1 Sortieren in zweidimensionalen Zellularautomaten

Der Einfachheit halber beschränken wir uns in diesem Kapitel auf das Sortieren *quadratischer* Muster. Mit n bezeichnen wir immer die Anzahl zu sortierender Elemente und mit $m = \sqrt{n}$ die Seitenlänge des Quadrates.

Die erste Frage, die man beantworten muss, wenn man sich mit zweidimensionalem Sortieren beschäftigt, ist, welche Anordnungen von Werten man denn als sortiert ansehen will. Man könnte zum Beispiel verlangen, dass jede Zeile für sich und jede Spalte für sich sortiert sein müssen, wie dies in der nachfolgenden Abbildung 1.1(a) angedeutet ist. Man kann dies dahingehend verschärfen, dass alle Zeilen hintereinander gesetzt eine lange sortierte Zeile ergeben müssen (Abbildung 1.1(b)). Dieser Ansatz erscheint nicht unplausibel, hat aber im Hinblick auf Zellularautomaten einen Nachteil. Elemente, die im Sinne der Sortierung aufeinanderfolgend sind, befinden sich nicht unbedingt in benachbarten Zellen; das ist zum Beispiel beim letzten Element einer Zeile und dem ersten Element der nächsten Zeile so. Vergleiche zwischen diesen Elementen würden recht zeitaufwendig werden. Eine der „bevorzugten" Reihenfolgen, die man mit Algorithmen für das zweidimensionale Sortieren auf Gittern (sei es nun bei Zellularautomaten oder auf Parallelrechnern) anstrebt, ist eine sortierte Anordnung der Elemente entlang einer Schlangenlinie (Abbildung 1.1(c)). Das soll auch das Ziel dieses Kapitels sein. Allerdings gibt es auch Autoren (zum Beispiel Kunde und Gürtzig 1997), die noch für noch andere Sortierreihenfolgen Algorithmen entwerfen.

1	2	6	7	15
3	5	8	14	16
4	9	13	17	22
10	12	18	21	23
11	19	20	24	25

(a)

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

(b)

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16
21	22	23	24	25

(c)

Abbildung 1.1: Verschiedene zweidimensionale Sortierungen.

1.1 Problem. (zweidimensionales Sortieren) Es sei A eine endliche und total geordnete Menge. Gesucht ist ein zweidimensionaler Zellularautomat, der jedes Muster $e : \mathbb{N}_m \times \mathbb{N}_m \to A$ in das *sortierte Muster* $s : \mathbb{N}_m \times \mathbb{N}_m \to A$ überführt mit den Eigenschaften:

1. Für alle $a \in A$ ist $N_a(e) = N_a(s)$.
2. Falls $i \in \mathbb{N}_m$ ungerade ist, gilt für alle $j \in \mathbb{N}_{m-1}$: $s(i,j) \leqslant s(i,j+1)$.
3. Falls $i \in \mathbb{N}_m$ gerade ist, gilt für alle $j \in \mathbb{N}_{m-1}$: $s(i,j) \geqslant s(i,j+1)$.
4. Falls $i \in \mathbb{N}_{m-1}$ ungerade ist, gilt $s(i,m) \leqslant s(i+1,m)$.
5. Falls $i \in \mathbb{N}_{m-1}$ gerade ist, gilt $s(i,1) \leqslant s(i+1,1)$.

Wir beginnen mit einem Algorithmus, der noch nicht zeitoptimal ist, aber später in einem solchen benutzt wird.

1.2 Algorithmus. (SHEARSORT) Der Algorithmus läuft in $1 + 2\lceil \log m \rceil$ Phasen ab:

In den Phasen $1, 3, 5, \ldots, 1 + 2\lceil \log m \rceil$ wird jede *Zeile* für sich sortiert (zum Beispiel mit dem Odd-even-Transposition-Sort aus Algorithmus 5.6), und zwar ungerade Zeilen nach rechts aufsteigend und gerade Zeilen nach links aufsteigend.

In den Phasen $2, 4, 6, \ldots, 2\lceil \log m \rceil$ wird jede *Spalte* für sich sortiert, und zwar kleine Werte nach oben. Abbildung 1.2 zeigt für eine Beispieleingabe die Konfigurationen nach jeweils einer Phase.

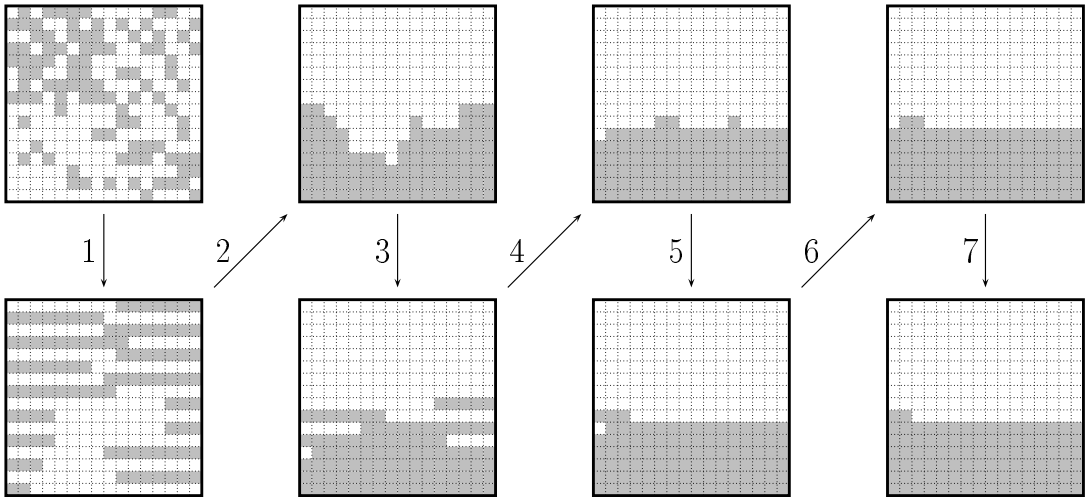

Abbildung 1.2: Eine Beispielrechnung für den Shearsortalgorithmus (7 Phasen).

Will man diesen Algorithmus in einem Zellularautomaten verwenden, so muss man zunächst in einer vorbereitenden Phase 0 dafür sorgen, dass allen Zellen die Information zur Verfügung steht, ob sie sich in einer geraden oder in einer ungeraden Zeile befinden, damit sie dementsprechend in die richtige Richtung aufsteigend sortieren können. Diese Initialisierung kann zum Beispiel von den Zellen in der obersten Zeile für ihre jeweilige Spalte angestoßen werden und dauert (etwa) m Schritte. Gleichzeitig kann in allen 4 Ecken ein FSSP-Algorithmus gestartet werden, der nach der gleichen Zeit endet und überall das eigentliche Sortieren startet.

Um das strenge Abwechseln von Zeilen- und Spaltensortieren zu erreichen, können ebenfalls FSSP-Algorithmen verwendet werden, die nach jeweils m Schritten eine Umschaltung bewirken.

1.3 Lemma. Algorithmus 1.2 leistet das Gewünschte und benötigt $\Theta(\sqrt{n} \log n)$ Schritte.

1.4 Es ist wichtig, sich darüber klar zu werden, dass hier etwas nicht vollkommen Selbstverständliches passiert. Ändert man SHEARSORT dahingehend ab, dass in allen Zeilen in die gleiche Richtung aufsteigend sortiert wird, so erhält man *nicht* einen Algorithmus, der wie in Abbildung 1.1(b) dargestellt sortiert, sondern nur einen, der Sortierungen wie in Abbildung 1.1(a) erzeugt!

1.5 Beweis (von Lemma 1.3) Das 0-1-Sortierlemma ist anwendbar.

Wir betrachten eine Konfiguration nach einer positiven geraden Anzahl von Phasen. Sie hat qualitativ folgendes Aussehen: Ganz oben befinden sich einige reine 0-Zeilen und ganz unten

einige reine 1-Zeilen. Die Zeilen dazwischen heißen gemischte Zeilen, da in ihnen sowohl 0- als auch 1-Elemente vorkommen.

Wir zeigen nun: Durch zwei aufeinander folgende Phasen wird die Anzahl der gemischten Zeilen mindestens halbiert.

Man betrachte dazu zwei aufeinanderfolgende Zeilen, von denen mindestens eine gemischt ist. Nach einer Zeilensortierphase ist eine der drei folgenden Situationen eingetreten (oder eine der drei gespiegelten):

$$\begin{array}{c|c|c} 0\cdots01\cdots11\cdots1 & 0\cdots01\cdots1 & 0\cdots00\cdots01\cdots1 \\ 1\cdots11\cdots10\cdots0 & 1\cdots10\cdots0 & 1\cdots10\cdots00\cdots0 \end{array}$$

Anschließend erfolgt eine Spaltensortierung: Wenn sie so vorgenommen wird, dass zunächst in jedem Zeilenpaar getrennt sortiert wird, dann entsteht offensichtlich immer mindestens eine reine Zeile:

$$\begin{array}{c|c|c} 0\cdots01\cdots10\cdots0 & 0\cdots00\cdots0 & 0\cdots00\cdots00\cdots0 \\ 1\cdots11\cdots11\cdots1 & 1\cdots11\cdots1 & 1\cdots10\cdots01\cdots1 \end{array}$$

Werden dann reine Zeilen ganz nach oben bzw. unten getauscht und schließlich irgendwie sortiert, so sinkt die Anzahl x der gemischten Zeilen um mindestens $\frac{x-1}{2}$.

Wird die Spaltensortierung anders vorgenommen, ist das Ergebnis aber natürlich trotzdem immer noch das gleiche. Also gilt sogar *immer*, dass die Anzahl x der gemischten Zeilen um mindestens $\frac{x-1}{2}$ sinkt.

Am Anfang liegen maximal m gemischte Zeilen vor. Folglich bleibt nach $\lceil \log m \rceil$ Zeilen- und $\lceil \log m \rceil$ Spaltenphasen höchstens eine gemischte Zeile übrig, die in der letzten Zeilenphase sortiert wird. Insgesamt genügen also $m + m(1 + 2\lceil \log m \rceil) \in \Theta(\sqrt{n} \log n)$ Schritte für die vollständige Sortierung. ∎

1.6 Übung. Man präzisiere im vorangegangenen Algorithmus die folgenden Aspekte so weit, dass offensichtlich ist, wie sie in einem Zellularautomaten realisiert werden können:

1. das regelmäßige Abwechseln zwischen Zeilen- und Spaltenphasen;
2. die Durchführung einer Zeilen- bzw. einer Spaltenphase;
3. das Erkennen des Endes des Sortiervorganges;

Wir kommen nun zu einem Algorithmus, dessen Laufzeit bis auf einen (kleinen!) konstanten Faktor optimal ist.

1.7 Algorithmus. (von Schnorr und Shamir) Im folgenden sei die Anzahl der zu sortierenden Elemente $n = k^8$ eine achte Potenz und sie seien wieder in einem quadratischen Muster angeordnet. Der Algorithmus von Schnorr und Shamir läuft in neun Phasen ab:

Phase 0: Teile das Muster in $k \times k$ *Blöcke* der Größe $k^3 \times k^3$.

Für die weitere Beschreibung des Algorithmus und für den später zu gebenden Beweis 1.9 seiner Korrektheit vereinbaren wir folgende Sprechweisen: k waagerecht nebeneinander angeordnete Blöcke heißen eine *Zeile von Blöcken* und k senkrecht übereinander angeordnete Blöcke heißen eine *Spalte von Blöcken*. k^4 nebeneinander liegende Einzelelemente werden als eine *volle Zeile* bezeichnet und k^4 übereinander angeordnete Einzelelemente als eine *volle Spalte*.

Phase 1: Sortiere jeden Block für sich in Schlangenlinienform.

Hier und bei den noch folgenden Sortierungen in Schlangenlinienform werde stets SHEARSORT verwendet.

Phase 2: Verteile die vollen Spalten jeder Spalte von Blöcken gleichmäßig auf alle Spalten von Blöcken.

Phase 3: Sortiere jeden Block für sich in Schlangenlinienform.

Phase 4: Sortiere jede volle Spalte (kleine Werte nach oben).

Phase 5: Sortiere in jeder Spalte von Blöcken die Blöcke $1+2$, $3+4$, ... jeweils gemeinsam in Schlangenlinienform.

Phase 6: Sortiere in jeder Spalte von Blöcken die Blöcke $2+3$, $4+5$, ... jeweils gemeinsam in Schlangenlinienform.

Phase 7: Sortiere jede volle Zeile (in ungeraden kleine Werte nach links, in geraden nach rechts)

Phase 8: Führe auf der „gesamten Schlangenlinie“ $2k^3$ „Odd-Even-Transposition-Sort-Schritte“ durch.

In Abbildung 1.3 sind für den einzig darstellbaren, nicht trivialen Fall $k = 2$ für ein Beispiel die Konfigurationen nach den einzelnen Phasen dargestellt.

Zum besseren Verständnis der Phase 2 ist in Abbildung 1.4 für den Fall $k = 3$ (*nicht* $k = 2$ wie oben) eine volle Zeile von Blöcken unmittelbar vor und nach ihrer Durchführung dargestellt. Die Farben machen kenntlich, von wo nach wo Spalten bewegt werden. Die detaillierte Implementierung der Spaltenpermutationen in Zeit $\Theta(\sqrt{n})$ in einem Zellularautomaten überlassen wir als Übung. Man überlegt sich zunächst, wie man die Spalten aus nur einer vollen Spalte von Blöcken an die richtigen Stellen transportiert.

1.8 Lemma. Algorithmus 1.7 sortiert quadratische Muster in $\Theta(\sqrt{n})$ Schritten.

1.9 Beweis Zum Nachweis der Korrektheit ist das 0-1-Sortierlemma anwendbar. Wir sprechen wieder von reinen und gemischten Zeilen (sowohl innerhalb eines Blockes als auch in einer ganzen Blockzeile). Es gilt nach

Phase 1: In jedem Block gibt es höchstens eine gemischte Zeile, d.h. zwei Spalten des gleichen Blockes unterscheiden sich um höchstens eine 1.

Phase 2: Da jeder Block aus jedem anderen (der gleichen Zeile von Blöcken) mindestens eine Spalte bekommen hat, unterscheiden sich zwei Blöcke der gleichen Zeile von Blöcken in der Anzahl Einsen um höchstens k.

Phase 3: Da $k < k^3$ ist, gibt es in jeder Zeile von Blöcken höchstens zwei gemischte volle Zeilen.

Phase 4: In jeder Spalte von Blöcken gibt es höchstens k gemischte Zeilen.

Phasen 5+6: Jede Spalte von Blöcken ist insgesamt in Schlangenlinienform. Da in den Phasen 3–6 kein Austausch zwischen verschiedenen Spalten von Blöcken vorgenommen wurde, gilt wie nach Phase 2 immer noch: Je zwei ganze Spalten von Blöcken unterscheiden sich in der Anzahl Einsen um höchstens k^2. Wegen $k^2 < k^3$ gibt es folglich nach Phase 6 insgesamt nur noch höchstens zwei gemischte volle Zeilen.

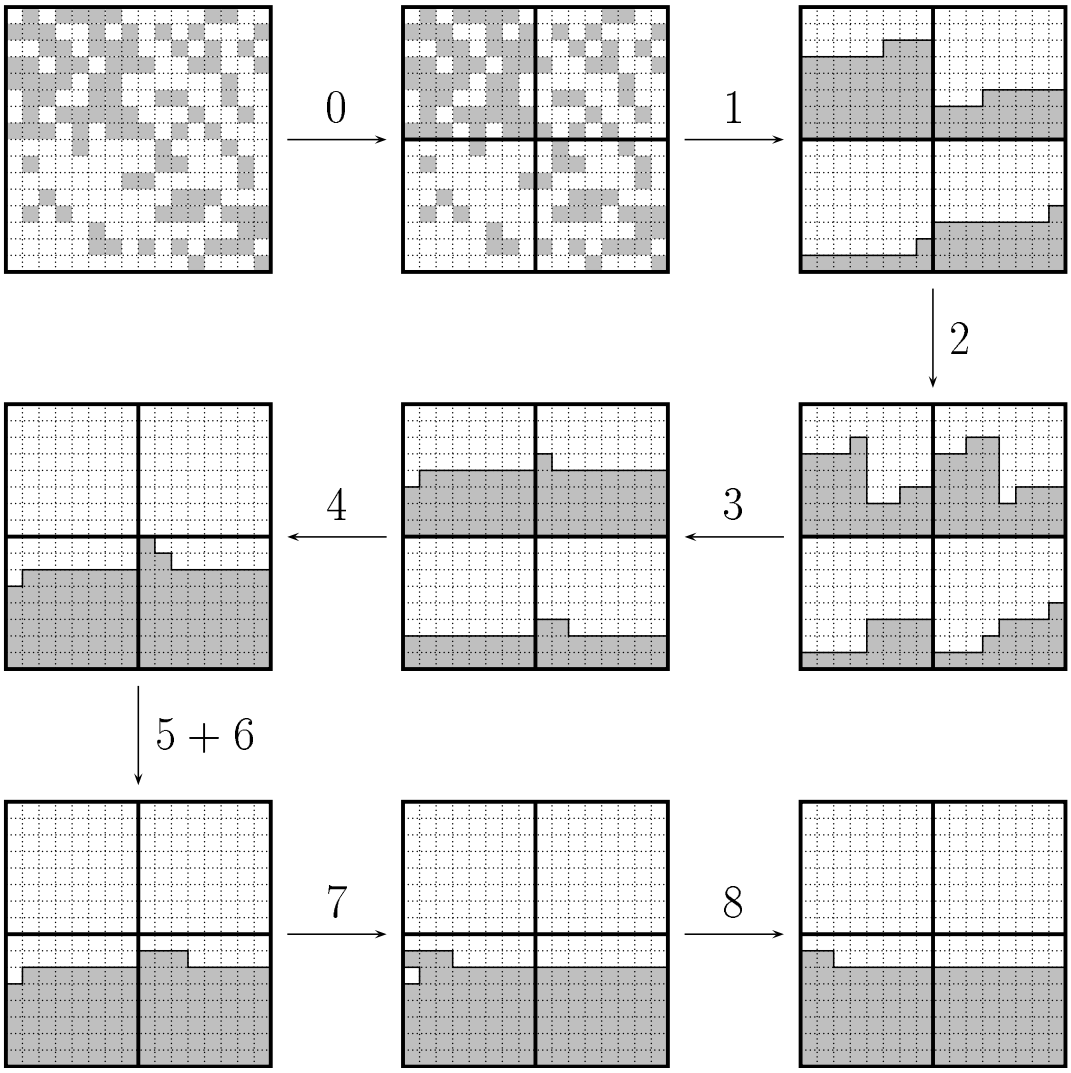

Abbildung 1.3: Die Phasen des Sortier-Algorithmus von Schnorr und Shamir.

Phase 7: Es gibt insgesamt nur noch höchstens zwei gemischte volle Zeilen: die eine enthält lauter Nullen und am „höheren“ Ende höchstens $k^2 \cdot k = k^3$ Einsen, die andere lauter Einsen und am „niedrigeren“ Ende höchstens $k^2 \cdot k = k^3$ Nullen. Es bedarf also höchstens noch $k^3 + k^3$ Sortierschritten entlang der Gesamtschlangenlinie um den Rest zu sortieren. Daher gilt nach

Phase 8: Alles ist sortiert.

Führt man wie gefordert alle Sortierungen mit `SHEARSORT` durch, so ergibt die Abschätzung des Zeitbedarfes für die einzelnen Phasen nach oben

$$k^4 + k^3 \cdot \log k^3 + k^4 + k^3 \cdot \log k^3 + k^4 + k^3 \cdot \log k^3 + k^3 \cdot \log k^3 + k^4 + k^3 \in \Theta(\sqrt{n}).$$

■

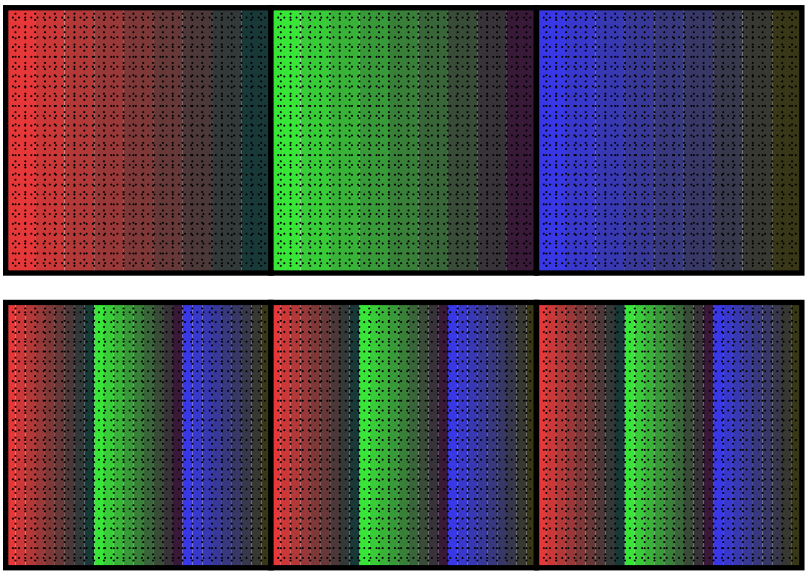

Abbildung 1.4: Umsortierung der Spalten während der Phase 2. Oben eine volle Zeile von Blöcken vor und unten nach der Permutation.

1.10 Man mache sich rückblickend klar, wie überaus hilfreich es für den Beweis der Korrektheit des Algorithmus war, dass man Knuths 0-1-Sortierlemma benutzen konnte!

In der Zeitabschätzung für den Algorithmus von Schnorr und Shamir gibt es 4 Summanden der Form k^4. Betrachtet man nicht Zellularautomaten, sondern Parallelrechner mit gitterförmigem Kommunikationsnetzwerk, bei denen die Phase 0 praktisch entfallen kann, weil jeder Prozessor seine Lage zu Blockgrenzen durch einfache Arithmetik aus seiner Prozessornummer errechnen kann, dann ergibt sich ein Zeitbedarf von $3\sqrt{n} + o(\sqrt{n})$. Man kann für diesen Fall sogar zeigen, dass die Konstante 3 optimal ist. Dies wollen wir im folgenden noch tun (siehe auch Leighton 1992).

1.11 Lemma. Jeder Algorithmus für das Zweidimensionale Sortieren in Schlangenlinienform benötigt mindestens $3\sqrt{n} - o(\sqrt{n})$ Schritte, sofern man beliebig viele verschiedene Werte sortieren können will.

1.12 Beweis Man betrachte eine anfängliche Werteverteilung wie in Abbildung 1.5 dargestellt: in dem oberen linken Dreieck mit Seitenlänge $2\sqrt[4]{n}$ werden nur die Werte 0 und n verwendet (wie, wird weiter unten diskutiert), und an den verbleibenden Stellen wird jeder der Werte 1 bis $n - 2\sqrt{n}$ genau einmal benutzt.

Der (nur einmal vorhandene) Wert x, der sich nach $2\sqrt{n} - 2\sqrt[4]{n} - 3$ Schritten im rechten unteren Eck des gesamten Quadrates befindet, kann noch nicht von einem der Anfangswerte im grauen Dreieck beeinflusst worden sein. Es wird sich dort also immer der gleiche Wert befinden, unabhängig davon, welche Werte anfangs im grauen Dreieck gespeichert sind.

Es sei $C(m)$ die Nummer der Spalte, in der sich x am Ende der Sortierung befinden wird, wenn im grauen Dreieck zu Beginn genau m mal der Wert n vorkommt. Jede Erhöhung von m um 1 bewirkt, dass sich auch die Spaltennummer um 1 ändert, da $0 < x < n$ ist. Ob sich die Spaltennummer erhöht oder erniedrigt hängt davon ab, ob in der betreffenden Zeile absteigend oder aufsteigend sortiert wird.

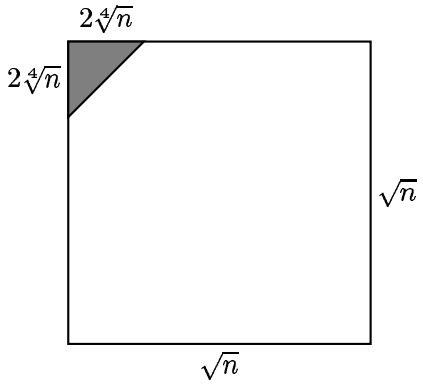

Abbildung 1.5:

Da das graue Dreieck die Größe $2\sqrt{n}$ hat, gibt es ein m', so dass $C(m') = 1$. Also muss in diesem Fall der Wert x noch von der letzten in die erste Spalte wandern. Dafür werden noch weitere $\sqrt{n} - 1$ Schritte benötigt, so dass sich ein Zeitbedarf von mindestens $3\sqrt{n} - 2\sqrt[4]{n} - 4$ ergibt. ■

Zusammenfassung

- In eindimensionalen Zellularautomaten kann man n Datenelemente in Zeit $O(n)$ sortieren.
- Im Zweidimensionalen ist die Schlangenlinie eine sinnvolle Sortierreihenfolge, weil in der Sortierung benachbarte Datenelemente am Ende auch in benachbarten Zellen liegen sollen.
- Sortieren von $\sqrt{n} \times \sqrt{n}$ Elementen in Schlangenlinienform ist in Zeit $O(\sqrt{n})$ möglich.
- Dabei ist $3\sqrt{n} - o(\sqrt{n})$ untere Zeitschranke, sofern man beliebig viele verschiedene Werte sortieren können will.

Literatur

Kunde, Manfred und Kay Gürtzig (1997). „Efficient Sorting and Routing on Reconfigurable Meshes Using Restricted Bus Length“. In: *IPPS: 11th International Parallel Processing Symposium*. IEEE Computer Society Press.

Leighton, Frank Thomson (1992). *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. San Mateo, CA 94403: Morgan Kaufmann Publ. ISBN: 1-55860-117-1.

