Praktikum (10 ECTS)

Invasive Algorithmen und Architekturen

Rolf Wanka, Jürgen Teich

Universität Erlangen-Nürnberg

http://www12.cs.fau.de/people/{rwanka|teich}/

{rwanka|teich}@cs.fau.de

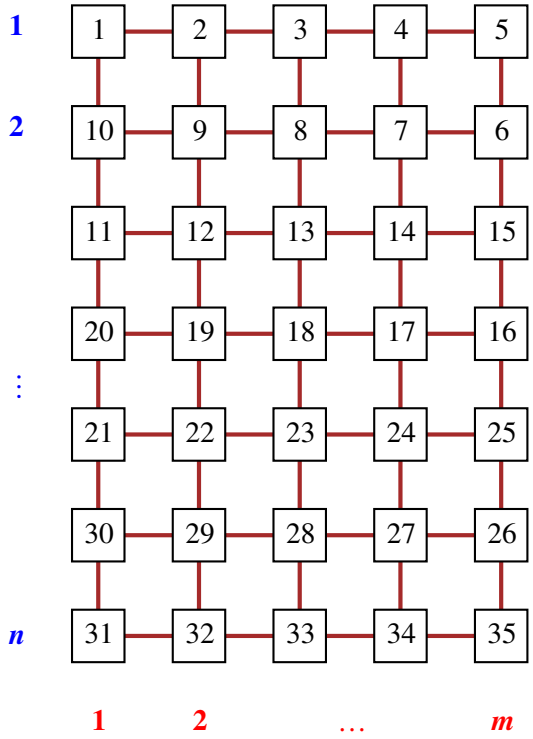
Überblick

1. Shearsort – Nicht-invasiv

2. Shearsort – Als invasives Verfahren

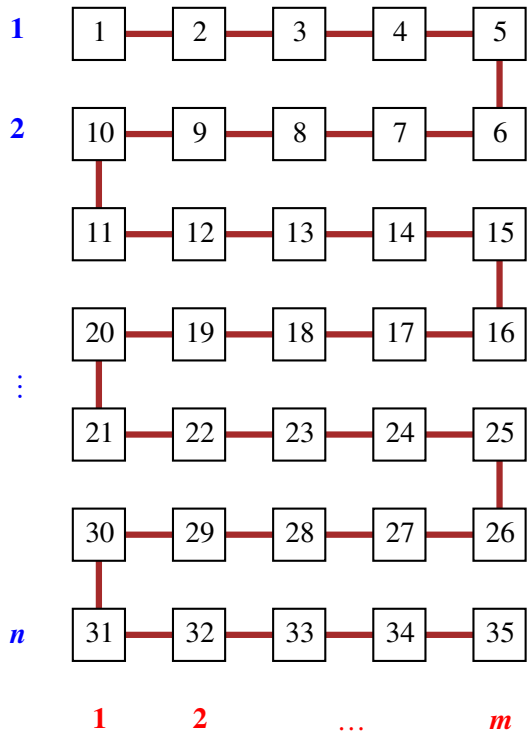
Paralleles Sortieren: ShearSort auf dem Gitter

Sortieren auf dem $n \times m$-Gitter: ShearSort, statisch funktioniert für beliebige n und m.



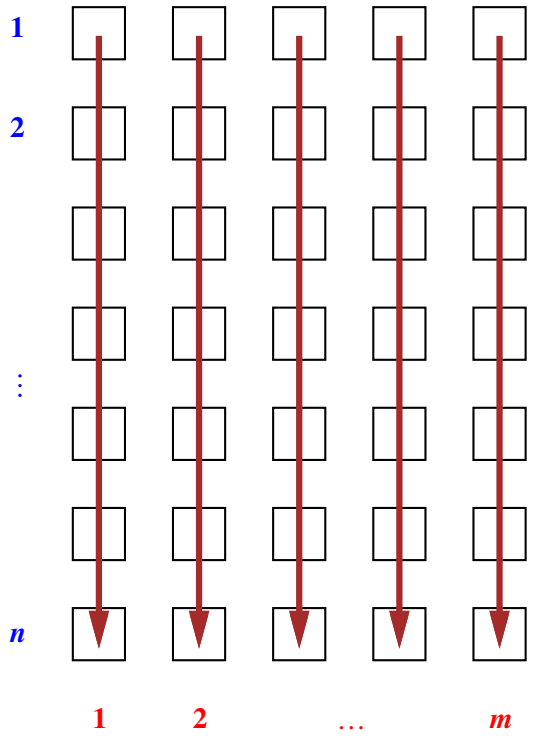
ShearSort: Numerierung

Sortieren auf dem $n \times m$-Gitter: ShearSort, Snake-like Numerierung



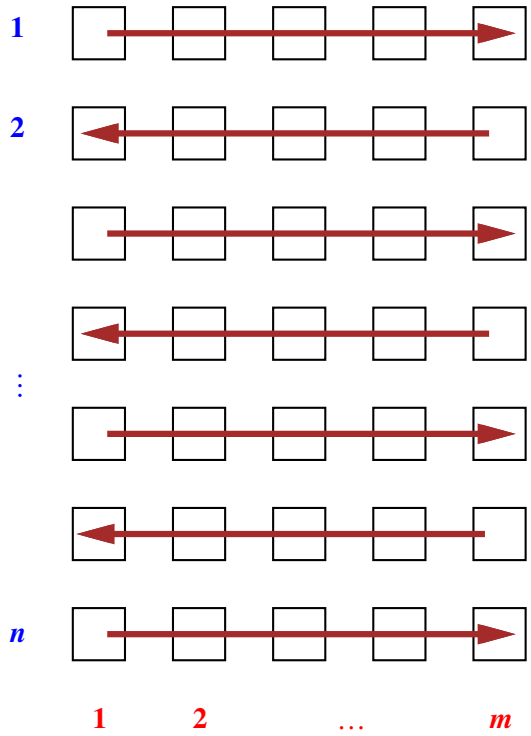
ShearSort: Arbeitsweise

Sortieren auf dem $n \times m$-Gitter: ShearSort



ShearSort: Arbeitsweise

Sortieren auf dem $n \times m$-Gitter: ShearSort



ShearSort: Allgemeine Laufzeit

Satz: [Scherson/Sen/Shamir]
ShearSort auf n Zeilen und m Spalten: im schlimmsten Fall

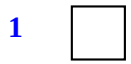
$$\lceil \log_2 n \rceil + 1$$

Runden.

Je Runde braucht man $n + m$ Schritte, wenn die Zeilen und Spalten lineare Arrays sind.

Invasion

Shearsort invasiv: $N = 63$ Schlüssel, Wunsch: 7×9-Gitter



Invasion

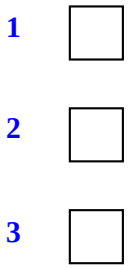
Shearsort invasiv: $N = 63$ **Schlüssel, Wunsch:** 7×9**-Gitter**

1 

2 

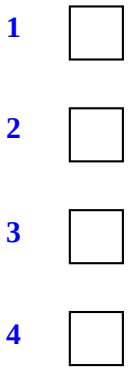
Invasion

Shearsort invasiv: $N = 63$ **Schlüssel, Wunsch:** 7×9**-Gitter**



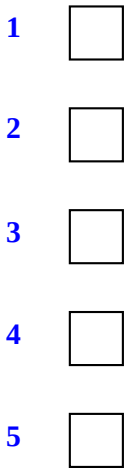
Invasion

Shearsort invasiv: $N = 63$ Schlüssel, Wunsch: 7×9-Gitter



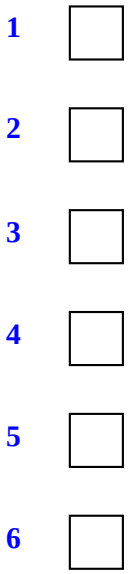
Invasion

Shearsort invasiv: $N = 63$ Schlüssel, Wunsch: 7×9-Gitter



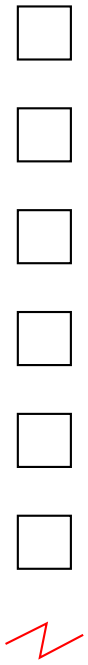
Invasion

Shearsort invasiv: $N = 63$ Schlüssel, Wunsch: 7×9-Gitter



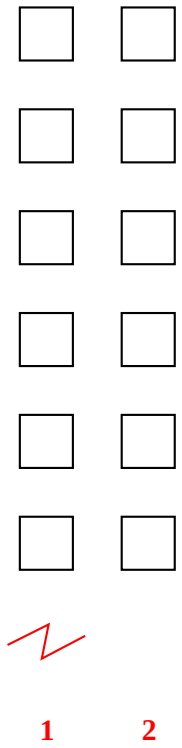
Invasion

Shearsort invasiv: $N = 63$ **Schlüssel, Wunsch:** 7×9**-Gitter**



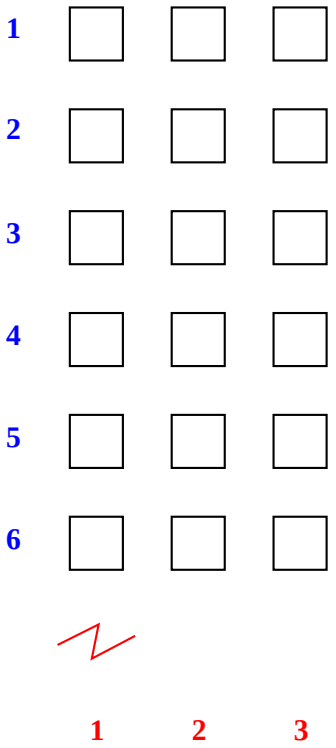
Invasion

Shearsort invasiv: $N = 63$ **Schlüssel, Wunsch:** 7×9**-Gitter**



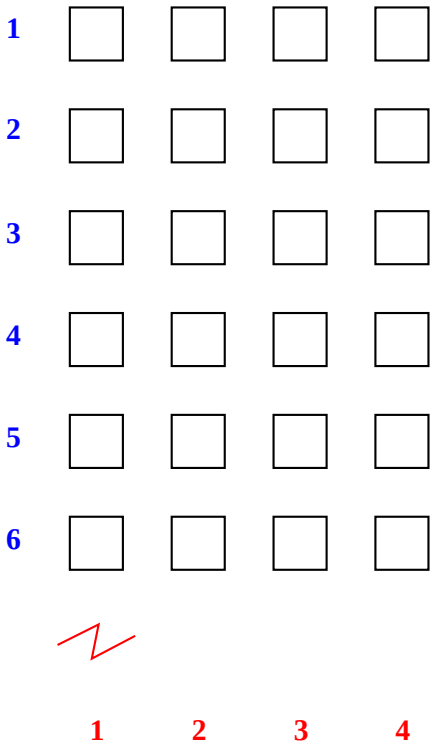
Invasion

Shearsort invasiv: $N = 63$ **Schlüssel, Wunsch:** 7×9**-Gitter**



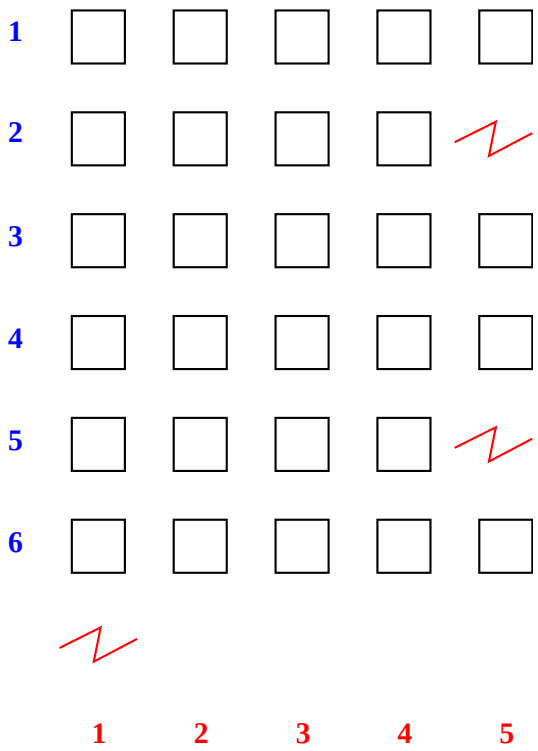
Invasion

Shearsort invasiv: $N = 63$ **Schlüssel, Wunsch:** 7×9**-Gitter**



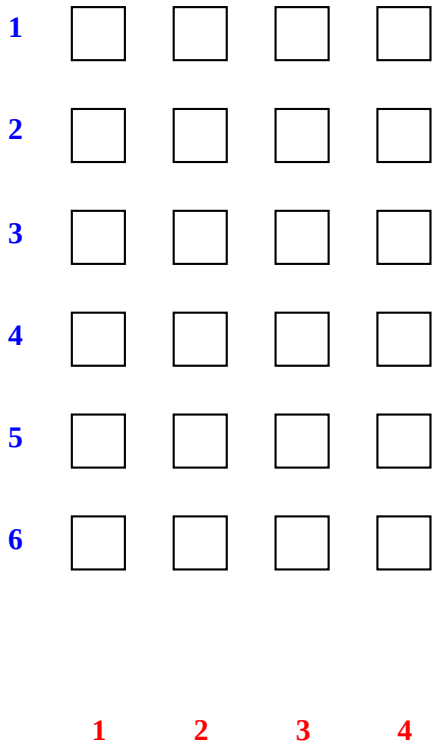
Invasion

Shearsort invasiv: $N = 63$ **Schlüssel, Wunsch:** 7×9**-Gitter**



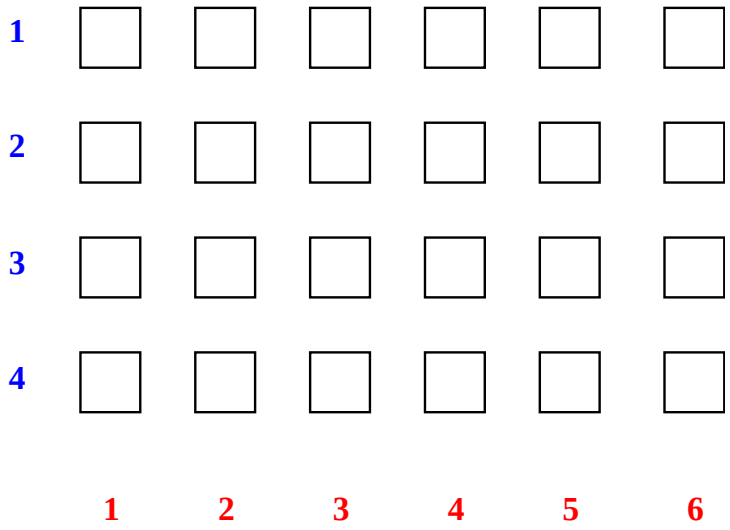
Invasion

Shearsort invasiv: $N = 63$ Schlüssel, Wunsch: 7×9-Gitter; bekommen: 6×4-Gitter



Invasion

Shearsort invasiv: 6×4**-Gitter**
besser uminterpretieren(!): 4×6**-Gitter**



Invasion

Jedes Prozessor-Element (PE) zuständig für

$$\lceil (7 \cdot 9)/(4 \cdot 6) \rceil = 3$$

Schlüssel.

$\Rightarrow$ **Wird gelöst durch Split&Merge**

Invasion

- bestimme optimale Werte für n und m;
  (Schätzung, was noch frei ist)
- Invasion nach Süden n;
- bekommen n' PEs;
- Invasion von jedem PE aus nach Osten m;
- bekommen Minimalzahl m' PEs;
- gib nicht benötigte PEs frei;
- PEs sind für $\lceil n \cdot m/(n' \cdot m') \rceil$ Schlüssel zuständig;
- if $n' > m'$
  then
    führe Shearsort auf dem $m' \times n'$-Gitter aus
  else
    führe Shearsort auf dem $n' \times m'$-Gitter aus

Code-Fragment

```
program HybridInvasiveSorter
{ /* Type alias definitions */
  /* Variable declarations */
  bool sucess[M];
  int Pinv[M];
  int Pmax;
  int keys[N];
  int sort[M];
  /* Parameter declarations */
  parameter M;
  parameter N;
  /* Program blocks */
  par (i >= 1 and i <= M)
  { /* Infect M arrays with algorithm SIS */
   sucess[i] = infect(PE(i,1), SIS);
  }
  seq {
   if AND[1 <= i <= M] success[i]
     {
        par (i >= 1 and i <= M) {
         Pinv[i] = invade(PE(i,1), EAST);
        }
        Pmax = MIN[1 <= i <= M] Pinv[i];
        /* Free PEs again such that all arrays have same size Pmax */
        par (i >= 1 and i <= M) {
         retreat(PE(i,1), Pmax+1, Pinv[i]);
        }
```

Code-Fragment

```
        repeat ⌈log M⌉ + 1 times
        {
          par (i >= 1 and i <= M) {
            if odd(i) {
              sort in row i the keys 2*Pmax*(i-1)+1 ... 2*Pmax*i into ascending order }
            else {
              sort in row i the keys 2*Pmax*(i-1)+1 ... 2*Pmax*i into descending order }
          }
          par (j >= 1 and j <= Pmax) {
              sort in colum j the keys j, j+2*Pmax, j+4*Pmax, j+6*Pmax ... into ascending order }
          par (j >= 1 and j <= Pmax) {
              sort in colum j the keys Pmax+j, j+3*Pmax, j+5*Pmax, j+7*Pmax ... into ascending
              order }
      }
    }
}
```

Konkrete Arbeiten

- Einarbeitung in einen ganze Reihe von Algorithmen wie Paralleles Sortieren, Tree-Traversal, k-SAT, Minimale Spannbäume, ...
- Einarbeitung in am LS vorhandene Parallele Architekturen wie *Field-programmable Gate Arrays* (FPGAs), Cell Processors (bekannt aus der PlayStation 3), Intel Quadcore, ...
- Die Algorithmen fit machen für die Invasion auf den Architekturen und implementieren.