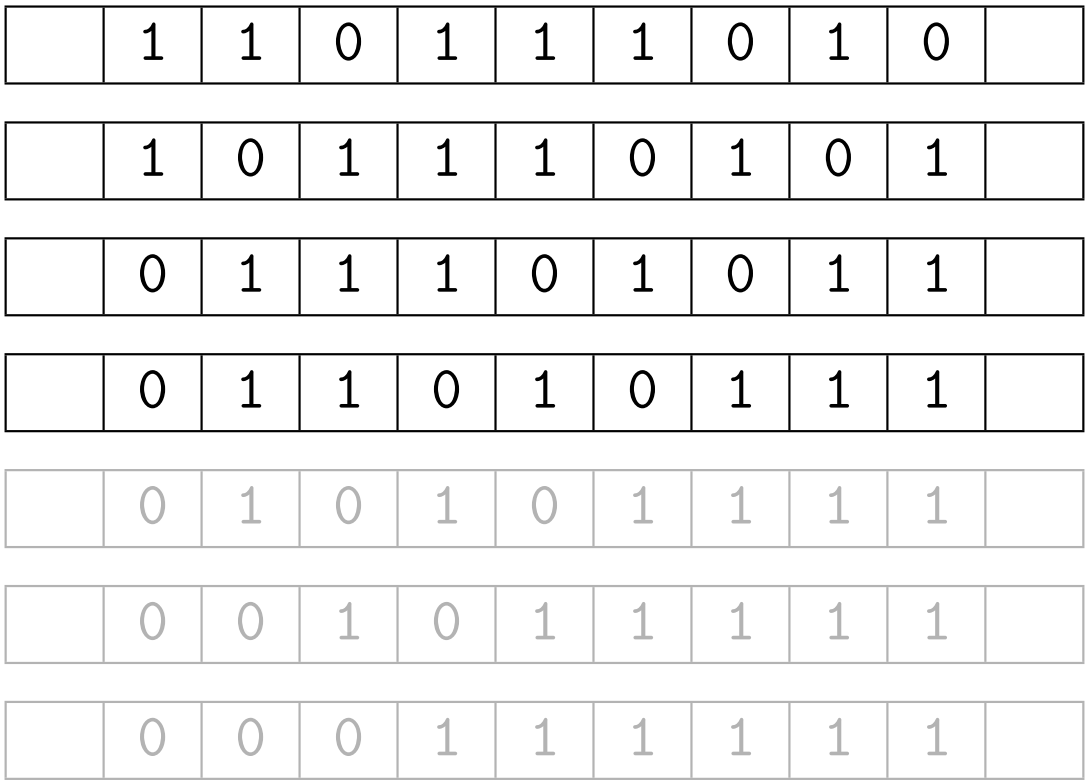
Beispiel


1 1 0 1 1 1 0 1 0
1 0 1 1 1 0 1 0 1
0 1 1 1 0 1 0 1 1
0 1 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

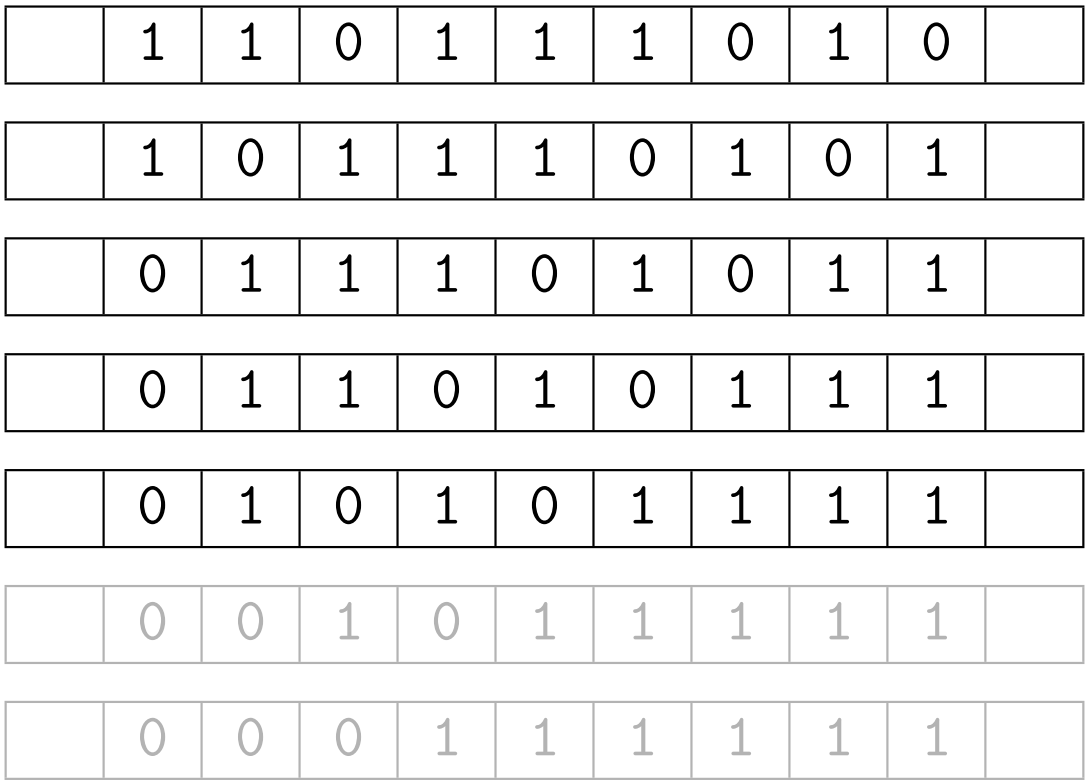
Beispiel


1 1 0 1 1 1 0 1 0
1 0 1 1 1 0 1 0 1
0 1 1 1 0 1 0 1 1
0 1 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

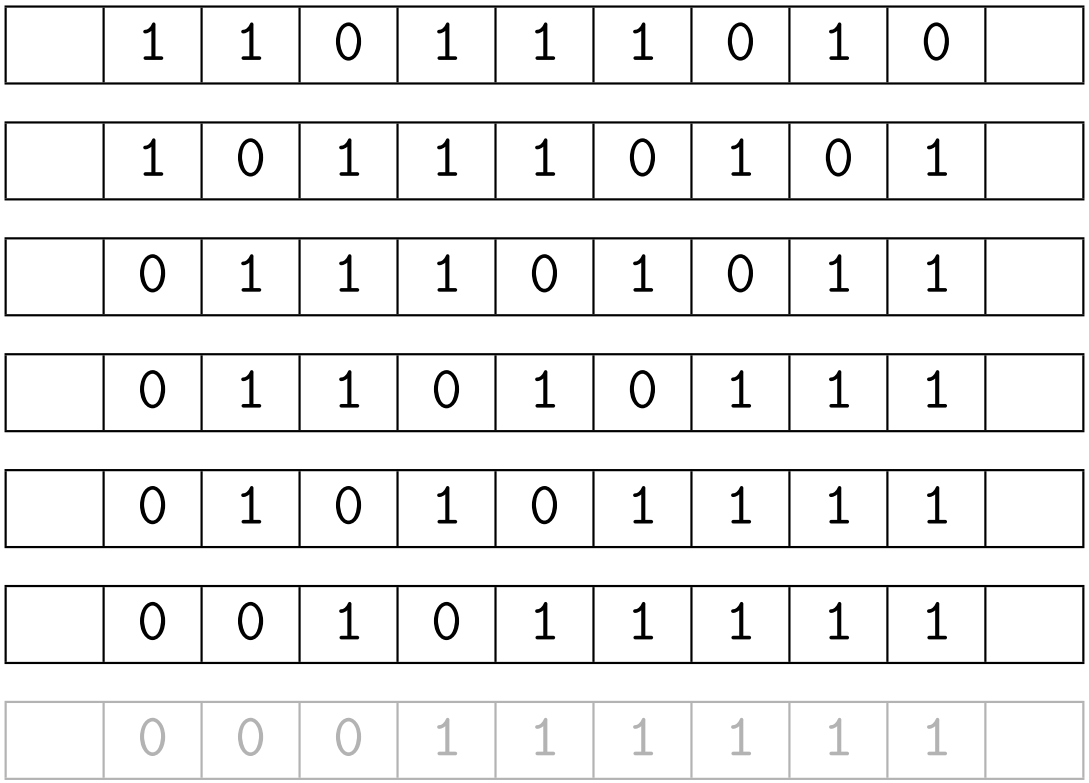
Beispiel


1 1 0 1 1 1 0 1 0
1 0 1 1 1 0 1 0 1
0 1 1 1 0 1 0 1 1
0 1 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

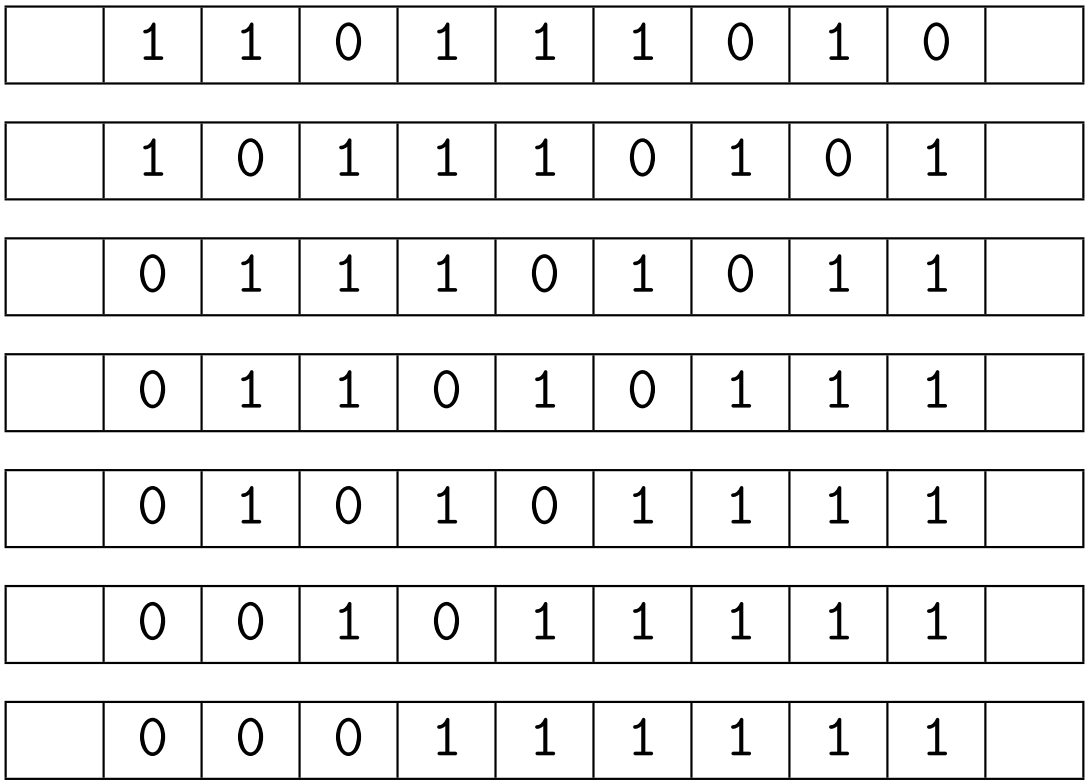
Beispiel


1 1 0 1 1 1 0 1 0
1 0 1 1 1 0 1 0 1
0 1 1 1 0 1 0 1 1
0 1 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

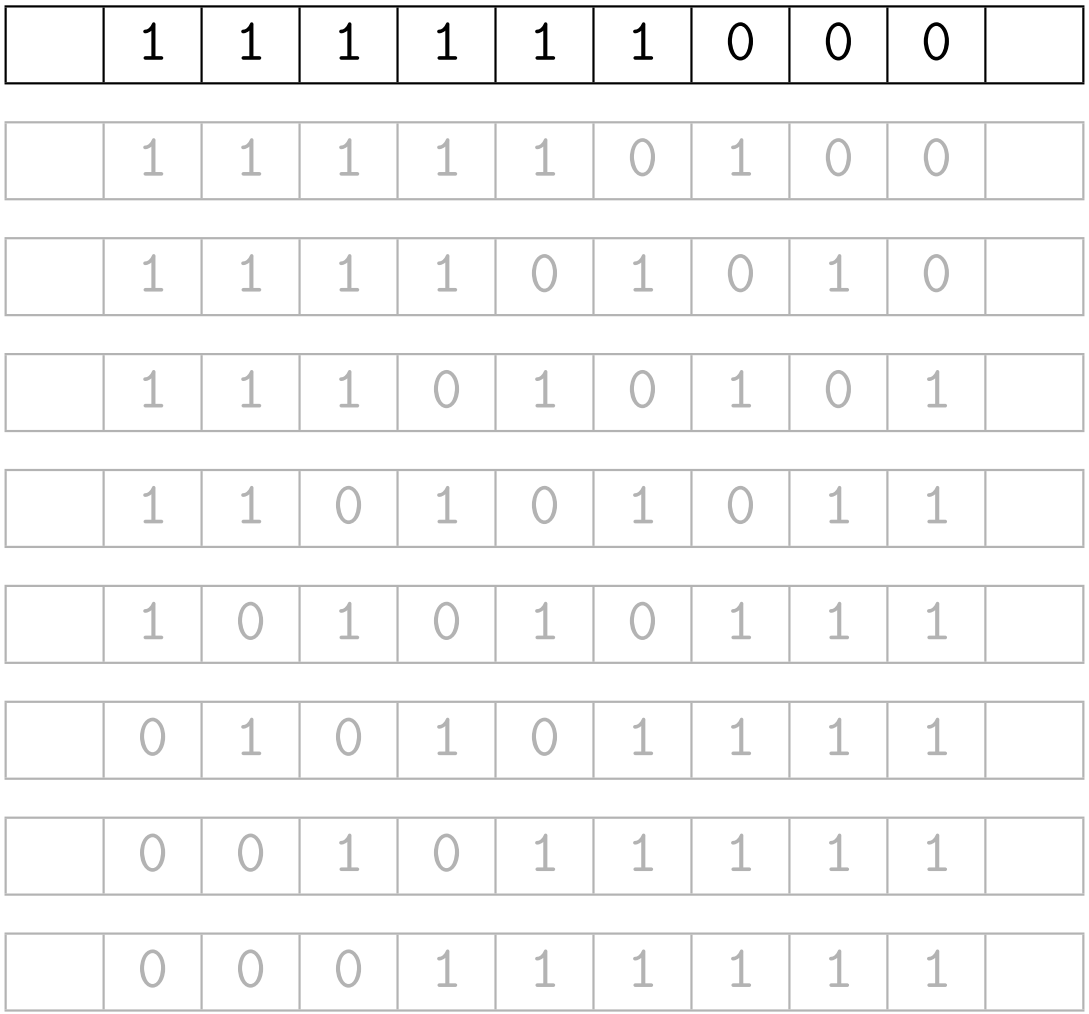
Beispiel


1 1 0 1 1 1 0 1 0
1 0 1 1 1 0 1 0 1
0 1 1 1 0 1 0 1 1
0 1 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

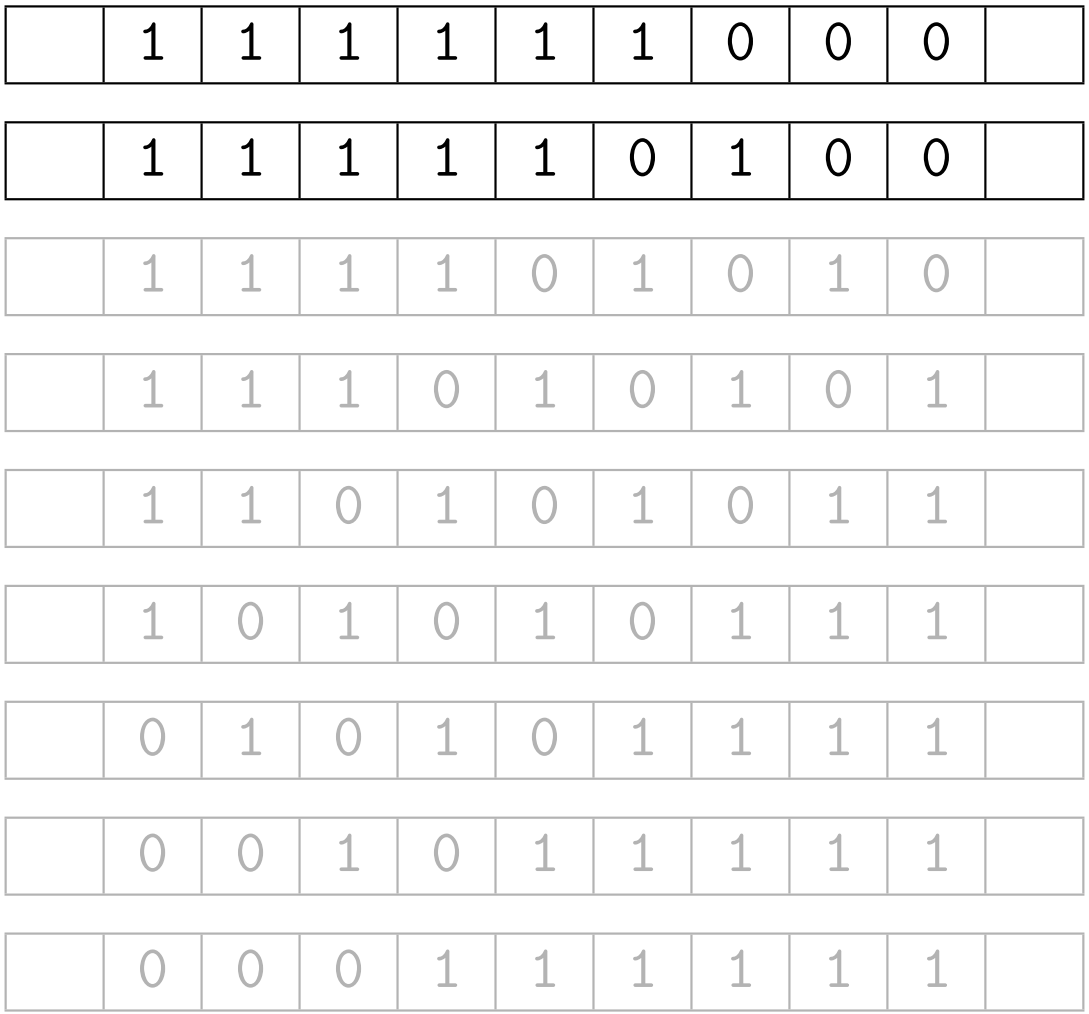
Beispiel


1 1 0 1 1 1 0 1 0
1 0 1 1 1 0 1 0 1
0 1 1 1 0 1 0 1 1
0 1 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

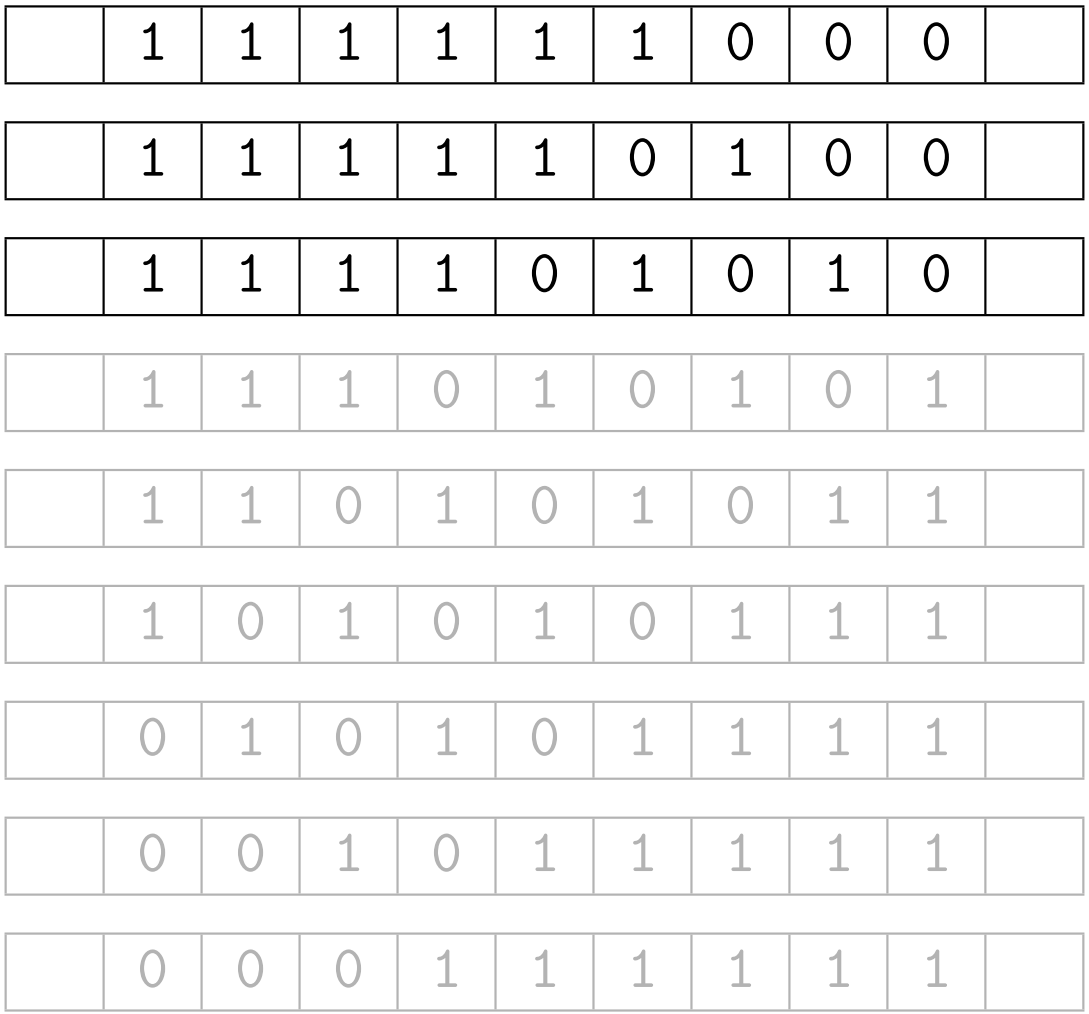
Beispiel 2


1 1 1 1 1 1 0 0 0
1 1 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0
1 1 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 1
1 0 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

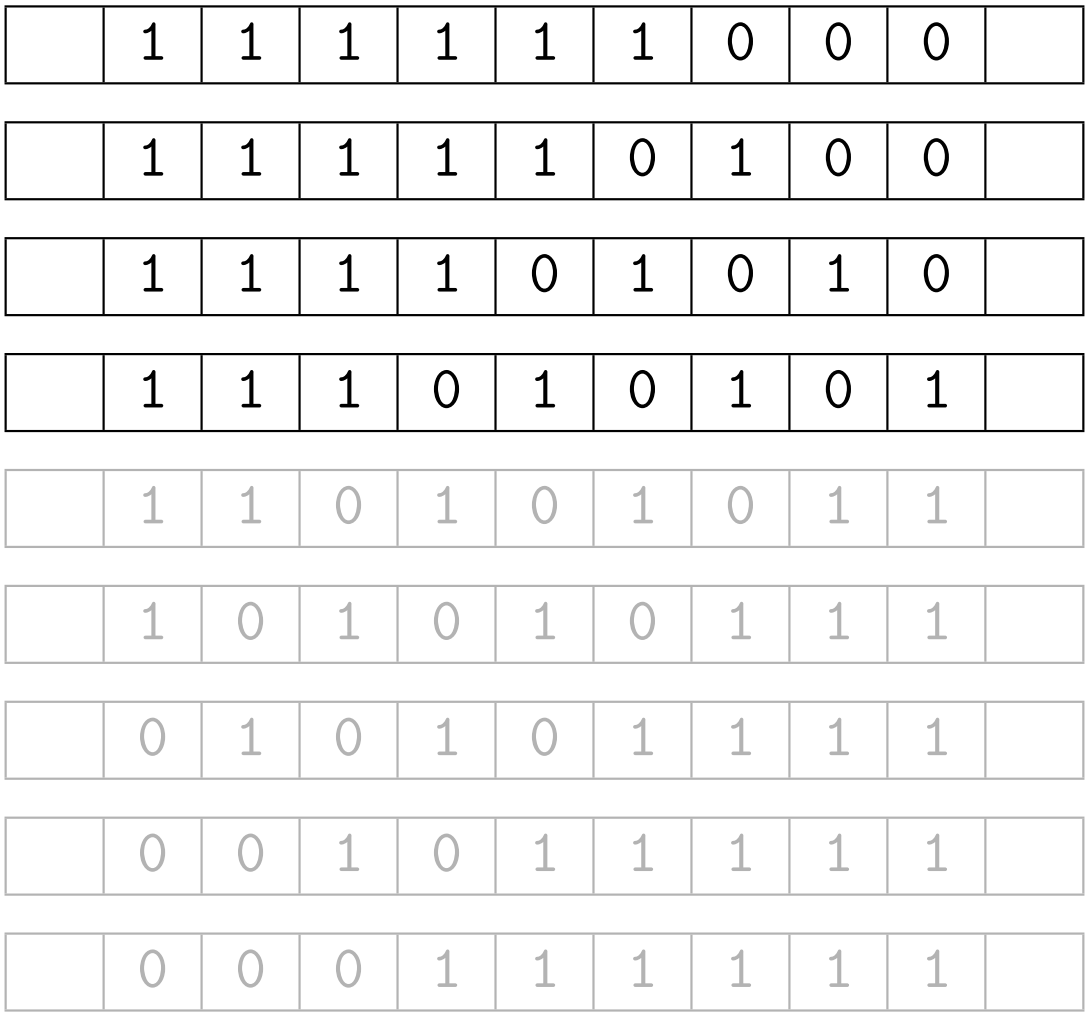
Beispiel 2


1 1 1 1 1 1 0 0 0
1 1 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0
1 1 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 1
1 0 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

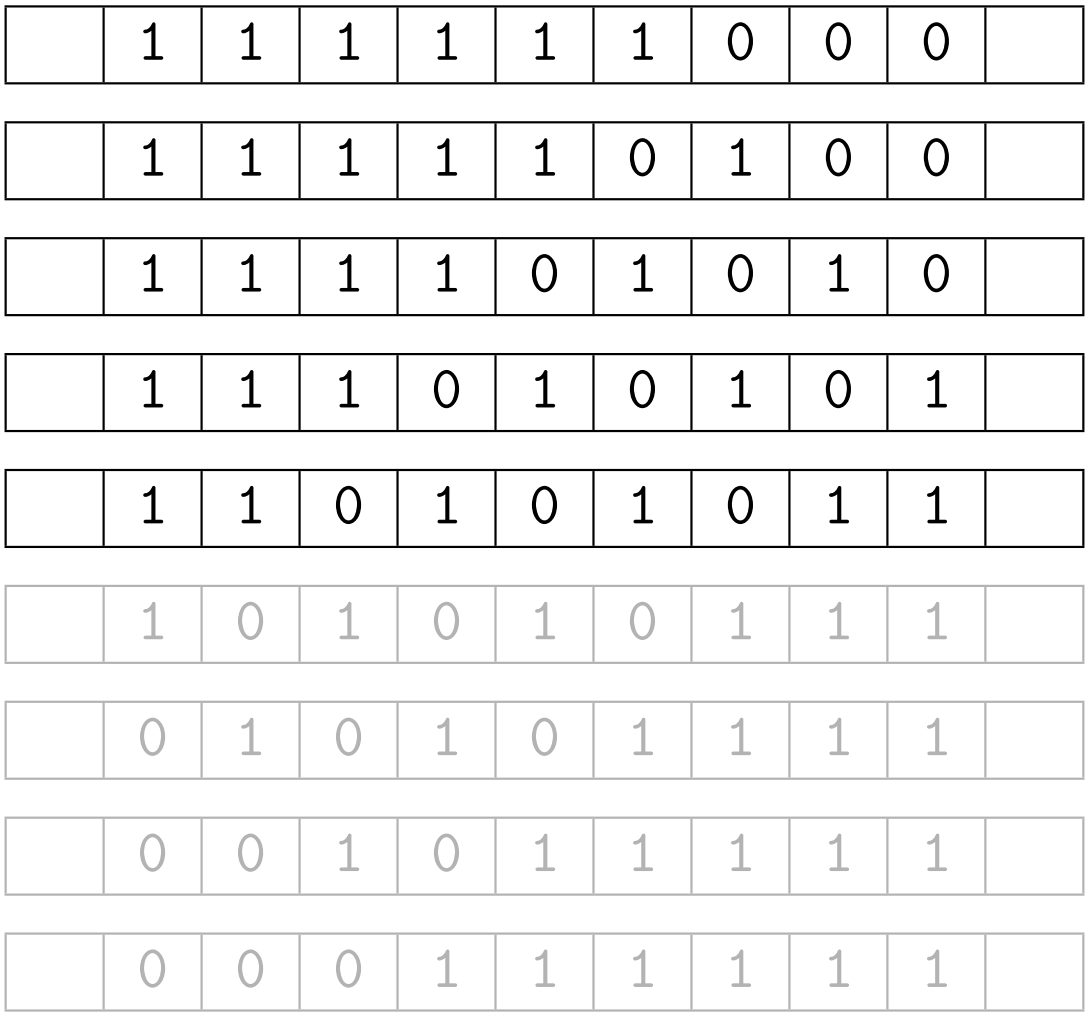
Beispiel 2


1 1 1 1 1 1 0 0 0
1 1 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0
1 1 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 1
1 0 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

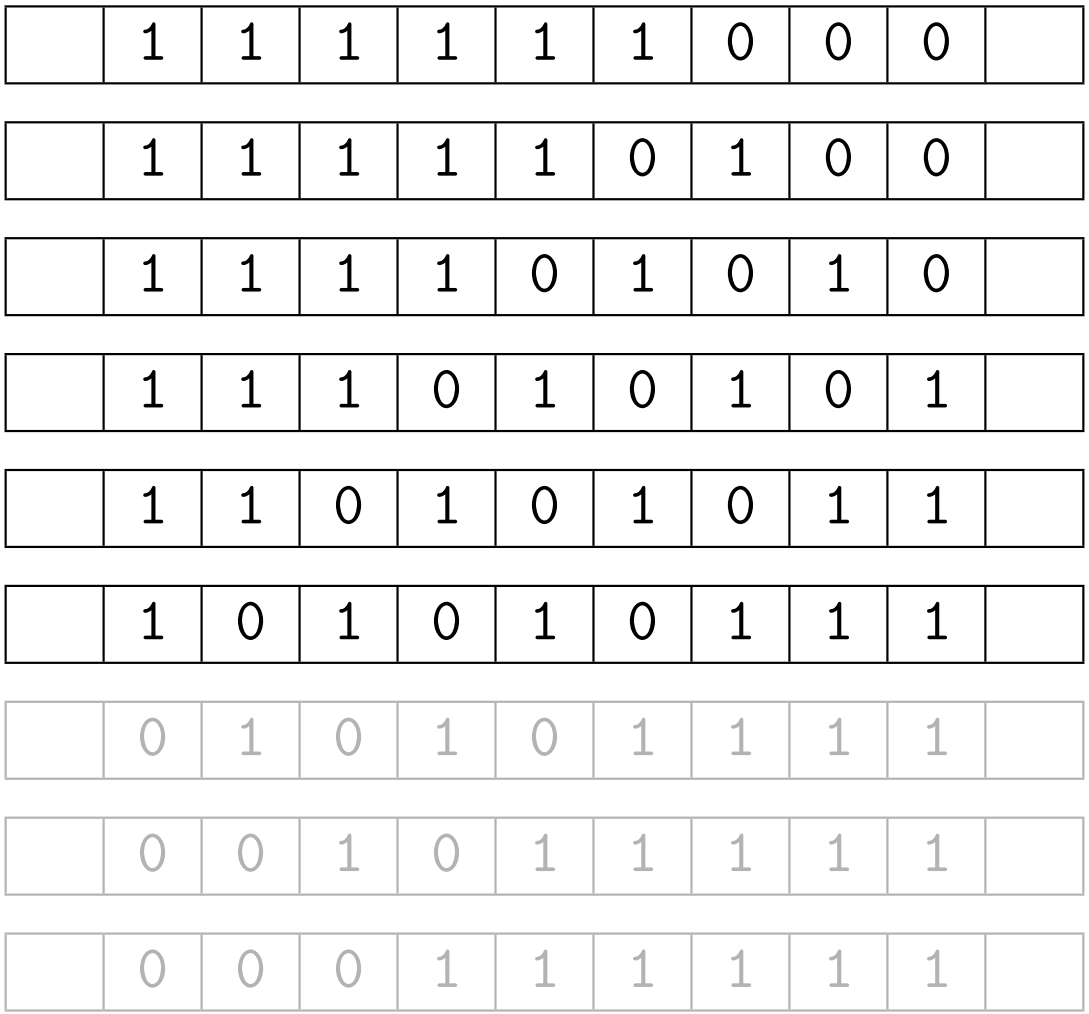
Beispiel 2


1 1 1 1 1 1 0 0 0
1 1 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0
1 1 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 1
1 0 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

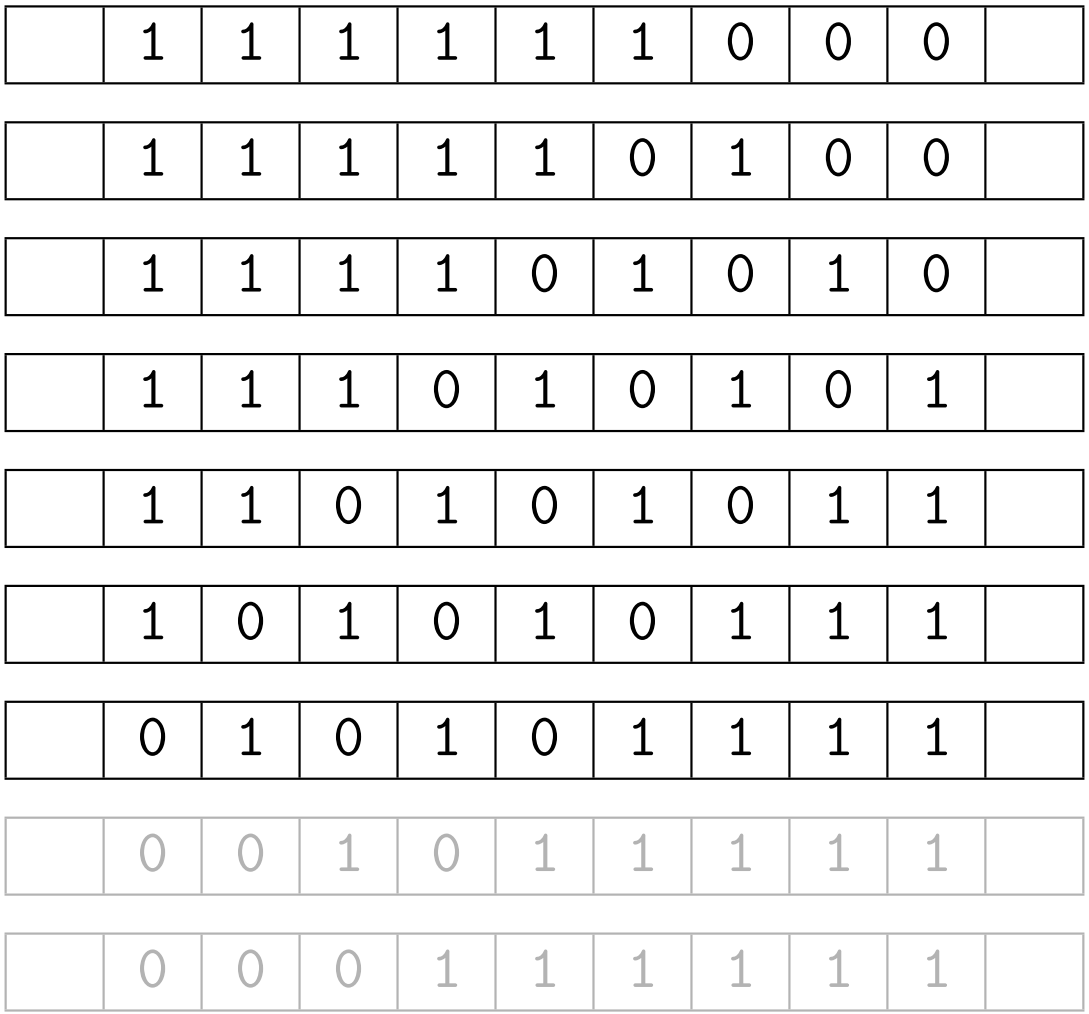
Beispiel 2


1 1 1 1 1 1 0 0 0
1 1 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0
1 1 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 1
1 0 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

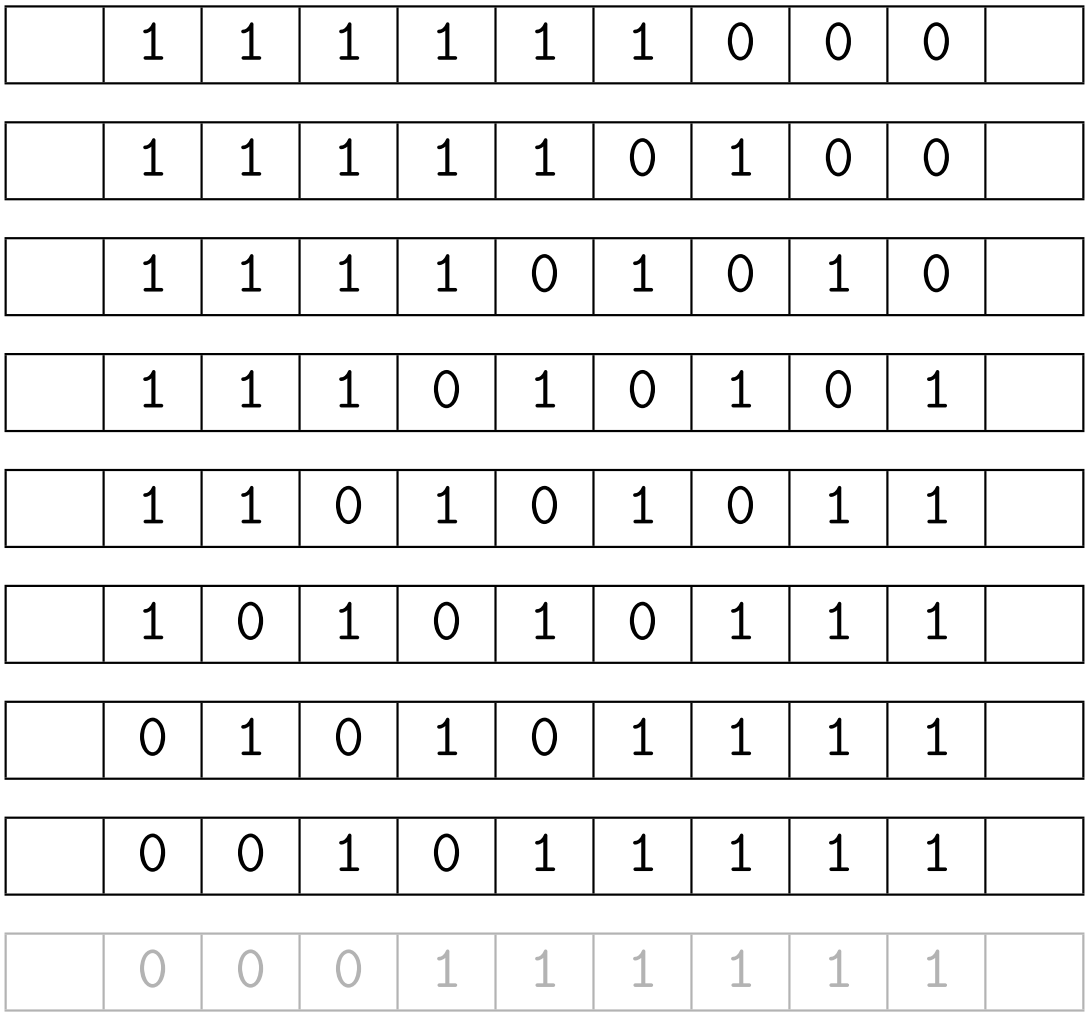
Beispiel 2


1 1 1 1 1 1 0 0 0
1 1 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0
1 1 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 1
1 0 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

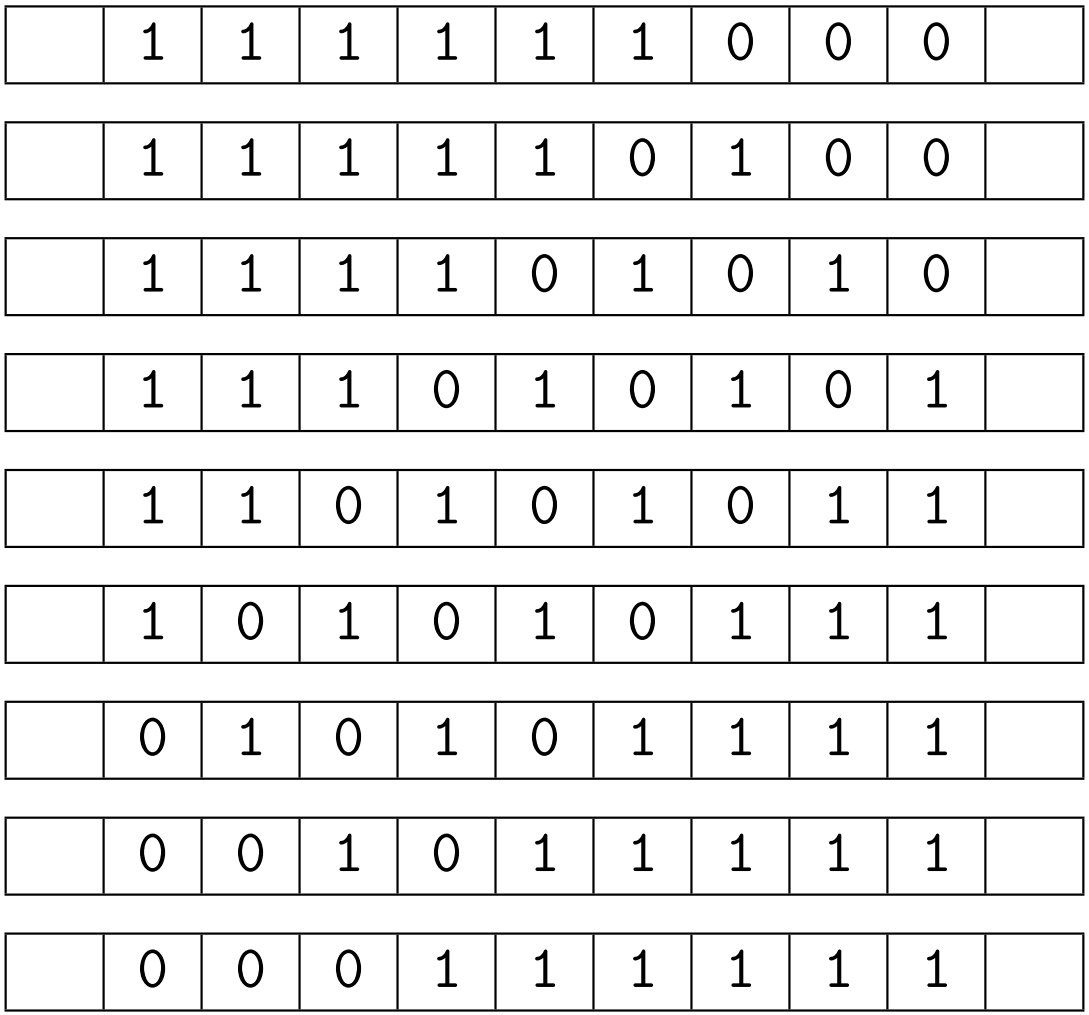
Beispiel 2


1 1 1 1 1 1 0 0 0
1 1 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0
1 1 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 1
1 0 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

Beispiel 2


1 1 1 1 1 1 0 0 0
1 1 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0
1 1 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 1
1 0 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

Beispiel 2


1 1 1 1 1 1 0 0 0
1 1 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0
1 1 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 1
1 0 1 0 1 0 1 1 1
0 1 0 1 0 1 1 1 1
0 0 1 0 1 1 1 1 1
0 0 0 1 1 1 1 1 1

Aufgabe: Sortieren von „Trits“

Gegeben: $A = \{0, 1, 2\}$ $R = \mathbb{Z}$ $N = H_1$

Gesucht: Zellularautomat mit Q und δ, so dass jedes Muster $w \in A^+$ in das Muster $0^{N_0(w)}1^{N_1(w)}2^{N_2(w)}$ überführt wird.

Wie mag das wohl gehen …

Problem: Sortieren von „Trits“

Problem:

$\ell(-1)$	$\ell(0)$	$\ell(1)$	$\delta(\ell)$
2	1	0	???

Ausweg:

Problem: Sortieren von „Trits“

Problem:

$\ell(-1)$	$\ell(0)$	$\ell(1)$	$\delta(\ell)$
2	1	0	???

Ausweg: Abwechselnd nach links und nach rechts (bzw. nach rechts und nach links) schauen.

Algorithmus (Odd-Even-Transposition-Sort)

beliebiges Eingabealphabet A

$Q = A \times \{\mathrm{L}, \mathrm{R}, \_\} \cup \{\square\}$ (identifiziere $s \in A$ mit $(s, \_)$)

δ gegeben durch die folgende Tabelle:

$\ell(-1)$	$\ell(0)$	$\ell(1)$	$\delta(\ell)$
für das eigentliche Sortieren:			
(s, R)	(t, L)	$\times$	$(\max(s,t), \mathrm{R})$
$\times$	(s, R)	(t, L)	$(\min(s,t), \mathrm{L})$
$\square$	(t, L)	$\times$	(t, R)
$\times$	(s, R)	$\square$	(s, L)
für die Initialisierung:			
$\square$	$(s, \_)$	$\times$	(s, L)
(s, L)	$(s, \_)$	$\times$	(s, L)
in allen nicht bereits spezifizierten Fällen:			
$\times$	z	$\times$	z

Beispielberechnung

	1	0	2	1	0	

	1	0	2	1	0	
	L					

	1	0	2	1	0	
	R	L				

	0	1	2	1	0	
	L	R	L			

	0	1	2	1	0	
	R	L	R	L		

	0	1	1	2	0	
	L	R	L	R	L	

	0	1	1	2	0	
	L	R	L	R	L	

	0	1	1	0	2	
	R	L	R	L	R	

	0	1	0	1	2	
	L	R	L	R	L	

	0	0	1	1	2	
	R	L	R	L	R	

	0	0	1	1	2	
	L	R	L	R	L	

	0	0	1	1	2	
	R	L	R	L	R	

Algorithmus (Odd-Even-Transposition-Sort)

Version 2

- nicht schön, LR-Muster erst aufzubauen
- Idee (P. Stumpf, 2017):
  Jede Zelle verhält sich von Anfang an
  - sowohl so, als hätte sie ein L
  - als auch so, als hätte sie ein R
- also im wesentlichen Zustandsmenge
  $Q = \ldots\ (A \times \{\mathtt{L}\}) \times (A \times \{\mathtt{R}\}) \ldots$
  - zeichne $((s, \mathtt{L}), (s', \mathtt{R}))$ einfach so: | s | s' | (*ein* Zustand!)
  - denn $Q \cong \ldots\ A \times A \ldots$

Algorithmus (Odd-Even-Transposition-Sort)

Version 2, P. Stumpf (2017)

beliebiges Eingabealphabet A

$Q = A \cup A \times A \cup \{\square\}$ bzw. s identifiziert mit (s, s)

δ gegeben durch die folgende Tabelle:

$\ell(-1)$	$\ell(0)$	$\ell(1)$	$\delta(\ell)$
für die Initialisierung:			
$\times$	s	$\times$	(s, s)
für das eigentliche Sortieren:			
(r, r')	(s, s')	(t, t')	$(\min(s', t), \max(r', s))$
$\square$	(s, s')	(t, t')	$(\min(s', t), s)$
(r, r')	(s, s')	$\square$	$(s', \max(r', s))$
in allen nicht bereits spezifizierten Fällen:			
$\times$	z	$\times$	z

Beispielberechnung

für Version 2

2	2	5	5	1	1	4	4	3	3	0	0
2	2	1	5	1	5	3	4	0	4	0	3
1	2	1	2	3	5	0	5	0	4	3	4
1	1	2	2	0	3	0	5	3	5	4	4
1	1	0	2	0	2	3	3	4	5	4	5
0	1	0	1	2	2	3	3	4	4	5	5
0	0	1	1	2	2	3	3	4	4	5	5

Lemma

Odd-Even-Transposition-Sort sortiert korrekt.

Wie beweist man das?

Lemma (0-1-Sortier-Lemma von Knuth)

Wenn ein Sortieralgorithmus ausschließlich aus Operationen der Art „Vergleich-und-Austausch-wenn-größer“ besteht und
wenn von vorneherein (unabhängig von den zu sortierenden Daten) feststeht, die Werte an welchen Positionen wann miteinander verglichen und gegebenenfalls vertauscht werden, dann gilt:

Der Algorithmus sortiert genau dann alle Eingabedatensätze,
wenn er alle Eingabedatensätze sortiert, die nur aus Nullen und Einsen bestehen.

Lemma (0-1-Sortier-Lemma von Knuth)

Wenn ein Sortieralgorithmus ausschließlich aus Operationen der Art „Vergleich-und-Austausch-wenn-größer“ besteht und
wenn von vorneherein (unabhängig von den zu sortierenden Daten) feststeht, die Werte an welchen Positionen wann miteinander verglichen und gegebenenfalls vertauscht werden, dann gilt:

Der Algorithmus sortiert genau dann alle Eingabedatensätze,
wenn er alle Eingabedatensätze sortiert, die nur aus Nullen und Einsen bestehen.

Lemma (0-1-Sortier-Lemma von Knuth)

Wenn ein Sortieralgorithmus ausschließlich aus Operationen der Art „Vergleich-und-Austausch-wenn-größer“ besteht und
wenn von vorneherein (unabhängig von den zu sortierenden Daten) feststeht, die Werte an welchen Positionen wann miteinander verglichen und gegebenenfalls vertauscht werden, dann gilt:

Der Algorithmus sortiert genau dann alle Eingabedatensätze,
wenn er alle Eingabedatensätze sortiert, die nur aus Nullen und Einsen bestehen.

Lemma (0-1-Sortier-Lemma von Knuth)

Wenn ein Sortieralgorithmus ausschließlich aus Operationen der Art „Vergleich-und-Austausch-wenn-größer“ besteht und
wenn von vorneherein (unabhängig von den zu sortierenden Daten) feststeht, die Werte an welchen Positionen wann miteinander verglichen und gegebenenfalls vertauscht werden, dann gilt:

Der Algorithmus sortiert genau dann alle Eingabedatensätze,
wenn er alle Eingabedatensätze sortiert, die nur aus Nullen und Einsen bestehen.

Beweis 1 (von Knuths Lemma)

Es sei $x_1, \ldots, x_n$ ein Eingabedatensatz, der falsch behandelt wird.
Zeige: auch ein 0-1-Eingabedatensatz $x_1^*, \ldots, x_n^*$ wird falsch behandelt.

Angenommen, richtig wäre $x_{\pi(1)} \leq x_{\pi(2)} \leq \cdots \leq x_{\pi(n)}$
aber der Algorithmus liefere $x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(n)}$

Es sei k die erste falsche Stelle, also $x_{\sigma(i)} = x_{\pi(i)}$ für $1 \leq i < k$
$x_{\sigma(k)} > x_{\pi(k)}$

Es sei r der Index mit $x_{\sigma(r)} = x_{\pi(k)}$ es ist $r > k$

Beweis 1 (von Knuths Lemma)

Es sei $x_1, \ldots, x_n$ ein Eingabedatensatz, der falsch behandelt wird.
Zeige: auch ein 0-1-Eingabedatensatz $x_1^*, \ldots, x_n^*$ wird falsch behandelt.

Angenommen, richtig wäre	$x_{\pi(1)}$	$\leq$	$x_{\pi(2)}$	$\leq$	$\cdots$	$\leq$	$x_{\pi(n)}$
aber der Algorithmus liefere	$x_{\sigma(1)}$	,	$x_{\sigma(2)}$	,	$\cdots$	,	$x_{\sigma(n)}$

Es sei k die erste falsche Stelle, also $x_{\sigma(i)} = x_{\pi(i)}$ für $1 \leq i < k$
$x_{\sigma(k)} > x_{\pi(k)}$

Es sei r der Index mit $x_{\sigma(r)} = x_{\pi(k)}$ es ist $r > k$

Beweis 1 (von Knuths Lemma)

Es sei $x_1, \ldots, x_n$ ein Eingabedatensatz, der falsch behandelt wird.
Zeige: auch ein 0-1-Eingabedatensatz $x_1^*, \ldots, x_n^*$ wird falsch behandelt.

Angenommen, richtig wäre	$x_{\pi(1)}$	$\leq$	$x_{\pi(2)}$	$\leq$	$\cdots$	$\leq$	$x_{\pi(n)}$
aber der Algorithmus liefere	$x_{\sigma(1)}$	,	$x_{\sigma(2)}$	,	$\cdots$	,	$x_{\sigma(n)}$

Es sei k die erste falsche Stelle, also $x_{\sigma(i)} = x_{\pi(i)}$ für $1 \leq i < k$

$x_{\sigma(k)} > x_{\pi(k)}$

Es sei r der Index mit $x_{\sigma(r)} = x_{\pi(k)}$ es ist $r > k$

Beweis 2

Definiere eine 0-1-Folge $x_1^*, \ldots, x_n^*$ wie folgt:

$$x_i^* = \left[x_i > x_{\pi(k)}\right] \qquad \text{d.h.} \qquad x_i^* = \begin{cases} 1 & \text{falls } x_i > x_{\pi(k)} \\ 0 & \text{falls } x_i \leq x_{\pi(k)} \end{cases}$$

Zum Beispiel: $x^*_{\sigma(k)} = \left[x_{\sigma(k)} > x_{\pi(k)}\right] = 1$

$$x^*_{\sigma(r)} = \left[x_{\sigma(r)} > x_{\pi(k)}\right] = 0$$

Beweis 2

Definiere eine 0-1-Folge $x_1^*, \ldots, x_n^*$ wie folgt:

$$x_i^* = \left[x_i > x_{\pi(k)}\right] \qquad \text{d.h.} \qquad x_i^* = \begin{cases} 1 & \text{falls } x_i > x_{\pi(k)} \\ 0 & \text{falls } x_i \leq x_{\pi(k)} \end{cases}$$

Zum Beispiel: $x^*_{\sigma(k)} = \left[x_{\sigma(k)} > x_{\pi(k)}\right] = 1$

$$x^*_{\sigma(r)} = \left[x_{\sigma(r)} > x_{\pi(k)}\right] = 0$$

Nun gilt:
$$\begin{aligned} x_i > x_j &\implies \left(x_j > x_{\pi(k)} \implies x_i > x_{\pi(k)}\right) \\ &\implies \left(x_j^* = 1 \implies x_i^* = 1\right) \\ &\implies x_i^* \geq x_j^* \end{aligned}$$

und analog $\quad x_i \leq x_j \implies x_i^* \leq x_j^*$.

Beweis 3

Also liefert der Algorithmus für die Eingabe $x_1^*, \ldots, x_n^*$
das gleiche Ergebnis, das man erhält,

wenn man an $x_1^*, \ldots, x_n^*$ die gleichen Vertauschungen vornimmt,
die der Algorithmus an $x_1, \ldots, x_n$ durchführt.

Der Algorithmus permutiert die x_i^* also gemäß σ.

Nach Beendigung des Algorithmus ergibt sich daher:

	$x^*_{\sigma(1)}$	$\cdots$	$x^*_{\sigma(k)}$	$\cdots$	$x^*_{\sigma(r)}$	$\cdots$	$x^*_{\sigma(n)}$
i.e.		$\cdots$	$[x_{\sigma(k)} > x_{\pi(k)}]$	$\cdots$	$[x_{\sigma(r)} > x_{\pi(k)}]$	$\cdots$	
i.e.		$\cdots$	1	$\cdots$	0	$\cdots$	

und das ist eine *nicht sortierte* Folge.

Beweis 3

Also liefert der Algorithmus für die Eingabe $x_1^*, \ldots, x_n^*$
das gleiche Ergebnis, das man erhält,

wenn man an $x_1^*, \ldots, x_n^*$ die gleichen Vertauschungen vornimmt,
die der Algorithmus an $x_1, \ldots, x_n$ durchführt.

Der Algorithmus permutiert die x_i^* also gemäß σ.

Nach Beendigung des Algorithmus ergibt sich daher:

	$x^*_{\sigma(1)}$	$\cdots$	$x^*_{\sigma(k)}$	$\cdots$	$x^*_{\sigma(r)}$	$\cdots$	$x^*_{\sigma(n)}$
i.e.		$\cdots$	$[x_{\sigma(k)} > x_{\pi(k)}]$	$\cdots$	$[x_{\sigma(r)} > x_{\pi(k)}]$	$\cdots$	
i.e.		$\cdots$	1	$\cdots$	0	$\cdots$	

und das ist eine *nicht sortierte* Folge.

Beweis 3

Also liefert der Algorithmus für die Eingabe $x_1^*, \ldots, x_n^*$
das gleiche Ergebnis, das man erhält,

wenn man an $x_1^*, \ldots, x_n^*$ die gleichen Vertauschungen vornimmt,
die der Algorithmus an $x_1, \ldots, x_n$ durchführt.

Der Algorithmus permutiert die x_i^* also gemäß σ.

Nach Beendigung des Algorithmus ergibt sich daher:

	$x^*_{\sigma(1)}$	$\cdots$	$x^*_{\sigma(k)}$	$\cdots$	$x^*_{\sigma(r)}$	$\cdots$	$x^*_{\sigma(n)}$
i.e.		$\cdots$	$[x_{\sigma(k)} > x_{\pi(k)}]$	$\cdots$	$[x_{\sigma(r)} > x_{\pi(k)}]$	$\cdots$	
i.e.		$\cdots$	1	$\cdots$	0	$\cdots$	

und das ist eine *nicht sortierte* Folge.

Beweis (Korrektheit von Odd-Even-Transposition-Sort)

Knuths Sortier-Lemma ist anwendbar.

Betrachte eine 0-1-Eingabefolge mit e Einsen.

Fasse den Algorithmus so auf, dass von links nach rechts R-Signale laufen:

1. Das erste Signal bewirkt, dass die in der Eingabe am weitesten rechts stehende 1 nach spätestens $2 + (n - 1) = n + 1$ Schritten am Ziel ist.
2. Das zweite Signal bewirkt, dass die in der Eingabe zweite 1 von rechts
   nach spätestens $4 + (n - 2) = n + 2$ Schritten am Ziel ist.

. . . und so weiter

e. Das e-te Signal bewirkt, dass die in der Eingabe e-te 1 von rechts nach spätestens $2e + (n - e) = n + e$ Schritten am Ziel ist.

Dann sind alle Einsen am Ziel und die 0-1-Folge ist sortiert.

Beweis (Korrektheit von Odd-Even-Transposition-Sort)

Knuths Sortier-Lemma ist anwendbar.
Betrachte eine 0-1-Eingabefolge mit e Einsen.
Fasse den Algorithmus so auf, dass von links nach rechts R-Signale laufen:

1. Das erste Signal bewirkt, dass die in der Eingabe am weitesten rechts stehende 1 nach spätestens $2 + (n - 1) = n + 1$ Schritten am Ziel ist.
2. Das zweite Signal bewirkt, dass die in der Eingabe zweite 1 von rechts
   nach spätestens $4 + (n - 2) = n + 2$ Schritten am Ziel ist.

... und so weiter

e. Das e-te Signal bewirkt, dass die in der Eingabe e-te 1 von rechts nach spätestens $2e + (n - e) = n + e$ Schritten am Ziel ist.

Dann sind alle Einsen am Ziel und die 0-1-Folge ist sortiert.

Beweis (Korrektheit von Odd-Even-Transposition-Sort)

Knuths Sortier-Lemma ist anwendbar.
Betrachte eine 0-1-Eingabefolge mit e Einsen.
Fasse den Algorithmus so auf, dass von links nach rechts R-Signale laufen:

1. Das erste Signal bewirkt, dass die in der Eingabe am weitesten rechts stehende 1 nach spätestens $2 + (n - 1) = n + 1$ Schritten am Ziel ist.
2. Das zweite Signal bewirkt, dass die in der Eingabe zweite 1 von rechts
   nach spätestens $4 + (n - 2) = n + 2$ Schritten am Ziel ist.

... und so weiter

e. Das e-te Signal bewirkt, dass die in der Eingabe e-te 1 von rechts nach spätestens $2e + (n - e) = n + e$ Schritten am Ziel ist.

Dann sind alle Einsen am Ziel und die 0-1-Folge ist sortiert.

Beweis (Korrektheit von Odd-Even-Transposition-Sort)

Knuths Sortier-Lemma ist anwendbar.
Betrachte eine 0-1-Eingabefolge mit e Einsen.
Fasse den Algorithmus so auf, dass von links nach rechts R-Signale laufen:

1. Das erste Signal bewirkt, dass die in der Eingabe am weitesten rechts stehende 1 nach spätestens $2 + (n - 1) = n + 1$ Schritten am Ziel ist.
2. Das zweite Signal bewirkt, dass die in der Eingabe zweite 1 von rechts
   nach spätestens $4 + (n - 2) = n + 2$ Schritten am Ziel ist.

. . . und so weiter

e. Das e-te Signal bewirkt, dass die in der Eingabe e-te 1 von rechts nach spätestens $2e + (n - e) = n + e$ Schritten am Ziel ist.

Dann sind alle Einsen am Ziel und die 0-1-Folge ist sortiert.

Beweis (Korrektheit von Odd-Even-Transposition-Sort)

Knuths Sortier-Lemma ist anwendbar.
Betrachte eine 0-1-Eingabefolge mit e Einsen.
Fasse den Algorithmus so auf, dass von links nach rechts R-Signale laufen:

1. Das erste Signal bewirkt, dass die in der Eingabe am weitesten rechts stehende 1 nach spätestens $2 + (n - 1) = n + 1$ Schritten am Ziel ist.
2. Das zweite Signal bewirkt, dass die in der Eingabe zweite 1 von rechts
   nach spätestens $4 + (n - 2) = n + 2$ Schritten am Ziel ist.

... und so weiter

e. Das e-te Signal bewirkt, dass die in der Eingabe e-te 1 von rechts nach spätestens $2e + (n - e) = n + e$ Schritten am Ziel ist.

Dann sind alle Einsen am Ziel und die 0-1-Folge ist sortiert.

Beweis (Korrektheit von Odd-Even-Transposition-Sort)

Knuths Sortier-Lemma ist anwendbar.
Betrachte eine 0-1-Eingabefolge mit e Einsen.
Fasse den Algorithmus so auf, dass von links nach rechts R-Signale laufen:

1. Das erste Signal bewirkt, dass die in der Eingabe am weitesten rechts stehende 1 nach spätestens $2 + (n - 1) = n + 1$ Schritten am Ziel ist.
2. Das zweite Signal bewirkt, dass die in der Eingabe zweite 1 von rechts
   nach spätestens $4 + (n - 2) = n + 2$ Schritten am Ziel ist.

... und so weiter

e. Das e-te Signal bewirkt, dass die in der Eingabe e-te 1 von rechts nach spätestens $2e + (n - e) = n + e$ Schritten am Ziel ist.

Dann sind alle Einsen am Ziel und die 0-1-Folge ist sortiert.

Beweis (Korrektheit von Odd-Even-Transposition-Sort)

Knuths Sortier-Lemma ist anwendbar.
Betrachte eine 0-1-Eingabefolge mit e Einsen.
Fasse den Algorithmus so auf, dass von links nach rechts R-Signale laufen:

1. Das erste Signal bewirkt, dass die in der Eingabe am weitesten rechts stehende 1 nach spätestens $2 + (n - 1) = n + 1$ Schritten am Ziel ist.
2. Das zweite Signal bewirkt, dass die in der Eingabe zweite 1 von rechts
   nach spätestens $4 + (n - 2) = n + 2$ Schritten am Ziel ist.

... und so weiter

e. Das e-te Signal bewirkt, dass die in der Eingabe e-te 1 von rechts nach spätestens $2e + (n - e) = n + e$ Schritten am Ziel ist.

Dann sind alle Einsen am Ziel und die 0-1-Folge ist sortiert.

Verallgemeinerte Sortieraufgaben

- höherdimensionale Eingaben: siehe späteres Kapitel
- Sortieren von Zahlen, deren Bits in benachbarten Zellen bespeichert sind
  - Sortieren gleich langer Dualzahlen (siehe späteres Kapitel), zum Beispiel:

    #0011#0101#0110#1011#0010#0011#1001#0000#

    wird sortiert zu

    #0000#0010#0011#0011#0101#0110#1001#1011#
  - Sortieren unterschiedlich langer Dualzahlen, zum Beispiel:

    #11#101#110#1011#10#11#1001#0#

    wird sortiert zu

    #0#10#11#11#101#110#1001#1011#

Verallgemeinerte Sortieraufgaben

- höherdimensionale Eingaben: siehe späteres Kapitel
- Sortieren von Zahlen, deren Bits in benachbarten Zellen bespeichert sind
  - Sortieren gleich langer Dualzahlen (siehe späteres Kapitel), zum Beispiel:

    #0011#0101#0110#1011#0010#0011#1001#0000#

    wird sortiert zu

    #0000#0010#0011#0011#0101#0110#1001#1011#
  - Sortieren unterschiedlich langer Dualzahlen, zum Beispiel:

    #11#101#110#1011#10#11#1001#0#

    wird sortiert zu

    #0#10#11#11#101#110#1001#1011#

Verallgemeinerte Sortieraufgaben

- höherdimensionale Eingaben: siehe späteres Kapitel
- Sortieren von Zahlen, deren Bits in benachbarten Zellen bespeichert sind
  - Sortieren gleich langer Dualzahlen (siehe späteres Kapitel), zum Beispiel:

    #0011#0101#0110#1011#0010#0011#1001#0000#

    wird sortiert zu

    #0000#0010#0011#0011#0101#0110#1001#1011#
  - Sortieren unterschiedlich langer Dualzahlen, zum Beispiel:

    #11#101#110#1011#10#11#1001#0#

    wird sortiert zu

    #0#10#11#11#101#110#1001#1011#

Verallgemeinerte Sortieraufgaben

- höherdimensionale Eingaben: siehe späteres Kapitel
- Sortieren von Zahlen, deren Bits in benachbarten Zellen bespeichert sind
  - Sortieren gleich langer Dualzahlen (siehe späteres Kapitel), zum Beispiel:

    #0011#0101#0110#1011#0010#0011#1001#0000#

    wird sortiert zu

    #0000#0010#0011#0011#0101#0110#1001#1011#
  - Sortieren unterschiedlich langer Dualzahlen, zum Beispiel:

    #11#101#110#1011#10#11#1001#0#

    wird sortiert zu

    #0#10#11#11#101#110#1001#1011#

Zusammenfassung

- Knuths 0-1-Sortierlemma sichert zu, dass bei gewissen Sortieralgorithmen für den Nachweis der Korrektheit die Betrachtung von 0-1-Eingaben genügt.
- Sortieren in eindimensionalen Zellularautomaten mit einem Datum je Zelle ist in Linearzeit möglich.

Zusammenfassung

- Knuths 0-1-Sortierlemma sichert zu, dass bei gewissen Sortieralgorithmen für den Nachweis der Korrektheit die Betrachtung von 0-1-Eingaben genügt.
- Sortieren in eindimensionalen Zellularautomaten mit einem Datum je Zelle ist in Linearzeit möglich.