POLITECNICO
MILANO 1863

Sorting on Two-Dimensional Processor Arrays

Hans Werner Lang

Conti Alessandro

04/06/2025

Table of Contents

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. Conclusion

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

Introduction

The definition of the sorting problem has to be generalized in order to cope with two-dimensional arrays (and moreover, arbitrary arrangements) of data.

Introduction (2)

Definition (Data Record)

A **Data Record** is a mapping

$$a : J \to A$$

Where $J = \{0, \dots, n-1\} \times \{0, \dots, n-1\}$ is a finite index set representing a two-dimensional array of size $n \times n$ and A is a set of data elements equipped with some order relation $\preceq$.
We denote the elements of the record as

$$\mathrm{A} = \mathrm{a_{i,j}}\Big|_{(i,j) \in J}$$

Introduction (3)

Example (Data Record)

Suppose:

- $J = \{0, 1\} \times \{0, 1\}$;
- $A = \mathbb{N}$;
- $\preceq = \leq$;
- $a : J \to \mathbb{N}$, that specifies a value for each index

$$a(0,0) = 4 \quad a(0,1) = 2 \quad a(1,0) = 7 \quad a(1,1) = 1$$

The data record can be represented as

$$\mathrm{A} = \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 7 & 1 \end{bmatrix}$$

Introduction (4)

Definition (Sorting Order)

A **Sorting Order** is a bijective mapping

$$\rho : J \to \{0, \ldots, |J| - 1\}$$

- *row-major*: $\rho(i, j) = i \cdot n + j$
- *snake-like*: $\rho(i, j) = \begin{cases} i \cdot n + j & i \text{ even} \\ i \cdot n + |J| - 1 - j & i \text{ odd} \end{cases}$

Introduction (5)

Example (Sorting Order)

- Row-major order

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

- Snake-like order

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 8 & 7 & 6 & 5 \\ 9 & 10 & 11 & 12 \\ 16 & 15 & 14 & 13 \end{bmatrix}$$

Introduction (6)

Definition (Generalized Sorting Problem)

The **Generalized Sorting Problem** is defined to reorder a data record $\mathrm{a}_{\mathrm{i,j}}\Big|_{(i,j)\in J}$ to a data record $\mathrm{a}_{\varphi(\mathrm{i,j})}\Big|_{(i,j)\in J}$ such that:

$$\forall (i_1, i_2), (j_1, j_2) \in J : \mathrm{a}_{\varphi(\mathrm{i}_1,\mathrm{i}_2)}\Big|_{(i,j)\in J} \preceq \mathrm{a}_{\varphi(\mathrm{i,j})}\Big|_{(j_1,j_2)\in J}$$
$$\textit{for } \rho(i_1, i_2) \prec \rho(j_1, j_2)$$

The function φ is a permutation of the index set J.

Example (Generalized Sorting Problem)

Suppose:

- $J = \{0,1\} \times \{0,1\}$;
- $A = \mathbb{N}$;
- $\preceq = \leq$;
- $\mathrm{A} = \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} = \begin{bmatrix} 7 & 3 \\ 4 & 9 \end{bmatrix}$;
- $\rho(i,j) = i \cdot n + j$;
- $\varphi : J \to J$ the permutation function

$$\varphi((0,0)) = (0,1) \quad \varphi((0,1)) = (1,0)$$
$$\varphi((1,0)) = (0,0) \quad \varphi((1,1)) = (1,1)$$

Now the sorted data record is

$$\mathrm{A}_\varphi = \begin{bmatrix} 3 & 4 \\ 7 & 9 \end{bmatrix}$$

Introduction (8)

Theorem (0-1-Principle)

If a sorting network sorts every sequence of 0*'s and* 1*'s, then it sorts every arbitrary sequence of values.*

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

LS3-Sort Introduction

The **LS3-Sort** is a very simple algorithm for sorting on a two-dimensional mesh.

The algorithm is based on a merge algorithm that merges 4 sorted $\frac{k}{2} \times \frac{k}{2}$-arrays to one sorted $k \times k$-array.

The sorting direction is the snake.

LS3-Sort Introduction (2)

The merge algorithm use of the basic operations *shuffle* and *oets*.

- *Shuffle*: a deck of cards can be mixed by exactly interleaving its two halves;
- *oets*: stands for one step of odd-even transposition sort.

Definition (Odd-Even Transposition Sort)

The **Odd-Even Transposition Sort** for n input data consists of n comparators stages. In each stage, either all inputs at odd index positions or all inputs at even index positions are compared with their neighbours. Odd and even stages alternate. The number of comparators is $n\frac{n-1}{2}$.

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. Conclusion

LS3-Sort

Algorithm 1 LS3-sort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
function LS3Sort(M)
    if n > 1 then
        subNW ← getSubmatrix(M, "NW")
        subNW′ ← LS3Sort(subNW)
        subNE ← getSubmatrix(M, "NE")
        subNE′ ← LS3Sort(subNE)
        subSW ← getSubmatrix(M, "SW")
        subSW′ ← LS3Sort(subSW)
        subSE ← getSubmatrix(M, "SE")
        subSE′ ← LS3Sort(subSE)
        M′ ← assemble(subNW′, subNE′, subSW′, subSE′)
        M′ ← LS3Merge(M′)
        return M′
    else
        return M
    end if
end function
```

LS3-Merge

Algorithm 2 LS3-merge Algorithm

Input:

$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are snake-sorted

Output:

$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: snake-sorted matrix

function $\textsc{LS3Merge}(\mathrm{M})$
    for $i \leftarrow 0$ **to** $k-1$ **do**
        $\mathrm{M}_{\mathrm{tmp}}[i,:] \leftarrow \textsc{shuffle}(\mathrm{M}[i,:])$
    end for
    for $j \leftarrow 0$ **to** $k-1$ **step** 2 **do**
        $\langle \mathrm{M}_{\mathrm{tmp}}[:,j], \mathrm{M}_{\mathrm{tmp}}[:,j+1] \rangle \leftarrow \textsc{sortSnakeLike}(\mathrm{M}_{\mathrm{tmp}}[:,j], \mathrm{M}_{\mathrm{tmp}}[:,j+1])$
    end for
    $\mathrm{M}' \leftarrow \textsc{oets}(\mathrm{M}_{\mathrm{tmp}})$
    return M'
end function

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

LS3-Sort Correctness - Initial Situation

Our initial situation is an array filled with 0 and 1, whose four subarrays are sorted in a snake-like direction.

Each subarray contains a certain number of full rows, labelled in the figure as `a`, `b`, `c`, and `d`, and possibly an incomplete row.

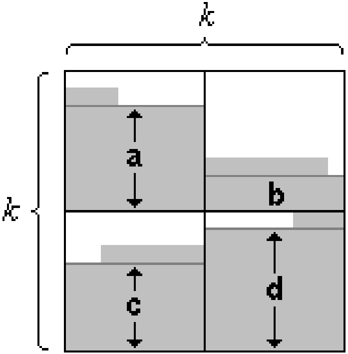


Figure: Initial situation

LS3-Sort Correctness (2) - Shuffle

After the SHUFFLE operation, in every double column there are $a + b + c + d$ ones that stem from the full rows plus at most four more ones from the incomplete rows.
Take the first row of subNW and the first row of subNE, join them to form the first row of the whole matrix. Then the second row of subNW and the second row of subNE, etc. Then do the same for subSW and subSE.

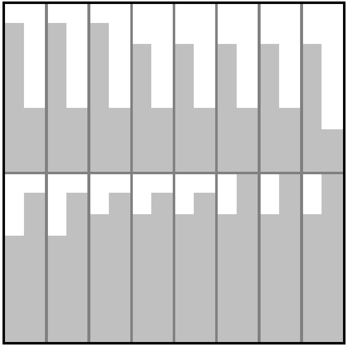


Figure: After Shuffle

LS3-Sort Correctness (3) - Sort Double Columns

Each double column is sorted in snake-like direction.

$a + b + c + d$ **is even**.
Then the 1s from the full rows form $\frac{a+b+c+d}{2}$ full rows.
Additionally, in each double column there are at most 4 1s from the incomplete rows. These 1s form an unsorted zone consisting of at most two rows.

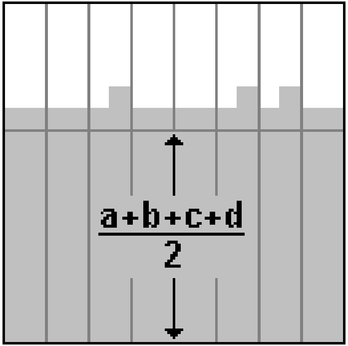


Figure: After Sort Double Column

LS3-Sort Correctness (4) - Sort Double Columns

Each double column is sorted in snake-like direction.

$a + b + c + d$ **is odd**.
Then the 1s from the full rows form a step in each double column.
However, this is only possible if all incomplete rows start from the same side. Since the sorting direction is the snake this implies that the numbers a, b, c, d are all even or all odd. But then their sum cannot be odd.

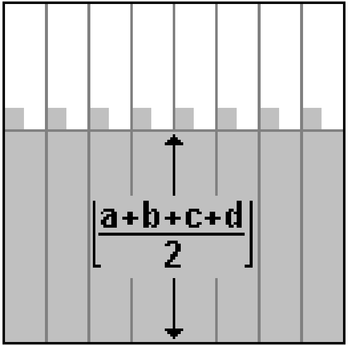


Figure: After Sort Double Column

LS3-Sort Correctness (5) - Apply OETS

Apply OETS and the unsorted zone is sorted.

The unsorted zone is at most $2k$, this means that requires $2k$ steps to sort that.

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

LS3-Merge Analysis

Given a matrix $k \times k$. The number of operation per step are:

- SHUFFLE along the rows: $\frac{1}{2}k$;
- SORTSNAKELIKE to each double column: $2k$;
- apply $2k$ OETS step to the snake: $2k$.

LS3-Sort Analysis

Given a matrix $n \times n$. The number of operation to sort a matrix are

$$\left(n + \frac{n}{2} + \frac{n}{4} + \ldots + 2\right)4.5 \leq 9n$$

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example
3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example
4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example
5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example
6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example
7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example
8 Conclusion

LS3-Sort Example

Algorithm 3 LS3-sort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function LS3SORT(M)
2      if n > 1 then
3          subNW ← GETSUBMATRIX(M, "NW")
4          subNW′ ← LS3SORT(subNW)
5          subNE ← GETSUBMATRIX(M, "NE")
6          subNE′ ← LS3SORT(subNE)
7          subSW ← GETSUBMATRIX(M, "SW")
8          subSW′ ← LS3SORT(subSW)
9          subSE ← GETSUBMATRIX(M, "SE")
10         subSE′ ← LS3SORT(subSE)
11         M′ ← ASSEMBLE(subNW′, subNE′, subSW′, subSE′)
12         M′ ← LS3MERGE(M′)
13         return M′
14     else
15         return M
16     end if
17 end function
```

Initial Situation

$$\begin{bmatrix} 0 & 1 & 7 & 12 \\ 4 & 5 & 9 & 8 \\ 13 & 6 & 15 & 14 \\ 3 & 2 & 11 & 10 \end{bmatrix}$$

Apply merge to the $\mathrm{submatrixNW}$ (line 12)

$$\begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix}$$

LS3-Sort Example (2)

Algorithm 4 LS3-merge Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are snake-sorted

Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: snake-sorted matrix

1 **function** $\text{LS3Merge}(\mathrm{M})$
2   **for** $i \leftarrow 0$ **to** $k-1$ **do**
3         $\mathrm{M_{tmp}}[i,:] \leftarrow \text{shuffle}(\mathrm{M}[1,:])$
4   **end for**
5   **for** $j \leftarrow 0$ **to** $k-1$ **step** 2 **do**
6         $\langle \mathrm{M_{tmp}}[:,j], \mathrm{M_{tmp}}[:,j+1] \rangle \leftarrow \text{sortSnakeLike}(\mathrm{M_{tmp}}[:,j], \mathrm{M_{tmp}}[:,j+1])$
7   **end for**
8     $\mathrm{M}' \leftarrow \text{oets}(\mathrm{M_{tmp}})$
9   **return** M'
10 **end function**

After shuffle (line 4)

$$\begin{bmatrix} 0 & 1 \\ 5 & 4 \end{bmatrix}$$

After sort the double columns (line 7)

$$\begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix}$$

After oets (line 9)

$$\begin{bmatrix} 0 & 1 \\ 5 & 4 \end{bmatrix}$$

LS3-Sort Example (3)

Algorithm 5 LS3-sort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function LS3Sort(M)
2      if n > 1 then
3          subNW ← getSubmatrix(M, "NW")
4          subNW′ ← LS3Sort(subNW)
5          subNE ← getSubmatrix(M, "NE")
6          subNE′ ← LS3Sort(subNE)
7          subSW ← getSubmatrix(M, "SW")
8          subSW′ ← LS3Sort(subSW)
9          subSE ← getSubmatrix(M, "SE")
10         subSE′ ← LS3Sort(subSE)
11         M′ ← assemble(subNW′, subNE′, subSW′, subSE′)
12         M′ ← LS3Merge(M′)
13         return M′
14     else
15         return M
16     end if
17 end function
```

Apply merge to the $\mathrm{submatrixNE}$ (line 12)

$$\begin{bmatrix} 7 & 12 \\ 9 & 8 \end{bmatrix}$$

LS3-Sort Example (4)

Algorithm 6 LS3-merge Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are snake-sorted

Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: snake-sorted matrix

1 **function** $\mathrm{LS3M{\scriptsize ERGE}}(\mathrm{M})$
2   **for** $i \leftarrow 0$ **to** $k-1$ **do**
3         $\mathrm{M}_{\mathrm{tmp}}[i,:] \leftarrow \textsc{shuffle}(\mathrm{M}[1,:])$
4   **end for**
5   **for** $j \leftarrow 0$ **to** $k-1$ **step** 2 **do**
6         $\langle \mathrm{M}_{\mathrm{tmp}}[:,j], \mathrm{M}_{\mathrm{tmp}}[:,j+1] \rangle \leftarrow \textsc{sortSnakeLike}(\mathrm{M}_{\mathrm{tmp}}[:,j], \mathrm{M}_{\mathrm{tmp}}[:,j+1])$
7   **end for**
8     $\mathrm{M}' \leftarrow \textsc{oets}(\mathrm{M}_{\mathrm{tmp}})$
9   **return** M'
10 **end function**

After shuffle (line 4)

$$\begin{bmatrix} 7 & 12 \\ 8 & 9 \end{bmatrix}$$

After sort the double columns (line 7)

$$\begin{bmatrix} 7 & 8 \\ 9 & 12 \end{bmatrix}$$

After oets (line 9)

$$\begin{bmatrix} 7 & 8 \\ 12 & 9 \end{bmatrix}$$

LS3-Sort Example (5)

Algorithm 7 LS3-sort Algorithm

Input:
$M \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$M' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function LS3SORT(M)
2      if n > 1 then
3          subNW ← GETSUBMATRIX(M, "NW")
4          subNW' ← LS3SORT(subNW)
5          subNE ← GETSUBMATRIX(M, "NE")
6          subNE' ← LS3SORT(subNE)
7          subSW ← GETSUBMATRIX(M, "SW")
8          subSW' ← LS3SORT(subSW)
9          subSE ← GETSUBMATRIX(M, "SE")
10         subSE' ← LS3SORT(subSE)
11         M' ← ASSEMBLE(subNW', subNE', subSW', subSE')
12         M' ← LS3MERGE(M')
13         return M'
14     else
15         return M
16     end if
17 end function
```

Apply merge to the submatrixSW (line 12)

$$\begin{bmatrix} 13 & 6 \\ 3 & 2 \end{bmatrix}$$

LS3-Sort Example (6)

Algorithm 8 LS3-merge Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k\times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are snake-sorted
Output:
$\mathrm{M}' \in \mathbb{Z}^{k\times k}$: snake-sorted matrix

```
1  function LS3Merge(M)
2      for i ← 0 to k − 1 do
3          M_tmp[i, :] ← shuffle(M[1, :])
4      end for
5      for j ← 0 to k − 1 step 2 do
6          ⟨M_tmp[:, j], M_tmp[:, j + 1]⟩ ← sortSnakeLike(M_tmp[:, j], M_tmp[:, j + 1])
7      end for
8      M' ← oets(M_tmp)
9      return M'
10 end function
```

After shuffle (line 4)

$$\begin{bmatrix} 13 & 6 \\ 2 & 3 \end{bmatrix}$$

After sort the double columns (line 7)

$$\begin{bmatrix} 2 & 3 \\ 6 & 13 \end{bmatrix}$$

After oets (line 9

$$\begin{bmatrix} 2 & 3 \\ 13 & 6 \end{bmatrix}$$

LS3-Sort Example (7)

Algorithm 9 LS3-sort Algorithm

Input:
$M \in \mathbb{Z}^{n \times n}$: unsorted matrix

Output:
$M' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function LS3Sort(M)
2      if n > 1 then
3          subNW ← getSubmatrix(M, "NW")
4          subNW′ ← LS3Sort(subNW)
5          subNE ← getSubmatrix(M, "NE")
6          subNE′ ← LS3Sort(subNE)
7          subSW ← getSubmatrix(M, "SW")
8          subSW′ ← LS3Sort(subSW)
9          subSE ← getSubmatrix(M, "SE")
10         subSE′ ← LS3Sort(subSE)
11         M′ ← assemble(subNW′, subNE′, subSW′, subSE′)
12         M′ ← LS3Merge(M′)
13         return M′
14     else
15         return M
16     end if
17 end function
```

Apply merge to the $\mathrm{submatrixSE}$ (line 12)

$$\begin{bmatrix} 15 & 14 \\ 11 & 10 \end{bmatrix}$$

LS3-Sort Example (8)

Algorithm 10 LS3-merge Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are snake-sorted
Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: snake-sorted matrix

1 **function** $\mathrm{LS3M\small{ERGE}}(\mathrm{M})$
2 **for** $i \leftarrow 0$ **to** $k-1$ **do**
3 $\mathrm{M}_{\mathrm{tmp}}[i,:] \leftarrow \mathrm{SHUFFLE}(\mathrm{M}[1,:])$
4 **end for**
5 **for** $j \leftarrow 0$ **to** $k-1$ **step** 2 **do**
6 $\langle \mathrm{M}_{\mathrm{tmp}}[:,j], \mathrm{M}_{\mathrm{tmp}}[:,j+1] \rangle \leftarrow \mathrm{SORTSNAKELIKE}(\mathrm{M}_{\mathrm{tmp}}[:,j], \mathrm{M}_{\mathrm{tmp}}[:,j+1])$
7 **end for**
8 $\mathrm{M}' \leftarrow \mathrm{OETS}(\mathrm{M}_{\mathrm{tmp}})$
9 **return** M'
10 **end function**

After shuffle (line 4)

$$\begin{bmatrix} 15 & 14 \\ 10 & 11 \end{bmatrix}$$

After sort the double columns (line 7)

$$\begin{bmatrix} 10 & 11 \\ 14 & 15 \end{bmatrix}$$

After oets (line 9)

$$\begin{bmatrix} 10 & 11 \\ 15 & 14 \end{bmatrix}$$

LS3-Sort Example (9)

Algorithm 11 LS3-sort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function LS3SORT(M)
2      if n > 1 then
3          subNW ← GETSUBMATRIX(M, "NW")
4          subNW′ ← LS3SORT(subNW)
5          subNE ← GETSUBMATRIX(M, "NE")
6          subNE′ ← LS3SORT(subNE)
7          subSW ← GETSUBMATRIX(M, "SW")
8          subSW′ ← LS3SORT(subSW)
9          subSE ← GETSUBMATRIX(M, "SE")
10         subSE′ ← LS3SORT(subSE)
11         M′ ← ASSEMBLE(subNW′, subNE′, subSW′, subSE′)
12         M′ ← LS3MERGE(M′)
13         return M′
14     else
15         return M
16     end if
17 end function
```

Apply merge to the matrix, whose submatrices are sorted in snake-like direction (line 12)

$$\begin{bmatrix} 0 & 1 & 7 & 8 \\ 5 & 4 & 12 & 9 \\ 2 & 3 & 10 & 11 \\ 13 & 6 & 15 & 14 \end{bmatrix}$$

LS3-Sort Example (10)

Algorithm 12 LS3-merge Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are snake-sorted
Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: snake-sorted matrix

```
1  function LS3MERGE(M)
2      for i ← 0 to k − 1 do
3          M_tmp[i, :] ← SHUFFLE(M[1, :])
4      end for
5      for j ← 0 to k − 1 step 2 do
6          ⟨M_tmp[:, j], M_tmp[:, j + 1]⟩ ← SORTSNAKELIKE(M_tmp[:, j], M_tmp[:, j + 1])
7      end for
8      M′ ← OETS(M_tmp)
9      return M′
10 end function
```

After shuffle (line 4)

$$\begin{bmatrix} 0 & 1 & 7 & 8 \\ 9 & 12 & 4 & 5 \\ 2 & 3 & 10 & 11 \\ 14 & 15 & 6 & 13 \end{bmatrix}$$

After sort the double columns (line 7)

$$\begin{bmatrix} 0 & 1 & 4 & 5 \\ 2 & 3 & 6 & 7 \\ 9 & 12 & 8 & 10 \\ 14 & 15 & 11 & 13 \end{bmatrix}$$

After oets (line 9)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. Conclusion

4-Way Mergesort Introduction

Definition (Roughly Sorted Array)

A $m \times n$-array of data is **Roughly Sorted**, if sorting of the rows suffices to sort the array completely

In a roughly sorted array each data element is already in its proper row.

Example (Roughly Sorted Array)

$$\begin{bmatrix} 3 & 0 & 2 & 1 \\ 4 & 5 & 6 & 7 \\ 11 & 10 & 9 & 8 \\ 13 & 12 & 15 & 14 \end{bmatrix}$$

4-Way Mergesort Introduction (2)

The idea of 4-way mergesort is to merge 4 roughly sorted $\frac{k}{2} \times \frac{k}{2}$-arrays to one roughly sorted $k \times k$-array.

The sorting direction is row-major.

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

4-Way MergeSort

Algorithm 13 4-Way Mergesort Algorithm

Input:

$\mathrm{M} \in \mathbb{Z}^{n\times n}$: unsorted matrix

Output:

$\mathrm{M}' \in \mathbb{Z}^{n\times n}$: row-major sorted matrix

```
function FOURWAYMERGESORT(M)
    M' ← ROUGHSORT(M)
    for i ← 0 to n − 1 do
        M'[i, :] ← SORT(M'[i, :], "ASC")
    end for
    return M'
end function
```

Algorithm 14 Roughsort Algorithm

Input:

$M \in \mathbb{Z}^{k \times k}$: unsorted matrix

Output:

$M' \in \mathbb{Z}^{k \times k}$: roughly-sorted matrix

```
function ROUGHSORT(M)
    if k > 1 then
        subNW ← GETSUBMATRIX(M, "NW")
        subNW' ← ROUGHSORT(subNW)
        subNE ← GETSUBMATRIX(M, "NE")
        subNE' ← ROUGHSORT(subNE)
        subSW ← GETSUBMATRIX(M, "SW")
        subSW' ← ROUGHSORT(subSW)
        subSE ← GETSUBMATRIX(M, "SE")
        subSE' ← ROUGHSORT(subSE)
        M' ← ASSEMBLE(subNW', subNE', subSW', subSE')
        M' ← FOURWAYMERGE(M')
        return M'
    else
        return M
    end if
end function
```

Algorithm 15 4-Way Merge Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are roughly-sorted

Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: roughly-sorted matrix

```
function FOURWAYMERGE(M)
    for i ← 0 to k − 1 do
        if i < k/2 then
            M'[i, :] ← SORT(M[i, :], "ASC")
        else
            M'[i, :] ← SORT(M[i, :], "DESC")
        end if
    end for
    for j ← 0 to k − 1 do
        M'[:, j] ← SORT(M'[:, j], "ASC")
    end for
    for i ← 0 to k − 1 do
        if i mod 2 = 1 then
            M'[i, :] ← SORT(M_tmp[i, :], "ASC")
        else
            M'[i, :] ← SORT(M[i, :], "DESC")
        end if
    end for
    for j ← 0 to k − 1 do
        M'[:, j] ← SORT(M'[:, j], "ASC")
    end for
end function
```

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. Conclusion

4-Way Merge Correctness - Initial Situation

Our initial situation is an array filled with 0 and 1, whose four subarrays are roughly sorted.

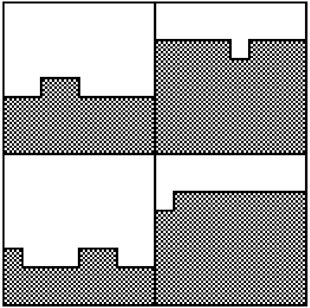


Figure: Initial situation

4-Way Merge Correctness (2) - Sort Rows

Sort the rows of the subarrays:

- ascending in the upper subarrays;
- descending in the lower subarrays.

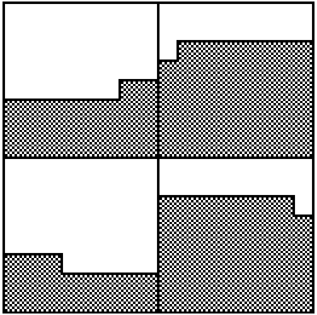


Figure: Sort the Rows of the Subarrays

4-Way Merge Correctness (3) - Sort Columns

Sort the column in ascending order.
Now, we have 2 roughly sorted $k \times \frac{k}{2}$-arrays.

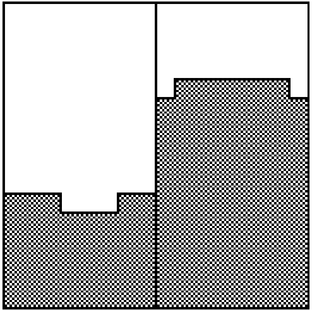


Figure: Sort the Columns

4-Way Merge Correctness (4) - Sort Rows

Sort the rows:

- odd rows ascending order;
- even rows descending order.

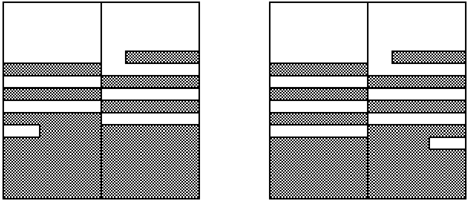


Figure: Sort the Rows

4-Way Merge Correctness (5) - Sort Columns

Sort the columns in ascending order.

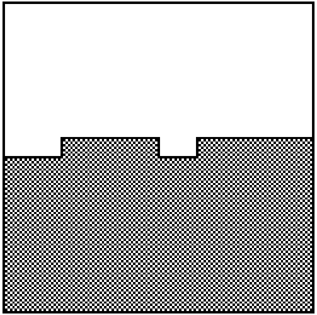


Figure: Sort the Columns

4-Way Merge Correctness (6) - Last Sort Rows

Sort the rows in ascending order.

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

4-Way Merge Analysis

Given a matrix $k \times k$. The number of operation per step are:

- *sort the rows of the subarrays*: $\frac{1}{2}k$;
- *sort the columns*: k;
- *sort the rows*: k;
- *sort the columns*: $\frac{1}{2}k$, since each column consists of two sorted subsequences shuffled into each other.

Roughsort Analysis

Given a matrix $n \times n$. The number of operation to sort a matrix are

$$\left(n + \frac{n}{2} + \frac{n}{4} + \ldots + 1\right)3 \leq 6n$$

4-Way Mergesort Analysis

Given a matrix $n \times n$. The number of operation to sort a matrix are

$$\left(n + \frac{n}{2} + \frac{n}{4} + \ldots + 1\right)3 + n \leq 7n$$

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

4-Way Mergesort Example

Algorithm 16 4-Way Mergesort Algorithm

Input:

$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix

Output:

$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: row-major sorted matrix

```
function FOURWAYMERGESORT(M)
    M' ← ROUGHSORT(M)
    for i ← 0 to n − 1 do
        M[i, :] ← SORT(M[i, :], "ASC")
    end for
    return M'
end function
```

Initial situation

$$\begin{bmatrix} 0 & 1 & 7 & 12 \\ 4 & 5 & 9 & 8 \\ 13 & 6 & 15 & 14 \\ 3 & 2 & 11 & 10 \end{bmatrix}$$

4-Way Mergesort Example (2)

Algorithm 17 Roughsort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: roughly-sorted matrix

```
function ROUGHSORT(M)
    if k > 1 then
        subNW ← GETSUBMATRIX(M, "NW")
        subNW′ ← ROUGHSORT(subNW)
        subNE ← GETSUBMATRIX(M, "NE")
        subNE′ ← ROUGHSORT(subNE)
        subSW ← GETSUBMATRIX(M, "SW")
        subSW′ ← ROUGHSORT(subSW)
        subSE ← GETSUBMATRIX(M, "SE")
        subSE′ ← ROUGHSORT(subSE)
        M′ ← ASSEMBLE(subNW′, subNE′, subSW′, subSE′)
        M′ ← FOURWAYMERGE(M′)
        return M′
    else
        return M
    end if
end function
```

Apply merge to the $\mathrm{submatrixNW}$

$$\begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix}$$

4-Way Mergesort Example (3)

Algorithm 18 4-Way Merge Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k\times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are roughly-sorted
Output:
$\mathrm{M}' \in \mathbb{Z}^{k\times k}$: roughly-sorted matrix

1 **function** $\textsc{fourWayMerge}(\mathrm{M})$
2   **for** $i \leftarrow 0$ **to** $k-1$ **do**
3   **if** $i < \frac{k}{2}$ **then**
4       $\mathrm{M}'[i,:] \leftarrow \textsc{sort}(\mathrm{M}[i,:], \text{"ASC"})$
5   **else**
6       $\mathrm{M}'[i,:] \leftarrow \textsc{sort}(\mathrm{M}[i,:], \text{"DESC"})$
7   **end if**
8   **end for**
9   **for** $j \leftarrow 0$ **to** $k-1$ **do**
10     $\mathrm{M}'[:,j] \leftarrow \textsc{sort}(\mathrm{M}'[:,j], \text{"ASC"})$
11   **end for**
12   **for** $i \leftarrow 0$ **to** $k-1$ **do**
13   **if** $i \mod 2 = 1$ **then**
14       $\mathrm{M}'[i,:] \leftarrow \textsc{sort}(\mathrm{M}_{\mathrm{tmp}}[i,:], \text{"ASC"})$
15   **else**
16       $\mathrm{M}'[i,:] \leftarrow \textsc{sort}(\mathrm{M}[i,:], \text{"DESC"})$
17   **end if**
18   **end for**
19   **for** $j \leftarrow 0$ **to** $k-1$ **do**
20     $\mathrm{M}'[:,j] \leftarrow \textsc{sort}(\mathrm{M}'[:,j], \text{"ASC"})$
21   **end for**
22 **end function**

After sort the rows of the submatrix (line 8)

$$\begin{bmatrix} 0 & 1 \\ 5 & 4 \end{bmatrix}$$

After sort the columns (line 11)

$$\begin{bmatrix} 0 & 1 \\ 5 & 4 \end{bmatrix}$$

After sort the rows in alternating order (line 18)

$$\begin{bmatrix} 1 & 0 \\ 4 & 5 \end{bmatrix}$$

After sort the columns (line 21)

$$\begin{bmatrix} 1 & 0 \\ 4 & 5 \end{bmatrix}$$

4-Way Mergesort Example (4)

Algorithm 19 Roughsort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: roughly-sorted matrix

```
function ROUGHSORT(M)
    if k > 1 then
        subNW ← GETSUBMATRIX(M, "NW")
        subNW′ ← ROUGHSORT(subNW)
        subNE ← GETSUBMATRIX(M, "NE")
        subNE′ ← ROUGHSORT(subNE)
        subSW ← GETSUBMATRIX(M, "SW")
        subSW′ ← ROUGHSORT(subSW)
        subSE ← GETSUBMATRIX(M, "SE")
        subSE′ ← ROUGHSORT(subSE)
        M′ ← ASSEMBLE(subNW′, subNE′, subSW′, subSE′)
        M′ ← FOURWAYMERGE(M′)
        return M′
    else
        return M
    end if
end function
```

Apply merge to the $\mathrm{submatrixNE}$

$$\begin{bmatrix} 7 & 12 \\ 9 & 8 \end{bmatrix}$$

4-Way Mergesort Example (5)

Algorithm 20 4-Way Merge Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are roughly-sorted

Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: roughly-sorted matrix

1 **function** FOURWAYMERGE(M)
2   **for** $i \leftarrow 0$ **to** $k-1$ **do**
3   **if** $i < \frac{k}{2}$ **then**
4       $\mathrm{M}'[i,:] \leftarrow \textsc{sort}(\mathrm{M}[i,:], \text{"ASC"})$
5   **else**
6       $\mathrm{M}'[i,:] \leftarrow \textsc{sort}(\mathrm{M}[i,:], \text{"DESC"})$
7   **end if**
8   **end for**
9   **for** $j \leftarrow 0$ **to** $k-1$ **do**
10     $\mathrm{M}'[:,j] \leftarrow \textsc{sort}(\mathrm{M}'[:,j], \text{"ASC"})$
11   **end for**
12   **for** $i \leftarrow 0$ **to** $k-1$ **do**
13   **if** $i \mod 2 = 1$ **then**
14       $\mathrm{M}'[i,:] \leftarrow \textsc{sort}(\mathrm{M}_{\mathrm{tmp}}[i,:], \text{"ASC"})$
15   **else**
16       $\mathrm{M}'[i,:] \leftarrow \textsc{sort}(\mathrm{M}[i,:], \text{"DESC"})$
17   **end if**
18   **end for**
19   **for** $j \leftarrow 0$ **to** $k-1$ **do**
20     $\mathrm{M}'[:,j] \leftarrow \textsc{sort}(\mathrm{M}'[:,j], \text{"ASC"})$
21   **end for**
22 **end function**

After sort the rows of the submatrix (line 8)

$$\begin{bmatrix} 7 & 12 \\ 9 & 8 \end{bmatrix}$$

After sort the columns (line 11)

$$\begin{bmatrix} 7 & 8 \\ 9 & 12 \end{bmatrix}$$

After sort the rows in alternating order (line 18)

$$\begin{bmatrix} 8 & 7 \\ 9 & 12 \end{bmatrix}$$

After sort the columns (line 21)

$$\begin{bmatrix} 8 & 7 \\ 9 & 12 \end{bmatrix}$$

4-Way Mergesort Example (6)

Algorithm 21 Roughsort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k\times k}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{k\times k}$: roughly-sorted matrix

```
function ROUGHSORT(M)
    if k > 1 then
        subNW ← GETSUBMATRIX(M, "NW")
        subNW' ← ROUGHSORT(subNW)
        subNE ← GETSUBMATRIX(M, "NE")
        subNE' ← ROUGHSORT(subNE)
        subSW ← GETSUBMATRIX(M, "SW")
        subSW' ← ROUGHSORT(subSW)
        subSE ← GETSUBMATRIX(M, "SE")
        subSE' ← ROUGHSORT(subSE)
        M' ← ASSEMBLE(subNW', subNE', subSW', subSE')
        M' ← FOURWAYMERGE(M')
        return M'
    else
        return M
    end if
end function
```

Apply merge to the $\mathrm{submatrixSW}$

$$\begin{bmatrix} 13 & 6 \\ 3 & 2 \end{bmatrix}$$

4-Way Mergesort Example (7)

Algorithm 22 4-Way Merge Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are roughly-sorted

Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: roughly-sorted matrix

```
1  function FOURWAYMERGE(M)
2      for i ← 0 to k − 1 do
3          if i < k/2 then
4              M'[i, :] ← SORT(M[i, :], "ASC")
5          else
6              M'[i, :] ← SORT(M[i, :], "DESC")
7          end if
8      end for
9      for j ← 0 to k − 1 do
10         M'[:, j] ← SORT(M'[:, j], "ASC")
11     end for
12     for i ← 0 to k − 1 do
13         if i  mod 2 = 1 then
14             M'[i, :] ← SORT(M_tmp[i, :], "ASC")
15         else
16             M'[i, :] ← SORT(M[i, :], "DESC")
17         end if
18     end for
19     for j ← 0 to k − 1 do
20         M'[:, j] ← SORT(M'[:, j], "ASC")
21     end for
22 end function
```

After sort the rows of the submatrix (line 8)

$$\begin{bmatrix} 6 & 13 \\ 3 & 2 \end{bmatrix}$$

After sort the columns (line 11)

$$\begin{bmatrix} 3 & 2 \\ 6 & 13 \end{bmatrix}$$

After sort the rows in alternating order (line 18)

$$\begin{bmatrix} 3 & 2 \\ 6 & 13 \end{bmatrix}$$

After sort the columns (line 21)

$$\begin{bmatrix} 3 & 2 \\ 6 & 13 \end{bmatrix}$$

4-Way Mergesort Example (8)

Algorithm 23 Roughsort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: roughly-sorted matrix

```
function ROUGHSORT(M)
    if k > 1 then
        subNW ← GETSUBMATRIX(M, "NW")
        subNW' ← ROUGHSORT(subNW)
        subNE ← GETSUBMATRIX(M, "NE")
        subNE' ← ROUGHSORT(subNE)
        subSW ← GETSUBMATRIX(M, "SW")
        subSW' ← ROUGHSORT(subSW)
        subSE ← GETSUBMATRIX(M, "SE")
        subSE' ← ROUGHSORT(subSE)
        M' ← ASSEMBLE(subNW', subNE', subSW', subSE')
        M' ← FOURWAYMERGE(M')
        return M'
    else
        return M
    end if
end function
```

Apply merge to the $\mathrm{submatrixSE}$

$$\begin{bmatrix} 15 & 14 \\ 11 & 10 \end{bmatrix}$$

4-Way Mergesort Example (9)

Algorithm 24 4-Way Merge Algorithm

Input:
$M \in \mathbb{Z}^{k \times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are roughly-sorted
Output:
$M' \in \mathbb{Z}^{k \times k}$: roughly-sorted matrix

```
 1 function FOURWAYMERGE(M)
 2     for i ← 0 to k − 1 do
 3         if i < k/2 then
 4             M'[i, :] ← SORT(M[i, :], "ASC")
 5         else
 6             M'[i, :] ← SORT(M[i, :], "DESC")
 7         end if
 8     end for
 9     for j ← 0 to k − 1 do
10         M'[:, j] ← SORT(M'[:, j], "ASC")
11     end for
12     for i ← 0 to k − 1 do
13         if i  mod 2 = 1 then
14             M'[i, :] ← SORT(M_tmp[i, :], "ASC")
15         else
16             M'[i, :] ← SORT(M[i, :], "DESC")
17         end if
18     end for
19     for j ← 0 to k − 1 do
20         M'[:, j] ← SORT(M'[:, j], "ASC")
21     end for
22 end function
```

After sort the rows of the submatrix (line 8)

$$\begin{bmatrix} 14 & 15 \\ 11 & 10 \end{bmatrix}$$

After sort the columns (line 11)

$$\begin{bmatrix} 11 & 10 \\ 14 & 15 \end{bmatrix}$$

After sort the rows in alternating order (line 18)

$$\begin{bmatrix} 11 & 10 \\ 14 & 15 \end{bmatrix}$$

After sort the columns (line 21)

$$\begin{bmatrix} 11 & 10 \\ 14 & 15 \end{bmatrix}$$

4-Way Mergesort Example (10)

Algorithm 25 Roughsort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: roughly-sorted matrix

```
function ROUGHSORT(M)
    if k > 1 then
        subNW ← GETSUBMATRIX(M, "NW")
        subNW′ ← ROUGHSORT(subNW)
        subNE ← GETSUBMATRIX(M, "NE")
        subNE′ ← ROUGHSORT(subNE)
        subSW ← GETSUBMATRIX(M, "SW")
        subSW′ ← ROUGHSORT(subSW)
        subSE ← GETSUBMATRIX(M, "SE")
        subSE′ ← ROUGHSORT(subSE)
        M′ ← ASSEMBLE(subNW′, subNE′, subSW′, subSE′)
        M′ ← FOURWAYMERGE(M′)
        return M′
    else
        return M
    end if
end function
```

Apply merge to the matrix, whose submatrices are roughly sorted

$$\begin{bmatrix} 1 & 0 & 8 & 7 \\ 4 & 5 & 9 & 12 \\ 3 & 2 & 11 & 10 \\ 6 & 13 & 14 & 15 \end{bmatrix}$$

4-Way Mergesort Example (11)

Algorithm 26 4-Way Merge Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{k \times k}$: unsorted matrix, whose submatrices of size $\frac{k}{2} \times \frac{k}{2}$ are roughly-sorted

Output:
$\mathrm{M}' \in \mathbb{Z}^{k \times k}$: roughly-sorted matrix

```
1  function FOURWAYMERGE(M)
2      for i ← 0 to k − 1 do
3          if i < k/2 then
4              M'[i, :] ← SORT(M[i, :], "ASC")
5          else
6              M'[i, :] ← SORT(M[i, :], "DESC")
7          end if
8      end for
9      for j ← 0 to k − 1 do
10         M'[:, j] ← SORT(M'[:, j], "ASC")
11     end for
12     for i ← 0 to k − 1 do
13         if i  mod 2 = 1 then
14             M'[i, :] ← SORT(M_tmp[i, :], "ASC")
15         else
16             M'[i, :] ← SORT(M[i, :], "DESC")
17         end if
18     end for
19     for j ← 0 to k − 1 do
20         M'[:, j] ← SORT(M'[:, j], "ASC")
21     end for
22 end function
```

After sort the rows of the submatrix (line 8)

$$\begin{bmatrix} 0 & 1 & 7 & 8 \\ 4 & 5 & 9 & 12 \\ 11 & 10 & 3 & 2 \\ 15 & 14 & 13 & 6 \end{bmatrix}$$

After sort the columns (line 11)

$$\begin{bmatrix} 0 & 1 & 3 & 2 \\ 4 & 5 & 7 & 6 \\ 11 & 10 & 9 & 8 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

After sort the rows in alternating order (line 18)

$$\begin{bmatrix} 3 & 2 & 1 & 0 \\ 4 & 5 & 6 & 7 \\ 11 & 10 & 9 & 8 \\ 12 & 13 & 14 & 15 \end{bmatrix}$$

After sort the columns (line 21)

$$\begin{bmatrix} 3 & 2 & 1 & 0 \\ 4 & 5 & 6 & 7 \\ 11 & 10 & 9 & 8 \\ 12 & 13 & 14 & 15 \end{bmatrix}$$

4-Way Mergesort Example (12)

Algorithm 27 4-Way Mergesort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: row-major sorted matrix

```
1 function FOURWAYMERGESORT(M)
2     M′ ← ROUGHSORT(M)
3     for i ← 0 to n − 1 do
4         M′[i, :] ← SORT(M′[i, :], "ASC")
5     end for
6     return M′
7 end function
```

After roughsort (line 3)

$$\begin{bmatrix} 3 & 2 & 1 & 0 \\ 4 & 5 & 6 & 7 \\ 11 & 10 & 9 & 8 \\ 12 & 13 & 14 & 15 \end{bmatrix}$$

4-Way Mergesort Example (13)

Algorithm 28 4-Way Mergesort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: row-major sorted matrix

1 **function** $\textsc{FourWayMergesort}(\mathrm{M})$
2     $\mathrm{M}' \leftarrow \textsc{RoughSort}(\mathrm{M})$
3   **for** $i \leftarrow 0$ **to** $n - 1$ **do**
4         $\mathrm{M}'[i, :] \leftarrow \textsc{Sort}(\mathrm{M}'[i, :], \text{"ASC"})$
5   **end for**
6   **return** M'
7 **end function**

After last row sorting (line 5)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix}$$

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 **Rotatesort**
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

Rotatesort Introduction

The rotatesort is characterized by consisting of a constant number of phases. A phase is a maximal sequence of operations that apply to rows or columns, respectively.

The sorting direction is snake-like.

Rotatesort Introduction (2)

The $n \times n$-array is decomposed into

- vertical slices: $n \times \sqrt{n}$-subarray;
- horizontal slices: $\sqrt{n} \times n$-subarray;
- blocks: $\sqrt{n} \times \sqrt{n}$-subarray.

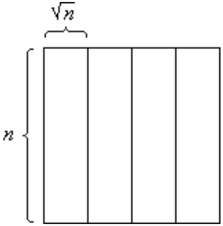


Figure: Vertical Slice

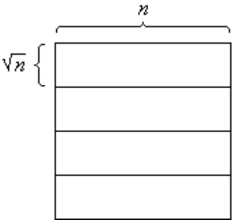


Figure: Horizontal Slice

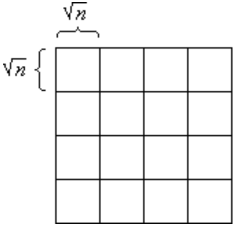


Figure: Block

Rotatesort Introduction (3)

Definition (Clean)

An $r \times s$-subarray is **Clean**, if it consists of 0s or 1s only.

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

Rotatesort

Algorithm 29 Rotatesort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: row-major sorted matrix

```
function ROTATESORT(M)
    for k ← 0 to √n − 1 do
        j ← k (√n + 1)
        verticalSlice ← M[:, j ... j + √n]
        M′[:, j ... j + √n] ← BALANCE(verticalSlice)
    end for
    M′ ← UNBLOCK(M′)
    for k ← 0 to √n − 1 do
        i ← k (√n + 1)
        horizontalSlice ← M′[i ... i + √n, :]
        M′[i ... i + √n, :] ← BALANCE(horizontalSlice^T)^T
    end for
    M′ ← UNBLOCK(M′)
    M′ ← SHEAR(M′)
    M′ ← SHEAR(M′)
    M′ ← SHEAR(M′)
    for i ← 0 to n − 1 do
        M′[i, :] ← SORT(M′[i, :], "ASC")
    end for
    return M′
end function
```

Balance

Algorithm 30 Balance Operation

Input:

$\mathrm{S} \in \mathbb{Z}^{k \times \sqrt{k}}$: unsorted slice

Output:

$\mathrm{S}' \in \mathbb{Z}^{k \times \sqrt{k}}$: sorted slice

```
function BALANCE(S, text)
    for j ← 0 to √k − 1 do
        S′[:, j] ← SORT(S[:, j], "ASC")
    end for
    for i ← 0 to k − 1 do
        S′[i, :] ← ROTATE(S′[i, :], i mod √k)
    end for
    for j ← 0 to √k − 1 do
        S′[:, j] ← SORT(S′[:, j], "ASC")
    end for
    return S′
end function
```

Unblock

Algorithm 31 Unblock Operation

Input:

$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix

Output:

$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: matrix in which the elements of each block are distributed over the entire width of the matrix

function $\textsc{unblock}(\mathrm{M})$
    for $i \leftarrow 0$ **to** $n - 1$ **do**
        $\mathrm{M}'[i, :] \leftarrow \textsc{rotate}(\mathrm{M}[i, :], i\sqrt{n} \mod n)$
    end for
    for $j \leftarrow 0$ **to** $n - 1$ **do**
        $\mathrm{M}'[:, j] \leftarrow \textsc{sort}(\mathrm{M}'[:, j], \text{"ASC"})$
    end for
    return M'
end function

Shear

Algorithm 32 Shear Operation

Input:

$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix

Output:

$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: matrix with half of dirty rows number

```
function SHEAR(M)
    for i ← 0 to n − 1 do
        if i mod 2 = 0 then
            M′[i, :] ← SORT(M[i, :], "ASC")
        else
            M′[i, :] ← SORT(M[i, :], "DESC")
        end if
    end for
    for j ← 0 to n − 1 do
        M′[:, j] ← SORT(M[:, j], "ASC")
    end for
    return M′
end function
```

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example
3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example
4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example
5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example
6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example
7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example
8 Conclusion

Rotatesort Correctness - Balance

BALANCE *Operation on vertical slice* reduces the number of dirty rows in the slice to at most $\sqrt{n}$. After sorting the columns each column of the slice is distributed to all columns of the slice by the rotation, then sorting again the column the 1's of each column contribute to a certain number of clean 1-rows and, possibly, a dirty row.

Altogether, there remain at most $2\sqrt{n}$ dirty blocks.

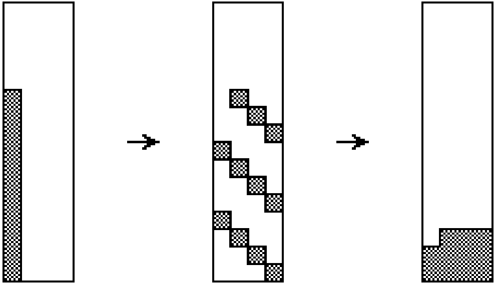


Figure: Balance Operation

Rotatesort Correctness (2) - Unblock

UNBLOCK *Operation.*

- distributes each block to all columns by the rotation;
- sorting the column, so, each clean block generates a clean row and each dirty block generates a dirty row.

Altogether, at most $2\sqrt{n}$ dirty rows that stem from the at most $2\sqrt{n}$ dirty blocks remain.

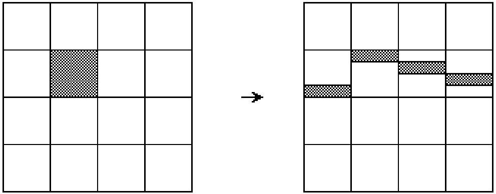


Figure: Unblock Operation

Rotatesort Correctness (3) - Balance

BALANCE *Operation on horizontal slice*.
The at most $2\sqrt{n}$ dirty rows contribute to at most 3 horizontal slices.

There remain at most 6 dirty blocks.

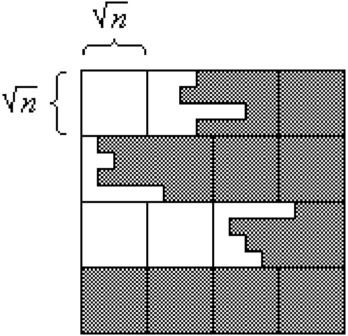


Figure: Balance Operation

Rotatesort Correctness (4) - Unblock

UNBLOCK *Operation* distributes the at most 6 dirty blocks to at most 6 dirty rows.

Rotatesort Correctness (5) - Shear

SHEAR *Operation* reduces every 2 dirty rows to at most 1 dirty row.

- Sorting the rows in alternating order;
- by sorting the columns a clean row is produced from every 2 dirty rows.

By the operation shear the number of dirty rows is halved. By the threefold application of shear the 6 dirty rows are reduced to 1 dirty row.

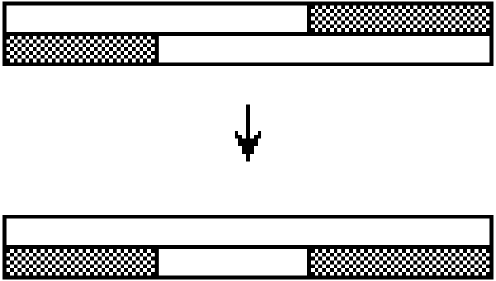


Figure: Shear Operation - clean row of 0

Rotatesort Correctness (6) - Sort

Eventually, *Sort the Last Unsorted Row*

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

Rotatesort Analysis

The analysis of rotatesort yields the following number of phases, where sorting or rotating a row or column:

- BALANCE: 3 phases;
- UNBLOCK: 2 phases;
- BALANCE: 3 phases;
- UNBLOCK: 2 phases;
- SHEAR: 3×2 phase;
- *sorting the last row*: 1 phase.

All phases require at most n elementary steps, so the algorithm is a $\mathcal{O}(17n)$

Rotatesort Analysis (2)

By a more careful analysis the constant is reduced to 10.

- In the BALANCE Operation, the rotation phases need not be counted, since they require only $\sqrt{n}$ elementary steps;
- in the UNBLOCK Operation, the rotation phase can be realized in $\frac{n}{2}$ elementary steps, by implementing a right rotation of $k > \frac{n}{2}$ and a left rotation of $n - k > \frac{n}{2}$ positions;
- the UNBLOCK Operation produces at most $3\sqrt{n}$ dirty rows; thus a corresponding number of sort steps suffices for sorting the columns.

This implies that the rotatesort requires at most $10n + \mathcal{O}(\sqrt{n})$ elementary steps.

Rotatesort Analysis (3)

Going to study complexity to constants:

- BALANCE operation on vertical slices requires:
  - sorting the columns: n;
  - rotate rows: $\sqrt{n}$;
  - sort columns: n
- UNBLOCK operation requires:
  - rotate rows: $\frac{n}{2}$;
  - sort columns: n
- BALANCE operation on vertical slices requires:
  - sorting the rows: n;
  - rotate columns: $\sqrt{n}$;
  - sort rows: n
- UNBLOCK requires:
  - rotate rows: $\frac{n}{2}$;
  - sort columns: $\leq 3\sqrt{n}$
- three times SHEAR operation requires:
  - sort rows: $3n$;
  - sort columns: $\leq 3\sqrt{n}$
- *sort the last unsorted row* requires: n.

So, the final complexity is $\leq 10n + 8\sqrt{n}$.

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example
3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example
4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example
5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example
6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example
7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example
8 Conclusion

Rotatesort Example

Algorithm 33 Rotatesort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: row-major sorted matrix

function $\textsc{Rotatesort}(\mathrm{M})$
  for $k \leftarrow 0$ **to** $\sqrt{n} - 1$ **do**
    $j \leftarrow k\left(\sqrt{n} + 1\right)$
    $\mathrm{verticalSlice} \leftarrow \mathrm{M}[:, j \dots j + \sqrt{n}]$
    $\mathrm{M}'[:, j \dots j + \sqrt{n}] \leftarrow \textsc{balance}(\mathrm{verticalSlice})$
  end for
  $\mathrm{M}' \leftarrow \textsc{unblock}(\mathrm{M}')$
  for $k \leftarrow 0$ **to** $\sqrt{n} - 1$ **do**
    $i \leftarrow k\left(\sqrt{n} + 1\right)$
    $\mathrm{horizontalSlice} \leftarrow \mathrm{M}'[i \dots i + \sqrt{n}, :]$
    $\mathrm{M}'[i \dots i + \sqrt{n}, :] \leftarrow \textsc{balance}(\mathrm{horizontalSlice}^{\intercal})^{\intercal}$
  end for
  $\mathrm{M}' \leftarrow \textsc{unblock}(\mathrm{M}')$
  $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
  $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
  $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
  for $i \leftarrow 0$ **to** $n - 1$ **do**
    $\mathrm{M}'[i, :] \leftarrow \textsc{sort}(\mathrm{M}'[i, :], \text{"ASC"})$
  end for
  return M'
end function

Initial situation

$$\begin{bmatrix} 0 & 1 & 7 & 12 \\ 4 & 5 & 9 & 8 \\ 13 & 6 & 15 & 14 \\ 3 & 2 & 11 & 10 \end{bmatrix}$$

Rotatesort Example (2)

Algorithm 34 Balance Operation

Input:

$\mathrm{S} \in \mathbb{Z}^{k \times \sqrt{k}}$: unsorted slice

Output:

$\mathrm{S}' \in \mathbb{Z}^{k \times \sqrt{k}}$: sorted slice

```
1  function BALANCE(S, text)
2      for j ← 0 to √k − 1 do
3          S′[:, j] ← SORT(S[:, j], "ASC")
4      end for
5      for i ← 0 to k − 1 do
6          S′[i, :] ← ROTATE(S′[i, :], i mod √k)
7      end for
8      for j ← 0 to √k − 1 do
9          S′[:, j] ← SORT(S′[:, j], "ASC")
10     end for
11     return S′
12 end function
```

First slice

$$\begin{bmatrix} 0 & 1 \\ 4 & 5 \\ 13 & 6 \\ 3 & 2 \end{bmatrix}$$

After rotate rows (line 7)

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \\ 6 & 13 \end{bmatrix}$$

After sort columns (line 4)

$$\begin{bmatrix} 0 & 1 \\ 3 & 2 \\ 4 & 5 \\ 13 & 6 \end{bmatrix}$$

After sort columns (line 10)

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \\ 6 & 13 \end{bmatrix}$$

Rotatesort Example (3)

Algorithm 35 Balance Operation

Input:
$S \in \mathbb{Z}^{k \times \sqrt{k}}$: unsorted slice
Output:
$S' \in \mathbb{Z}^{k \times \sqrt{k}}$: sorted slice

1 **function** BALANCE(S, *text*)
2 **for** $j \leftarrow 0$ **to** $\sqrt{k} - 1$ **do**
3 $S'[:, j] \leftarrow$ SORT($S[:, j]$, "ASC")
4 **end for**
5 **for** $i \leftarrow 0$ **to** $k - 1$ **do**
6 $S'[i, :] \leftarrow$ ROTATE($S'[i, :]$, $i \mod \sqrt{k}$)
7 **end for**
8 **for** $j \leftarrow 0$ **to** $\sqrt{k} - 1$ **do**
9 $S'[:, j] \leftarrow$ SORT($S'[:, j]$, "ASC")
10 **end for**
11 **return** S'
12 **end function**

Second slice

$$\begin{bmatrix} 7 & 12 \\ 9 & 8 \\ 15 & 14 \\ 11 & 10 \end{bmatrix}$$

After rotate rows (line 7)

$$\begin{bmatrix} 7 & 8 \\ 10 & 9 \\ 11 & 12 \\ 14 & 15 \end{bmatrix}$$

After sort columns (line 4)

$$\begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \\ 15 & 14 \end{bmatrix}$$

After sort columns (line 10)

$$\begin{bmatrix} 7 & 8 \\ 10 & 9 \\ 11 & 12 \\ 14 & 15 \end{bmatrix}$$

Rotatesort Example (4)

Algorithm 36 Rotatesort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: row-major sorted matrix

1 **function** $\textsc{Rotatesort}(\mathrm{M})$
2 **for** $k \leftarrow 0$ **to** $\sqrt{n} - 1$ **do**
3 $j \leftarrow k\left(\sqrt{n} + 1\right)$
4 $\mathrm{verticalSlice} \leftarrow \mathrm{M}[:, j \dots j + \sqrt{n}]$
5 $\mathrm{M}'[:, j \dots j + \sqrt{n}] \leftarrow \textsc{balance}(\mathrm{verticalSlice})$
6 **end for**
7 $\mathrm{M}' \leftarrow \textsc{unblock}(\mathrm{M}')$
8 **for** $k \leftarrow 0$ **to** $\sqrt{n} - 1$ **do**
9 $i \leftarrow k\left(\sqrt{n} + 1\right)$
10 $\mathrm{horizontalSlice} \leftarrow \mathrm{M}'[i \dots i + \sqrt{n}, :]$
11 $\mathrm{M}'[i \dots i + \sqrt{n}, :] \leftarrow \textsc{balance}(\mathrm{horizontalSlice}^{\intercal})^{\intercal}$
12 **end for**
13 $\mathrm{M}' \leftarrow \textsc{unblock}(\mathrm{M}')$
14 $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
15 $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
16 $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
17 **for** $i \leftarrow 0$ **to** $n - 1$ **do**
18 $\mathrm{M}'[i, :] \leftarrow \textsc{sort}(\mathrm{M}'[i, :], \text{"ASC"})$
19 **end for**
20 **return** M'
21 **end function**

After balance vertical slice (line 6)

$$\begin{bmatrix} 0 & 1 & 7 & 8 \\ 2 & 3 & 10 & 9 \\ 4 & 5 & 11 & 12 \\ 6 & 13 & 14 & 15 \end{bmatrix}$$

Rotatesort Example (5)

Algorithm 37 Unblock Operation

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix

Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: matrix in which the elements of each block are distributed over the entire width of the matrix

```
1 function UNBLOCK(M)
2     for i ← 0 to n − 1 do
3         M′[i, :] ← ROTATE(M[i, :], i√n  mod n)
4     end for
5     for j ← 0 to n − 1 do
6         M′[:, j] ← SORT(M′[:, j], "ASC")
7     end for
8     return M′
9 end function
```

After rotate rows (line 4)

$$\begin{bmatrix} 0 & 1 & 7 & 8 \\ 10 & 9 & 2 & 3 \\ 4 & 5 & 11 & 12 \\ 14 & 15 & 6 & 13 \end{bmatrix}$$

After sort columns (line 7)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 8 \\ 10 & 9 & 7 & 12 \\ 14 & 15 & 11 & 13 \end{bmatrix}$$

Rotatesort Example (6)

Algorithm 38 Rotatesort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: row-major sorted matrix

1 **function** $\textsc{Rotatesort}(\mathrm{M})$
2 **for** $k \leftarrow 0$ **to** $\sqrt{n} - 1$ **do**
3 $j \leftarrow k\left(\sqrt{n} + 1\right)$
4 $\mathrm{verticalSlice} \leftarrow \mathrm{M}[:, j \ldots j + \sqrt{n}]$
5 $\mathrm{M}'[:, j \ldots j + \sqrt{n}] \leftarrow \textsc{balance}(\mathrm{verticalSlice})$
6 **end for**
7 $\mathrm{M}' \leftarrow \textsc{unblock}(\mathrm{M}')$
8 **for** $k \leftarrow 0$ **to** $\sqrt{n} - 1$ **do**
9 $i \leftarrow k\left(\sqrt{n} + 1\right)$
10 $\mathrm{horizontalSlice} \leftarrow \mathrm{M}'[i \ldots i + \sqrt{n}, :]$
11 $\mathrm{M}'[i \ldots i + \sqrt{n}, :] \leftarrow \textsc{balance}(\mathrm{horizontalSlice}^\intercal)^\intercal$
12 **end for**
13 $\mathrm{M}' \leftarrow \textsc{unblock}(\mathrm{M}')$
14 $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
15 $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
16 $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
17 **for** $i \leftarrow 0$ **to** $n - 1$ **do**
18 $\mathrm{M}'[i, :] \leftarrow \textsc{sort}(\mathrm{M}'[i, :], \text{"ASC"})$
19 **end for**
20 **return** M'
21 **end function**

After unblock (line 7)

$$
\begin{bmatrix}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 8 \\
10 & 9 & 7 & 12 \\
14 & 15 & 11 & 13
\end{bmatrix}
$$

Rotatesort Example (7)

Algorithm 39 Balance Operation

Input:

$\mathrm{S} \in \mathbb{Z}^{k \times \sqrt{k}}$: unsorted slice

Output:

$\mathrm{S}' \in \mathbb{Z}^{k \times \sqrt{k}}$: sorted slice

1 **function** $\textsc{Balance}(\mathrm{S}, \textit{text})$
2   **for** $j \leftarrow 0$ **to** $\sqrt{k} - 1$ **do**
3         $\mathrm{S}'[:, j] \leftarrow \textsc{sort}(\mathrm{S}[:, j], \text{"ASC"})$
4   **end for**
5   **for** $i \leftarrow 0$ **to** $k - 1$ **do**
6         $\mathrm{S}'[i, :] \leftarrow \textsc{rotate}(\mathrm{S}'[i, :], i \mod \sqrt{k})$
7   **end for**
8   **for** $j \leftarrow 0$ **to** $\sqrt{k} - 1$ **do**
9         $\mathrm{S}'[:, j] \leftarrow \textsc{sort}(\mathrm{S}'[:, j], \text{"ASC"})$
10   **end for**
11   **return** S'
12 **end function**

The horizontal slice is treated as vertical slice and then transposed.
First slice

$$\begin{bmatrix} 0 & 4 \\ 1 & 5 \\ 2 & 6 \\ 3 & 8 \end{bmatrix}$$

After sort columns (line 4)

$$\begin{bmatrix} 0 & 4 \\ 1 & 5 \\ 2 & 6 \\ 3 & 8 \end{bmatrix}$$

After rotate rows (line 7)

$$\begin{bmatrix} 0 & 4 \\ 5 & 1 \\ 2 & 6 \\ 8 & 3 \end{bmatrix}$$

After sort columns (line 10)

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 5 & 4 \\ 8 & 6 \end{bmatrix}$$

Rotatesort Example (8)

Algorithm 40 Balance Operation

Input:

$S \in \mathbb{Z}^{k \times \sqrt{k}}$: unsorted slice

Output:

$S' \in \mathbb{Z}^{k \times \sqrt{k}}$: sorted slice

```
1  function BALANCE(S, text)
2      for j ← 0 to √k − 1 do
3          S′[:, j] ← SORT(S[:, j], "ASC")
4      end for
5      for i ← 0 to k − 1 do
6          S′[i, :] ← ROTATE(S′[i, :], i mod √k)
7      end for
8      for j ← 0 to √k − 1 do
9          S′[:, j] ← SORT(S′[:, j], "ASC")
10     end for
11     return S′
12 end function
```

The horizontal slice is treated as vertical slice and then transposed. Second slice

$$\begin{bmatrix} 10 & 14 \\ 9 & 15 \\ 7 & 11 \\ 12 & 13 \end{bmatrix}$$

After sort columns (line 4)

$$\begin{bmatrix} 7 & 11 \\ 9 & 13 \\ 10 & 14 \\ 12 & 15 \end{bmatrix}$$

After rotate rows (line 7)

$$\begin{bmatrix} 7 & 11 \\ 13 & 9 \\ 10 & 14 \\ 15 & 12 \end{bmatrix}$$

After sort columns (line 10)

$$\begin{bmatrix} 7 & 9 \\ 10 & 11 \\ 13 & 12 \\ 15 & 14 \end{bmatrix}$$

Rotatesort Example (9)

Algorithm 41 Rotatesort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: row-major sorted matrix

1 **function** $\textsc{Rotatesort}(\mathrm{M})$
2   **for** $k \leftarrow 0$ **to** $\sqrt{n} - 1$ **do**
3     $j \leftarrow k\left(\sqrt{n} + 1\right)$
4     $\mathrm{verticalSlice} \leftarrow \mathrm{M}[:, j \ldots j + \sqrt{n}]$
5     $\mathrm{M}'[:, j \ldots j + \sqrt{n}] \leftarrow \textsc{balance}(\mathrm{verticalSlice})$
6   **end for**
7   $\mathrm{M}' \leftarrow \textsc{unblock}(\mathrm{M}')$
8   **for** $k \leftarrow 0$ **to** $\sqrt{n} - 1$ **do**
9     $i \leftarrow k\left(\sqrt{n} + 1\right)$
10     $\mathrm{horizontalSlice} \leftarrow \mathrm{M}'[i \ldots i + \sqrt{n}, :]$
11     $\mathrm{M}'[i \ldots i + \sqrt{n}, :] \leftarrow \textsc{balance}(\mathrm{horizontalSlice}^\intercal)^\intercal$
12   **end for**
13   $\mathrm{M}' \leftarrow \textsc{unblock}(\mathrm{M}')$
14   $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
15   $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
16   $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
17   **for** $i \leftarrow 0$ **to** $n - 1$ **do**
18     $\mathrm{M}'[i, :] \leftarrow \textsc{sort}(\mathrm{M}'[i, :], \text{"ASC"})$
19   **end for**
20   **return** M'
21 **end function**

After balance horizontal slice (line 12)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 5 & 4 & 7 & 6 \\ 10 & 9 & 8 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

Rotatesort Example (10)

Algorithm 42 Unblock Operation

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: matrix in which the elements of each block are distributed over the entire width of the matrix

```
1 function UNBLOCK(M)
2     for i ← 0 to n − 1 do
3         M′[i, :] ← ROTATE(M[i, :], i√n mod n)
4     end for
5     for j ← 0 to n − 1 do
6         M′[:, j] ← SORT(M′[:, j], "ASC")
7     end for
8     return M′
9 end function
```

After rotate rows (line 4)

$$\begin{bmatrix} 0 & 2 & 5 & 8 \\ 4 & 6 & 1 & 3 \\ 7 & 10 & 13 & 15 \\ 12 & 14 & 9 & 11 \end{bmatrix}$$

After sort columns (line 7)

$$\begin{bmatrix} 0 & 2 & 1 & 3 \\ 4 & 6 & 5 & 8 \\ 7 & 10 & 9 & 11 \\ 12 & 14 & 13 & 15 \end{bmatrix}$$

Rotatesort Example (11)

Algorithm 43 Rotatesort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: row-major sorted matrix

1 **function** $\textsc{Rotatesort}(\mathrm{M})$
2 **for** $k \leftarrow 0$ **to** $\sqrt{n} - 1$ **do**
3 $j \leftarrow k\left(\sqrt{n} + 1\right)$
4 $\mathrm{verticalSlice} \leftarrow \mathrm{M}[:, j \ldots j + \sqrt{n}]$
5 $\mathrm{M}'[:, j \ldots j + \sqrt{n}] \leftarrow \textsc{balance}(\mathrm{verticalSlice})$
6 **end for**
7 $\mathrm{M}' \leftarrow \textsc{unblock}(\mathrm{M}')$
8 **for** $k \leftarrow 0$ **to** $\sqrt{n} - 1$ **do**
9 $i \leftarrow k\left(\sqrt{n} + 1\right)$
10 $\mathrm{horizontalSlice} \leftarrow \mathrm{M}'[i \ldots i + \sqrt{n}, :]$
11 $\mathrm{M}'[i \ldots i + \sqrt{n}, :] \leftarrow \textsc{balance}(\mathrm{horizontalSlice}^{\intercal})^{\intercal}$
12 **end for**
13 $\mathrm{M}' \leftarrow \textsc{unblock}(\mathrm{M}')$
14 $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
15 $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
16 $\mathrm{M}' \leftarrow \textsc{shear}(\mathrm{M}')$
17 **for** $i \leftarrow 0$ **to** $n - 1$ **do**
18 $\mathrm{M}'[i, :] \leftarrow \textsc{sort}(\mathrm{M}'[i, :], \text{"ASC"})$
19 **end for**
20 **return** M'
21 **end function**

After unblock (line 13)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 10 & 9 & 8 & 11 \\ 13 & 12 & 15 & 14 \end{bmatrix}$$

Rotatesort Example (12)

Algorithm 44 Shear Operation

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: matrix with half of dirty rows number

```
1  function SHEAR(M)
2      for i ← 0 to n − 1 do
3          if i  mod 2 = 0 then
4              M′[i, :] ← SORT(M[i, :], "ASC")
5          else
6              M′[i, :] ← SORT(M[i, :], "DESC")
7          end if
8      end for
9      for j ← 0 to n − 1 do
10         M′[:, j] ← SORT(M[:, j], "ASC")
11     end for
12     return M′
13 end function
```

After sort rows (line 8)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 8 & 6 & 5 & 4 \\ 7 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

After sort columns (line 11

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

Rotatesort Example (13)

Algorithm 45 Rotatesort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n\times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n\times n}$: row-major sorted matrix

```
1  function ROTATESORT(M)
2      for k ← 0 to √n − 1 do
3          j ← k (√n + 1)
4          verticalSlice ← M[:, j ... j + √n]
5          M′[:, j ... j + √n] ← BALANCE(verticalSlice)
6      end for
7      M′ ← UNBLOCK(M′)
8      for k ← 0 to √n − 1 do
9          i ← k (√n + 1)
10         horizontalSlice ← M′[i ... i + √n, :]
11         M′[i ... i + √n, :] ← BALANCE(horizontalSliceᵀ)ᵀ
12     end for
13     M′ ← UNBLOCK(M′)
14     M′ ← SHEAR(M′)
15     M′ ← SHEAR(M′)
16     M′ ← SHEAR(M′)
17     for i ← 0 to n − 1 do
18         M′[i, :] ← SORT(M′[i, :], "ASC")
19     end for
20     return M′
21 end function
```

After first shear (line 14)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

Rotatesort Example (14)

Algorithm 46 Shear Operation

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: matrix with half of dirty rows number

```
1  function SHEAR(M)
2      for i ← 0 to n − 1 do
3          if i  mod 2 = 0 then
4              M′[i, :] ← SORT(M[i, :], "ASC")
5          else
6              M′[i, :] ← SORT(M[i, :], "DESC")
7          end if
8      end for
9      for j ← 0 to n − 1 do
10         M′[:, j] ← SORT(M[:, j], "ASC")
11     end for
12     return M′
13 end function
```

After sort rows (line 8)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 8 & 6 & 5 & 4 \\ 7 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

After sort columns (line 11)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

Rotatesort Example (15)

Algorithm 47 Rotatesort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: row-major sorted matrix

```
1  function ROTATESORT(M)
2      for k ← 0 to √n − 1 do
3          j ← k (√n + 1)
4          verticalSlice ← M[:, j . . . j + √n]
5          M′[:, j . . . j + √n] ← BALANCE(verticalSlice)
6      end for
7      M′ ← UNBLOCK(M′)
8      for k ← 0 to √n − 1 do
9          i ← k (√n + 1)
10         horizontalSlice ← M′[i . . . i + √n, :]
11         M′[i . . . i + √n, :] ← BALANCE(horizontalSlice^T)^T
12     end for
13     M′ ← UNBLOCK(M′)
14     M′ ← SHEAR(M′)
15     M′ ← SHEAR(M′)
16     M′ ← SHEAR(M′)
17     for i ← 0 to n − 1 do
18         M′[i, :] ← SORT(M′[i, :], "ASC")
19     end for
20     return M′
21 end function
```

After second shear (line 15)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

Rotatesort Example (16)

Algorithm 48 Shear Operation

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: matrix with half of dirty rows number

```
1  function SHEAR(M)
2      for i ← 0 to n − 1 do
3          if i mod 2 = 0 then
4              M′[i, :] ← SORT(M[i, :], "ASC")
5          else
6              M′[i, :] ← SORT(M[i, :], "DESC")
7          end if
8      end for
9      for j ← 0 to n − 1 do
10         M′[:, j] ← SORT(M[:, j], "ASC")
11     end for
12     return M′
13 end function
```

After sort rows (line 8)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 8 & 6 & 5 & 4 \\ 7 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

After sort columns (line 11)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

Rotatesort Example (17)

Algorithm 49 Rotatesort Algorithm

Input:
$M \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$M' \in \mathbb{Z}^{n \times n}$: row-major sorted matrix

```
 1 function ROTATESORT(M)
 2     for k ← 0 to √n − 1 do
 3         j ← k (√n + 1)
 4         verticalSlice ← M[:, j ... j + √n]
 5         M'[:, j ... j + √n] ← BALANCE(verticalSlice)
 6     end for
 7     M' ← UNBLOCK(M')
 8     for k ← 0 to √n − 1 do
 9         i ← k (√n + 1)
10         horizontalSlice ← M'[i ... i + √n, :]
11         M'[i ... i + √n, :] ← BALANCE(horizontalSlice^T)^T
12     end for
13     M' ← UNBLOCK(M')
14     M' ← SHEAR(M')
15     M' ← SHEAR(M')
16     M' ← SHEAR(M')
17     for i ← 0 to n − 1 do
18         M'[i, :] ← SORT(M'[i, :], "ASC")
19     end for
20     return M'
21 end function
```

After third shear (line 16)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

After sorting the last row (line 19)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example
3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example
4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example
5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example
6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example
7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example
8 Conclusion

3n-Sort Introduction

The algorithm 3n-sort of Schnorr and Shamir is more complicated than rotatesort and 4-way mergesort.

The sorting direction is snake-like.

3n-Sort Introduction (2)

The $n \times n$-array is decomposed into

- vertical slices: $n \times n^{\frac{3}{4}}$-subarray;
- horizontal slices: $n^{\frac{3}{4}} \times n$-subarray;
- blocks: $n^{\frac{3}{4}} \times n^{\frac{3}{4}}$-subarray.

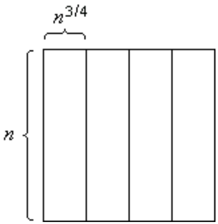


Figure: Vertical Slice

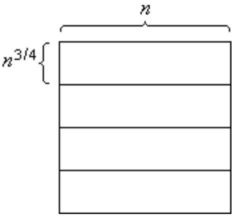


Figure: Horizontal Slice

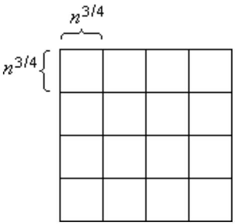


Figure: Block

3n-Sort Introduction (3)

Definition (k-way Unshuffle)

A **k-way Unshuffle** is a permutation that corresponds to dealing n cards to k players, it requires the $k \mid n$.

Example (4-Way Unshuffle)

Suppose to have 12 numbers, $0, 1, \ldots, 11$ and 4 players:

- player 1 receives: $0, 4, 8$;
- player 2 receives: $1, 5, 9$;
- player 3 receives: $2, 6, 10$;
- player 4 receives: $3, 7, 11$.

3n-Sort Introduction (4)

In a mathematical way:

- set size: n, in the example $n = 12$;
- number of way: k, in the example $k = 4$;
- block size: $m = \left\lceil \frac{n}{k} \right\rceil$, in the example $m = 3$;
- k-way unshuffle permutation:

$$\sigma : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$$
$$\sigma : i \mapsto (i \mod k) \cdot m + \left\lfloor \frac{i}{k} \right\rfloor$$

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

3n-Sort

Algorithm 50 3n-Sort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

function THREENSORT(M)
  for $i \leftarrow 0$ **to** $n-1$ **step** $n^{3/4}$ **do**
    for $j \leftarrow 0$ **to** $n-1$ **step** $n^{3/4}$ **do**
      $\text{block} \leftarrow \mathrm{M}[i \ldots i + n^{3/4} - 1,\ j \ldots j + n^{3/4} - 1]$
      $\mathrm{M}'[i \ldots i + n^{3/4} - 1,\ j \ldots j + n^{3/4} - 1] \leftarrow \textsc{sort}(\text{block}, \text{"ASC"})$
    end for
  end for
  for $i \leftarrow 0$ **to** $n-1$ **do**
    $\mathrm{M}' \leftarrow \textsc{kWayUnshuffle}(\mathrm{M}[i,:], n^{\frac{1}{4}})$
  end for
  for $i \leftarrow 0$ **to** $n-1$ **step** $n^{3/4}$ **do**
    for $j \leftarrow 0$ **to** $n-1$ **step** $n^{3/4}$ **do**
      $\text{block} \leftarrow \mathrm{M}[i \ldots i + n^{3/4} - 1,\ j \ldots j + n^{3/4} - 1]$
      $\mathrm{M}'[i \ldots i + n^{3/4} - 1,\ j \ldots j + n^{3/4} - 1] \leftarrow \textsc{sort}(\text{block}, \text{"ASC"})$
    end for
  end for
  for $j \leftarrow 0$ **to** $n-1$ **do**
    $\mathrm{M}'[:, j] \leftarrow \textsc{sort}(\mathrm{M}'[:, j], \text{"ASC"})$
  end for
  for $k \leftarrow 0$ **to** $n^{\frac{1}{4}} - 1$ **do**
    $j \leftarrow k\left(n^{\frac{1}{4}} + 1\right)$
    $\text{verticalSlice} \leftarrow \mathrm{M}'[:, j \ldots j + n^{\frac{1}{4}}]$
    $\mathrm{M}'[:, j \ldots j + n^{\frac{1}{4}}] \leftarrow \textsc{sort}(\text{verticalSlice}, \text{"ASC"})$
  end for
  for $i \leftarrow 0$ **to** $n-1$ **do**
    if $i \mod 2 = 0$ **then**
      $\mathrm{M}'[i, :] \leftarrow \textsc{sort}(\mathrm{M}'[i, :], \text{"ASC"})$
    else
      $\mathrm{M}'[i, :] \leftarrow \textsc{sort}(\mathrm{M}'[i, :], \text{"DESC"})$
    end if
  end for
  $\mathrm{M}' \leftarrow \textsc{oets}(\mathrm{M}')$
  return M'
end function

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example
3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example
4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example
5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example
6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example
7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example
8 Conclusion

3n-Sort Correctness - Sort blocks

Sort the blocks:
after sorting the blocks each block has at most one incomplete row.

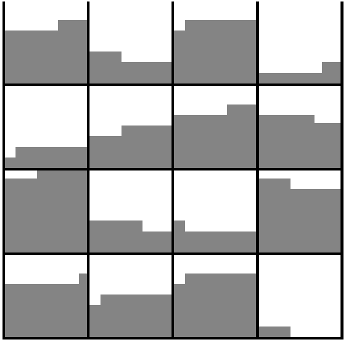


Figure: Sort the Block

3n-Sort Correctness (2) - $n^{\frac{1}{4}}$-Way Unshuffle Rows

$n^{\frac{1}{4}}$*-way unshuffle along the rows* distributes the columns of a block to all $n^{\frac{1}{4}}$ blocks of the same horizontal slice in a round-robin way.
If a block has an incomplete row, some blocks receive a one more from this block than others

Altogether, a block can receive at most $n^{\frac{1}{4}}$ ones more than any other.

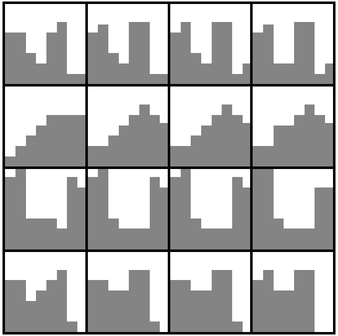


Figure: $n^{\frac{1}{4}}$-way unshuffle

3n-Sort Correctness (3) - Sort Blocks

Sort the block:
after sorting the blocks again each block contains at most one incomplete row.

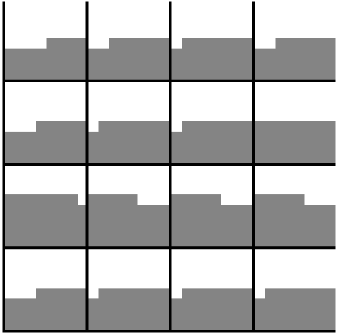


Figure: Sort the Block

3n-Sort Correctness (4) - Sort Columns

Sort the Columns:
after sorting the columns each vertical slice has at most $n^{\frac{1}{4}}$ dirty rows.

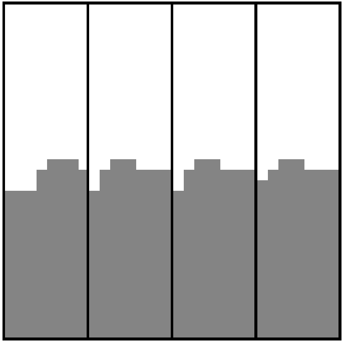


Figure: Sort the Columns

3n-Sort Correctness (5) - Sort Vertical Slices

Sort the Vertical Slices:
after sorting the vertical slices all vertical slices have almost the same number of complete 1-rows: the difference can be at most 1.

The region of length $n^{\frac{1}{2}}$ on the snake in each vertical slice that can contain these additional 1s.

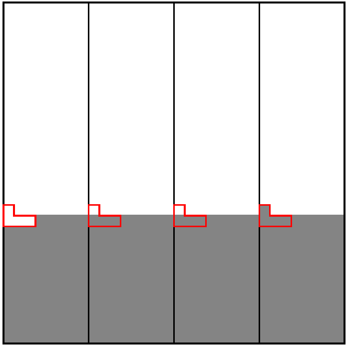


Figure: Sort the Vertical Slices

3n-Sort Correctness (6) - Sort Rows

Sort the Rows in Alternating Order:
the two last dirty rows are sorted in alternating direction.

Possibly still unsorted are the at most $n^{\frac{1}{4}} n^{\frac{1}{2}} = n^{\frac{3}{4}}$.

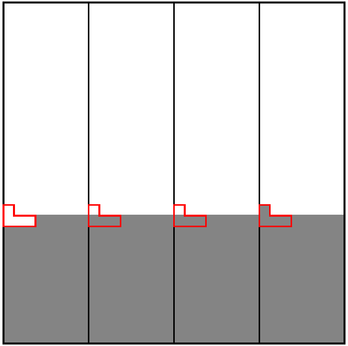


Figure: Sort the Rows

3n-Sort Correctness (7) - OETS

Apply $n^{\frac{3}{4}}$-steps of OETS *to the Snake*:
the unsorted zone of length at most $n^{\frac{3}{4}}$ is sorted according to the snake.

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. Conclusion

3n-Sort Analysis

- *sorting the block*: $\mathcal{O}\left(n^{\frac{3}{4}}\right)$;
- kWayUnshuffle: n;
- *sorting the block*: $\mathcal{O}\left(n^{\frac{3}{4}}\right)$;
- *sorting the columns*: n;
- *sorting the vertical slices*: $\mathcal{O}\left(n^{\frac{3}{4}}\right)$;
- *sorting the rows*: n;
- OETS steps: $\mathcal{O}\left(n^{\frac{3}{4}}\right)$.

This implies that the 3n-sort of Schnorr and Shamir requires at most $3n + \mathcal{O}\left(n^{\frac{3}{4}}\right)$.

3n-Sort Analysis (2)

Going to study complexity to constants:

- *sorting the block*: can be sorted using any linear sorting algorithm for example the 4-way mergesort, this implies that $\leq 7n^{3/4}$;
- KWAYUNSHUFFLE: n;
- *sorting the block*: can be sorted using any linear sorting algorithm for example the 4-way mergesort, this implies that $\leq 7n^{3/4}$;
- *sorting the columns*: n;
- *sorting the vertical slices*: can be sorted by sorting each vertical slice block independently, then sorting the blocks vertically overlapping by $n^{\frac{1}{4}}$ rows, this costs $\leq 7n^{\frac{3}{4}} + n^{\frac{1}{4}}$
- *sorting the rows*: n;
- OETS steps: each step is a comparison-exchange phase, so requires $n^{\frac{3}{4}}$.

So, the final complexity is $\leq 3n + 7n^{\frac{3}{4}} + n^{\frac{1}{4}}$.

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. **3n-Sort of Schnorr and Shamir**
   - Algorithm
   - Correctness
   - Analysis
   - **Example**
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. Conclusion

3n-Sort Example

Algorithm 51 3n-Sort Algorithm

Input:
$M \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$M' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
function THREENSORT(M)
    for i ← 0 to n − 1 step n^{3/4} do
        for j ← 0 to n − 1 step n^{3/4} do
            block ← M[i...i + n^{3/4} − 1, j...j + n^{3/4} − 1]
            M'[i...i + n^{3/4} − 1, j...j + n^{3/4} − 1] ← SORT(block, "ASC")
        end for
    end for
    for i ← 0 to n − 1 do
        M' ← KWAYUNSHUFFLE(M[i, :], n^{1/4})
    end for
    for i ← 0 to n − 1 step n^{3/4} do
        for j ← 0 to n − 1 step n^{3/4} do
            block ← M[i...i + n^{3/4} − 1, j...j + n^{3/4} − 1]
            M'[i...i + n^{3/4} − 1, j...j + n^{3/4} − 1] ← SORT(block, "ASC")
        end for
    end for
    for j ← 0 to n − 1 do
        M'[:, j] ← SORT(M'[:, j], "ASC")
    end for
    for k ← 0 to n^{1/4} − 1 do
        j ← k (n^{1/4} + 1)
        verticalSlice ← M'[:, j...j + n^{1/4}]
        M'[:, j...j + n^{1/4}] ← SORT(verticalSlice, "ASC")
    end for
    for i ← 0 to n − 1 do
        if i mod 2 = 0 then
            M'[i, :] ← SORT(M'[i, :], "ASC")
        else
            M'[i, :] ← SORT(M'[i, :], "DESC")
        end if
    end for
    M' ← OETS(M')
    return M'
end function
```

Initial situation

$$\begin{bmatrix} 0 & 1 & 7 & 12 \\ 4 & 5 & 9 & 8 \\ 13 & 6 & 15 & 14 \\ 3 & 2 & 11 & 10 \end{bmatrix}$$

3n-Sort Example (2)

Algorithm 52 3n-Sort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function THREENSORT(M)
2      for i ← 0 to n − 1 step n^{3/4} do
3          for j ← 0 to n − 1 step n^{3/4} do
4              block ← M[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1]
5              M'[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1] ← SORT(block, "ASC")
6          end for
7      end for
8      for i ← 0 to n − 1 do
9          M' ← KWAYUNSHUFFLE(M[i, :], n^{1/4})
10     end for
11     for i ← 0 to n − 1 step n^{3/4} do
12         for j ← 0 to n − 1 step n^{3/4} do
13             block ← M[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1]
14             M'[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1] ← SORT(block, "ASC")
15         end for
16     end for
17     for j ← 0 to n − 1 do
18         M'[:, j] ← SORT(M'[:, j], "ASC")
19     end for
20     for k ← 0 to n^{1/4} − 1 do
21         j ← k (n^{1/4} + 1)
22         verticalSlice ← M'[:, j ... j + n^{1/4}]
23         M'[:, j ... j + n^{1/4}] ← SORT(verticalSlice, "ASC")
24     end for
25     for i ← 0 to n − 1 do
26         if i mod 2 = 0 then
27             M'[i, :] ← SORT(M'[i, :], "ASC")
28         else
29             M'[i, :] ← SORT(M'[i, :], "DESC")
30         end if
31     end for
32     M' ← OETS(M')
33     return M'
34 end function
```

After sort the blocks (line 7)

$$\begin{bmatrix} 0 & 1 & 7 & 8 \\ 4 & 5 & 9 & 12 \\ 2 & 3 & 10 & 11 \\ 6 & 13 & 14 & 15 \end{bmatrix}$$

3n-Sort Example (3)

Algorithm 53 3n-Sort Algorithm

Input:
$M \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$M' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function THREENSORT(M)
2      for i ← 0 to n − 1 step n^{3/4} do
3          for j ← 0 to n − 1 step n^{3/4} do
4              block ← M[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1]
5              M'[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1] ← SORT(block, "ASC")
6          end for
7      end for
8      for i ← 0 to n − 1 do
9          M' ← KWAYUNSHUFFLE(M[i, :], n^{1/4})
10     end for
11     for i ← 0 to n − 1 step n^{3/4} do
12         for j ← 0 to n − 1 step n^{3/4} do
13             block ← M[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1]
14             M'[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1] ← SORT(block, "ASC")
15         end for
16     end for
17     for j ← 0 to n − 1 do
18         M'[:, j] ← SORT(M'[:, j], "ASC")
19     end for
20     for k ← 0 to n^{1/4} − 1 do
21         j ← k (n^{1/4} + 1)
22         verticalSlice ← M'[:, j ... j + n^{1/4}]
23         M'[:, j ... j + n^{1/4}] ← SORT(verticalSlice, "ASC")
24     end for
25     for i ← 0 to n − 1 do
26         if i mod 2 = 0 then
27             M'[i, :] ← SORT(M'[i, :], "ASC")
28         else
29             M'[i, :] ← SORT(M'[i, :], "DESC")
30         end if
31     end for
32     M' ← OETS(M')
33     return M'
34 end function
```

After apply $n^{\frac{3}{4}}$-way unshuffle along the rows (line 10)

$$
\begin{bmatrix}
0 & 7 & 1 & 8 \\
4 & 9 & 5 & 12 \\
2 & 10 & 3 & 11 \\
6 & 14 & 13 & 15
\end{bmatrix}
$$

3n-Sort Example (4)

Algorithm 54 3n-Sort Algorithm

Input:
$M \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$M' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function threeNSort(M)
2      for i ← 0 to n − 1 step n^{3/4} do
3          for j ← 0 to n − 1 step n^{3/4} do
4              block ← M[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1]
5              M'[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1] ← sort(block, "ASC")
6          end for
7      end for
8      for i ← 0 to n − 1 do
9          M' ← kWayUnshuffle(M[i, :], n^{1/4})
10     end for
11     for i ← 0 to n − 1 step n^{3/4} do
12         for j ← 0 to n − 1 step n^{3/4} do
13             block ← M[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1]
14             M'[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1] ← sort(block, "ASC")
15         end for
16     end for
17     for j ← 0 to n − 1 do
18         M'[:, j] ← sort(M'[:, j], "ASC")
19     end for
20     for k ← 0 to n^{1/4} − 1 do
21         j ← k (n^{1/4} + 1)
22         verticalSlice ← M'[:, j ... j + n^{1/4}]
23         M'[:, j ... j + n^{1/4}] ← sort(verticalSlice, "ASC")
24     end for
25     for i ← 0 to n − 1 do
26         if i mod 2 = 0 then
27             M'[i, :] ← sort(M'[i, :], "ASC")
28         else
29             M'[i, :] ← sort(M'[i, :], "DESC")
30         end if
31     end for
32     M' ← oets(M')
33     return M'
34 end function
```

After sort the blocks (line 16)

$$\begin{bmatrix} 0 & 4 & 1 & 5 \\ 7 & 9 & 8 & 12 \\ 2 & 6 & 3 & 11 \\ 10 & 14 & 13 & 15 \end{bmatrix}$$

3n-Sort Example (5)

Algorithm 55 3n-Sort Algorithm

Input:
$M \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$M' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function THREENSORT(M)
2      for i ← 0 to n − 1 step n^{3/4} do
3          for j ← 0 to n − 1 step n^{3/4} do
4              block ← M[i...i + n^{3/4} − 1, j...j + n^{3/4} − 1]
5              M'[i...i + n^{3/4} − 1, j...j + n^{3/4} − 1] ← SORT(block, "ASC")
6          end for
7      end for
8      for i ← 0 to n − 1 do
9          M' ← KWAYUNSHUFFLE(M[i, :], n^{1/4})
10     end for
11     for i ← 0 to n − 1 step n^{3/4} do
12         for j ← 0 to n − 1 step n^{3/4} do
13             block ← M[i...i + n^{3/4} − 1, j...j + n^{3/4} − 1]
14             M'[i...i + n^{3/4} − 1, j...j + n^{3/4} − 1] ← SORT(block, "ASC")
15         end for
16     end for
17     for j ← 0 to n − 1 do
18         M'[:, j] ← SORT(M'[:, j], "ASC")
19     end for
20     for k ← 0 to n^{1/4} − 1 do
21         j ← k (n^{1/4} + 1)
22         verticalSlice ← M'[:, j...j + n^{1/4}]
23         M'[:, j...j + n^{1/4}] ← SORT(verticalSlice, "ASC")
24     end for
25     for i ← 0 to n − 1 do
26         if i mod 2 = 0 then
27             M'[i, :] ← SORT(M'[i, :], "ASC")
28         else
29             M'[i, :] ← SORT(M'[i, :], "DESC")
30         end if
31     end for
32     M' ← OETS(M')
33     return M'
34 end function
```

After sort columns (line 19)

$$\begin{bmatrix} 0 & 4 & 1 & 5 \\ 2 & 6 & 3 & 11 \\ 7 & 9 & 8 & 12 \\ 10 & 14 & 13 & 15 \end{bmatrix}$$

3n-Sort Example (6)

Algorithm 56 3n-Sort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

1 **function** THREENSORT(M)
2 **for** $i \leftarrow 0$ **to** $n-1$ **step** $n^{3/4}$ **do**
3 **for** $j \leftarrow 0$ **to** $n-1$ **step** $n^{3/4}$ **do**
4 $\text{block} \leftarrow \mathrm{M}[i \dots i + n^{3/4} - 1,\ j \dots j + n^{3/4} - 1]$
5 $\mathrm{M}'[i \dots i + n^{3/4} - 1,\ j \dots j + n^{3/4} - 1] \leftarrow \textsc{sort}(\text{block}, \text{"ASC"})$
6 **end for**
7 **end for**
8 **for** $i \leftarrow 0$ **to** $n-1$ **do**
9 $\mathrm{M}' \leftarrow \textsc{kWayUnshuffle}(\mathrm{M}[i, :], n^{\frac{1}{4}})$
10 **end for**
11 **for** $i \leftarrow 0$ **to** $n-1$ **step** $n^{3/4}$ **do**
12 **for** $j \leftarrow 0$ **to** $n-1$ **step** $n^{3/4}$ **do**
13 $\text{block} \leftarrow \mathrm{M}[i \dots i + n^{3/4} - 1,\ j \dots j + n^{3/4} - 1]$
14 $\mathrm{M}'[i \dots i + n^{3/4} - 1,\ j \dots j + n^{3/4} - 1] \leftarrow \textsc{sort}(\text{block}, \text{"ASC"})$
15 **end for**
16 **end for**
17 **for** $j \leftarrow 0$ **to** $n-1$ **do**
18 $\mathrm{M}'[:, j] \leftarrow \textsc{sort}(\mathrm{M}'[:, j], \text{"ASC"})$
19 **end for**
20 **for** $k \leftarrow 0$ **to** $n^{\frac{1}{4}} - 1$ **do**
21 $j \leftarrow k\left(n^{\frac{1}{4}} + 1\right)$
22 $\text{verticalSlice} \leftarrow \mathrm{M}'[:, j \dots j + n^{\frac{1}{4}}]$
23 $\mathrm{M}'[:, j \dots j + n^{\frac{1}{4}}] \leftarrow \textsc{sort}(\text{verticalSlice}, \text{"ASC"})$
24 **end for**
25 **for** $i \leftarrow 0$ **to** $n-1$ **do**
26 **if** $i \mod 2 = 0$ **then**
27 $\mathrm{M}'[i, :] \leftarrow \textsc{sort}(\mathrm{M}'[i, :], \text{"ASC"})$
28 **else**
29 $\mathrm{M}'[i, :] \leftarrow \textsc{sort}(\mathrm{M}'[i, :], \text{"DESC"})$
30 **end if**
31 **end for**
32 $\mathrm{M}' \leftarrow \textsc{oets}(\mathrm{M}')$
33 **return** M'
34 **end function**

After sort vertical slices (line 24)

$$\begin{bmatrix} 0 & 2 & 1 & 3 \\ 4 & 6 & 5 & 8 \\ 7 & 9 & 11 & 12 \\ 10 & 14 & 13 & 15 \end{bmatrix}$$

3n-Sort Example (7)

Algorithm 57 3n-Sort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function THREENSORT(M)
2      for i ← 0 to n − 1 step n^{3/4} do
3          for j ← 0 to n − 1 step n^{3/4} do
4              block ← M[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1]
5              M′[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1] ← SORT(block, "ASC")
6          end for
7      end for
8      for i ← 0 to n − 1 do
9          M′ ← KWAYUNSHUFFLE(M[i, :], n^{1/4})
10     end for
11     for i ← 0 to n − 1 step n^{3/4} do
12         for j ← 0 to n − 1 step n^{3/4} do
13             block ← M[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1]
14             M′[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1] ← SORT(block, "ASC")
15         end for
16     end for
17     for j ← 0 to n − 1 do
18         M′[:, j] ← SORT(M′[:, j], "ASC")
19     end for
20     for k ← 0 to n^{1/4} − 1 do
21         j ← k (n^{1/4} + 1)
22         verticalSlice ← M′[:, j ... j + n^{1/4}]
23         M′[:, j ... j + n^{1/4}] ← SORT(verticalSlice, "ASC")
24     end for
25     for i ← 0 to n − 1 do
26         if i mod 2 = 0 then
27             M′[i, :] ← SORT(M′[i, :], "ASC")
28         else
29             M′[i, :] ← SORT(M′[i, :], "DESC")
30         end if
31     end for
32     M′ ← OETS(M′)
33     return M′
34 end function
```

After sort rows (line 31)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 8 & 6 & 5 & 4 \\ 7 & 9 & 11 & 12 \\ 15 & 14 & 13 & 10 \end{bmatrix}$$

3n-Sort Example (8)

Algorithm 58 3n-Sort Algorithm

Input:
$M \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$M' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function THREENSORT(M)
2      for i ← 0 to n − 1 step n^{3/4} do
3          for j ← 0 to n − 1 step n^{3/4} do
4              block ← M[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1]
5              M'[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1] ← SORT(block, "ASC")
6          end for
7      end for
8      for i ← 0 to n − 1 do
9          M' ← KWAYUNSHUFFLE(M[i, :], n^{1/4})
10     end for
11     for i ← 0 to n − 1 step n^{3/4} do
12         for j ← 0 to n − 1 step n^{3/4} do
13             block ← M[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1]
14             M'[i ... i + n^{3/4} − 1, j ... j + n^{3/4} − 1] ← SORT(block, "ASC")
15         end for
16     end for
17     for j ← 0 to n − 1 do
18         M'[:, j] ← SORT(M'[:, j], "ASC")
19     end for
20     for k ← 0 to n^{1/4} − 1 do
21         j ← k (n^{1/4} + 1)
22         verticalSlice ← M'[:, j ... j + n^{1/4}]
23         M'[:, j ... j + n^{1/4}] ← SORT(verticalSlice, "ASC")
24     end for
25     for i ← 0 to n − 1 do
26         if i mod 2 = 0 then
27             M'[i, :] ← SORT(M'[i, :], "ASC")
28         else
29             M'[i, :] ← SORT(M'[i, :], "DESC")
30         end if
31     end for
32     M' ← OETS(M')
33     return M'
34 end function
```

After $n^{\frac{3}{4}}$ steps of OETS (line 32)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. Conclusion

2D Odd-Even Transposition Sort Introduction

The 2D odd-even transposition sort is the simplest algorithm for sorting two-dimensional arrays, but it is very slow, time complexity $\Theta\left(n^2\right)$.
The idea of odd-even transposition sort is generalized to two-dimensional arrays in a straightforward way. The elements are not just compared with their right and left neighbours, but also with their upper and lower neighbours.

The sorting direction in the array is snake-like.

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. Conclusion

2D Odd-Even Transposition Sort

Algorithm 59 2D Odd-Even Transposition Sort Algorithm

Input:

$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix

Output:

$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
function twoDOddEvenTranspositionSort(M)
    while swapTakePlace do
        swapTakePlace ← false
        for i ← 0 to n − 1 do
            for j ← 1 to n − 2 step 2 do
                if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
                    swap M[i][j] and M[i][j + 1]
                    swapTakePlace ← true
                else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
                    swap M[i][j] and M[i][j + 1]
                    swapTakePlace ← true
                end if
            end for
        end for
        for i ← 0 to n − 1 do
            for j ← 0 to n − 2 step 2 do
                if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
                    swap M[i][j] and M[i][j + 1]
                    swapTakePlace ← true
                else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
                    swap M[i][j] and M[i][j + 1]
                    swapTakePlace ← true
                end if
            end for
        end for
        for j ← 0 to n − 1 do
            for i ← 1 to n − 2 step 2 do
                if M[i][j] > M[i + 1][j] then
                    swap M[i][j] and M[i + 1][j]
                    swapTakePlace ← true
                end if
            end for
        end for
        for j ← 0 to n − 1 do
            for i ← 0 to n − 2 step 2 do
                if M[i][j] > M[i + 1][j] then
                    swap M[i][j] and M[i + 1][j]
                    swapTakePlace ← true
                end if
            end for
        end for
    end while
    return M′
end function
```

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. Conclusion

2D Odd-Even Transposition Sort Correctness

In two-dimensional odd-even transposition sort, the elements are not just compared with their right and left neighbours, but also with their upper and lower neighbours.

2D Odd-Even Transposition Sort Correctness (2)

The four steps of two-dimensional odd-even transposition sort are repeated in round-robin order.

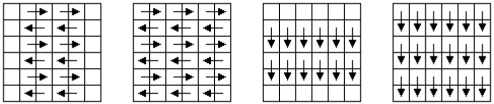


Figure: 2D OETS Steps

2D Odd-Even Transposition Sort Correctness (3)

In each step, comparisons occur between pairs of adjacent cells:

- *rows*: comparisons between horizontal neighbours;
- *columns*: comparisons between vertical neighbours.

If two cells have the values 1 and 0, swapping puts them in the correct position according to the direction of comparison.

2D Odd-Even Transposition Sort Correctness (4)

- Rows are neatly “cleaned” with row steps: the 1s move toward the final direction of the snake;
- columns help propagate the 1s downward, to the next rows in the snake.

2D Odd-Even Transposition Sort Correctness (5)

After a sufficient number of iterations, the 1s “slide” like grains of sand along the surface of the array following the snake path. Eventually, all the 1’s will be at the bottommost positions in the snake-like linear sequence.

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- **Analysis**
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

2D Odd-Even Transposition Sort Analysis

The 2D odd-even transposition sort is the simplest algorithm for sorting two-dimensional arrays, but it is very slow, time complexity $\Theta\left(n^2\right)$.

2D Odd-Even Transposition Sort Analysis (2)

If we ask what the constants of $\Theta(n^2)$ are, here is what we can say

$$T(n) = c \cdot n^2$$

where c is the constant factor depending on

- how many iterations are needed to fully sort in the worst case, but this is not fixed and depends on input;
- each iteration consist of
  - an odd step on rows;
  - an even step on rows;
  - an odd step on columns;
  - an even step on columns;

If we assume k full rounds are needed, then:

$$T(n) = 4k \cdot n^2$$

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. Conclusion

2D Odd-Even Transposition Sort Example

Algorithm 60 2D Odd-Even Transposition Sort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n\times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n\times n}$: snake-sorted matrix

```
function TWODODDEVENTRANSPOSITIONSORT(M)
    while swapTakePlace do
        swapTakePlace ← false
        for i ← 0 to n − 1 do
            for j ← 1 to n − 2 step 2 do
                if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
                    swap M[i][j] and M[i][j + 1]
                    swapTakePlace ← true
                else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
                    swap M[i][j] and M[i][j + 1]
                    swapTakePlace ← true
                end if
            end for
        end for
        for i ← 0 to n − 1 do
            for j ← 0 to n − 2 step 2 do
                if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
                    swap M[i][j] and M[i][j + 1]
                    swapTakePlace ← true
                else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
                    swap M[i][j] and M[i][j + 1]
                    swapTakePlace ← true
                end if
            end for
        end for
        for j ← 0 to n − 1 do
            for i ← 1 to n − 2 step 2 do
                if M[i][j] > M[i + 1][j] then
                    swap M[i][j] and M[i + 1][j]
                    swapTakePlace ← true
                end if
            end for
        end for
        for j ← 0 to n − 1 do
            for i ← 0 to n − 2 step 2 do
                if M[i][j] > M[i + 1][j] then
                    swap M[i][j] and M[i + 1][j]
                    swapTakePlace ← true
                end if
            end for
        end for
    end while
    return M′
end function
```

Initial situation

$$\begin{bmatrix} 0 & 1 & 7 & 12 \\ 4 & 5 & 9 & 8 \\ 13 & 6 & 15 & 14 \\ 3 & 2 & 11 & 10 \end{bmatrix}$$

2D Odd-Even Transposition Sort Example (2)

Algorithm 61 2D Odd-Even Transposition Sort Algorithm

Input:
$M \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$M' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
 1 function twoDOddEvenTranspositionSort(M)
 2     while swapTakePlace do
 3         swapTakePlace ← false
 4         for i ← 0 to n − 1 do
 5             for j ← 1 to n − 2 step 2 do
 6                 if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
 7                     swap M[i][j] and M[i][j + 1]
 8                     swapTakePlace ← true
 9                 else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
10                     swap M[i][j] and M[i][j + 1]
11                     swapTakePlace ← true
12                 end if
13             end for
14         end for
15         for i ← 0 to n − 1 do
16             for j ← 0 to n − 2 step 2 do
17                 if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
18                     swap M[i][j] and M[i][j + 1]
19                     swapTakePlace ← true
20                 else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
21                     swap M[i][j] and M[i][j + 1]
22                     swapTakePlace ← true
23                 end if
24             end for
25         end for
26         for j ← 0 to n − 1 do
27             for i ← 1 to n − 2 step 2 do
28                 if M[i][j] > M[i + 1][j] then
29                     swap M[i][j] and M[i + 1][j]
30                     swapTakePlace ← true
31                 end if
32             end for
33         end for
34         for j ← 0 to n − 1 do
35             for i ← 0 to n − 2 step 2 do
36                 if M[i][j] > M[i + 1][j] then
37                     swap M[i][j] and M[i + 1][j]
38                     swapTakePlace ← true
39                 end if
40             end for
41         end for
42     end while
43     return M′
44 end function
```

After odd oets step to the rows (line 14)

$$\begin{bmatrix} 0 & 1 & 7 & 12 \\ 4 & 9 & 5 & 8 \\ 13 & 6 & 15 & 14 \\ 3 & 11 & 2 & 10 \end{bmatrix}$$

After even oets step to the rows (line 25)

$$\begin{bmatrix} 0 & 1 & 7 & 12 \\ 9 & 4 & 8 & 5 \\ 6 & 13 & 14 & 15 \\ 11 & 3 & 10 & 2 \end{bmatrix}$$

After odd oets step to the columns (line 33)

$$\begin{bmatrix} 0 & 1 & 7 & 12 \\ 6 & 4 & 8 & 5 \\ 9 & 13 & 14 & 15 \\ 11 & 3 & 10 & 2 \end{bmatrix}$$

After even oets step to the columns (line 41)

$$\begin{bmatrix} 0 & 1 & 7 & 5 \\ 6 & 4 & 8 & 12 \\ 9 & 3 & 10 & 2 \\ 11 & 13 & 14 & 15 \end{bmatrix}$$

2D Odd-Even Transposition Sort Example (3)

Algorithm 62 2D Odd-Even Transposition Sort Algorithm

Input:
$M \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$M' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
 1 function TWODODDEVENTRANSPOSITIONSORT(M)
 2     while swapTakePlace do
 3         swapTakePlace ← false
 4         for i ← 0 to n − 1 do
 5             for j ← 1 to n − 2 step 2 do
 6                 if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
 7                     swap M[i][j] and M[i][j + 1]
 8                     swapTakePlace ← true
 9                 else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
10                     swap M[i][j] and M[i][j + 1]
11                     swapTakePlace ← true
12                 end if
13             end for
14         end for
15         for i ← 0 to n − 1 do
16             for j ← 0 to n − 2 step 2 do
17                 if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
18                     swap M[i][j] and M[i][j + 1]
19                     swapTakePlace ← true
20                 else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
21                     swap M[i][j] and M[i][j + 1]
22                     swapTakePlace ← true
23                 end if
24             end for
25         end for
26         for j ← 0 to n − 1 do
27             for i ← 1 to n − 2 step 2 do
28                 if M[i][j] > M[i + 1][j] then
29                     swap M[i][j] and M[i + 1][j]
30                     swapTakePlace ← true
31                 end if
32             end for
33         end for
34         for j ← 0 to n − 1 do
35             for i ← 0 to n − 2 step 2 do
36                 if M[i][j] > M[i + 1][j] then
37                     swap M[i][j] and M[i + 1][j]
38                     swapTakePlace ← true
39                 end if
40             end for
41         end for
42     end while
43     return M′
44 end function
```

After odd oets step to the rows (line 14)

$$\begin{bmatrix} 0 & 1 & 7 & 5 \\ 6 & 8 & 4 & 12 \\ 9 & 3 & 10 & 2 \\ 11 & 14 & 13 & 15 \end{bmatrix}$$

After even oets step to the rows (line 25)

$$\begin{bmatrix} 0 & 1 & 5 & 7 \\ 8 & 6 & 12 & 4 \\ 3 & 9 & 2 & 10 \\ 14 & 11 & 15 & 13 \end{bmatrix}$$

After odd oets step to the columns (line 33)

$$\begin{bmatrix} 0 & 1 & 5 & 7 \\ 3 & 6 & 2 & 4 \\ 8 & 9 & 12 & 10 \\ 14 & 11 & 15 & 13 \end{bmatrix}$$

After even oets step to the columns (line 41)

$$\begin{bmatrix} 0 & 1 & 2 & 4 \\ 3 & 6 & 5 & 7 \\ 8 & 9 & 12 & 10 \\ 14 & 11 & 15 & 13 \end{bmatrix}$$

2D Odd-Even Transposition Sort Example (4)

Algorithm 63 2D Odd-Even Transposition Sort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n\times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n\times n}$: snake-sorted matrix

```
1  function TWODODDEVENTRANSPOSITIONSORT(M)
2      while swapTakePlace do
3          swapTakePlace ← false
4          for i ← 0 to n − 1 do
5              for j ← 1 to n − 2 step 2 do
6                  if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
7                      swap M[i][j] and M[i][j + 1]
8                      swapTakePlace ← true
9                  else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
10                     swap M[i][j] and M[i][j + 1]
11                     swapTakePlace ← true
12                 end if
13             end for
14         end for
15         for i ← 0 to n − 1 do
16             for j ← 0 to n − 2 step 2 do
17                 if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
18                     swap M[i][j] and M[i][j + 1]
19                     swapTakePlace ← true
20                 else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
21                     swap M[i][j] and M[i][j + 1]
22                     swapTakePlace ← true
23                 end if
24             end for
25         end for
26         for j ← 0 to n − 1 do
27             for i ← 1 to n − 2 step 2 do
28                 if M[i][j] > M[i + 1][j] then
29                     swap M[i][j] and M[i + 1][j]
30                     swapTakePlace ← true
31                 end if
32             end for
33         end for
34         for j ← 0 to n − 1 do
35             for i ← 0 to n − 2 step 2 do
36                 if M[i][j] > M[i + 1][j] then
37                     swap M[i][j] and M[i + 1][j]
38                     swapTakePlace ← true
39                 end if
40             end for
41         end for
42     end while
43     return M′
44 end function
```

After odd oets step to the rows (line 14)

$$\begin{bmatrix} 0 & 1 & 2 & 4 \\ 3 & 6 & 5 & 7 \\ 8 & 9 & 12 & 10 \\ 14 & 15 & 11 & 13 \end{bmatrix}$$

After even oets step to the rows (line 25)

$$\begin{bmatrix} 0 & 1 & 2 & 4 \\ 6 & 3 & 7 & 5 \\ 8 & 9 & 10 & 12 \\ 15 & 14 & 13 & 11 \end{bmatrix}$$

After odd oets step to the columns (line 33)

$$\begin{bmatrix} 0 & 1 & 2 & 4 \\ 6 & 3 & 7 & 5 \\ 8 & 9 & 10 & 12 \\ 15 & 14 & 13 & 11 \end{bmatrix}$$

After even oets step to the columns (line 41)

$$\begin{bmatrix} 0 & 1 & 2 & 4 \\ 6 & 3 & 7 & 5 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

2D Odd-Even Transposition Sort Example (5)

Algorithm 64 2D Odd-Even Transposition Sort Algorithm

Input:
$M \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$M' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
 1 function twoDOddEvenTranspositionSort(M)
 2     while swapTakePlace do
 3         swapTakePlace ← false
 4         for i ← 0 to n − 1 do
 5             for j ← 1 to n − 2 step 2 do
 6                 if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
 7                     swap M[i][j] and M[i][j + 1]
 8                     swapTakePlace ← true
 9                 else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
10                     swap M[i][j] and M[i][j + 1]
11                     swapTakePlace ← true
12                 end if
13             end for
14         end for
15         for i ← 0 to n − 1 do
16             for j ← 0 to n − 2 step 2 do
17                 if i mod 2 = 0 ∧ M[i][j] > M[i][j + 1] then
18                     swap M[i][j] and M[i][j + 1]
19                     swapTakePlace ← true
20                 else if i mod 2 = 1 ∧ M[i][j + 1] > M[i][j] then
21                     swap M[i][j] and M[i][j + 1]
22                     swapTakePlace ← true
23                 end if
24             end for
25         end for
26         for j ← 0 to n − 1 do
27             for i ← 1 to n − 2 step 2 do
28                 if M[i][j] > M[i + 1][j] then
29                     swap M[i][j] and M[i + 1][j]
30                     swapTakePlace ← true
31                 end if
32             end for
33         end for
34         for j ← 0 to n − 1 do
35             for i ← 0 to n − 2 step 2 do
36                 if M[i][j] > M[i + 1][j] then
37                     swap M[i][j] and M[i + 1][j]
38                     swapTakePlace ← true
39                 end if
40             end for
41         end for
42     end while
43     return M′
44 end function
```

After odd oets step to the rows (line 14)

$$\begin{bmatrix} 0 & 1 & 2 & 4 \\ 6 & 7 & 3 & 5 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

After even oets step to the rows (line 25)

$$\begin{bmatrix} 0 & 1 & 2 & 4 \\ 7 & 6 & 5 & 3 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

After odd oets step to the columns (line 33)

$$\begin{bmatrix} 0 & 1 & 2 & 4 \\ 7 & 6 & 5 & 3 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

After even oets step to the columns (line 41)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 **Shearsort**
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

Shearsort Introduction

The shearsort is a very simple algorithm for sorting two-dimensional arrays.
It just sorts the rows and the columns of the array in turn.

The sorting direction in the array is snake-like.

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. Conclusion

Shearsort

Algorithm 65 Shearsort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
function SHEARSORT(M)
    for k ← 0 to log n − 1 do
        for i ← 0 to n − 1 do
            M′[i, :] ← SORT(M[i, :], "ASC")
        end for
        for j ← 0 to n − 1 do
            M′[:, j] ← SORT(M′[:, j], "ASC")
        end for
        k ← k + 1
    end for
    for i ← 0 to n − 1 do
        M′[i, :] ← SORT(M′[i, :], "ASC")
    end for
end function
```

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

Shearsort Correctness

After sorting the rows in the first step, there are $\frac{n}{2}$ rows that are sorted from left to right and $\frac{n}{2}$ rows that are sorted from right to left. When sorting the columns, every two of these rows are combined to at least one clean row.

After the first iteration the array consist of some clean 0-rows, some clean 1-rows and an unsorted zone in between consisting of at most $\frac{n}{2}$ rows.

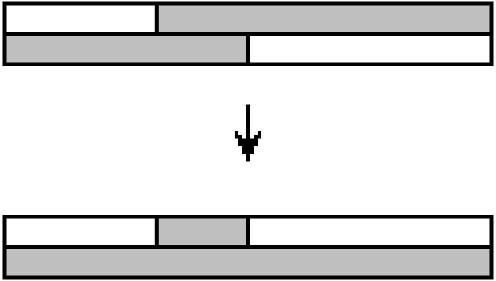


Figure: First Iteration

Shearsort Correctness (2)

In the second iteration, every two rows of the unsorted zone are again combined to at least one clean row. Thus, after the second iteration the unsorted zone consists of at most $\frac{n}{4}$ rows, and so on.

Shearsort Correctness (3)

In the second iteration, every two rows of the unsorted zone are again combined to at least one clean row. Thus, after the second iteration the unsorted zone consists of at most $\frac{n}{4}$ rows, and so on.

After $\log n$ steps, there is at most one dirty row. In the additional last step this row is sorted, and the whole array is sorted.

1 Introduction
2 LS3-Sort
- Algorithms
- Correctness
- Analysis
- Example

3 4-Way Mergesort
- Algorithms
- Correctness
- Analysis
- Example

4 Rotatesort
- Algorithms
- Correctness
- Analysis
- Example

5 3n-Sort of Schnorr and Shamir
- Algorithm
- Correctness
- Analysis
- Example

6 2D Odd-Even Transposition Sort
- Algorithm
- Correctness
- Analysis
- Example

7 Shearsort
- Algorithm
- Correctness
- Analysis
- Example

8 Conclusion

Shearsort Analysis

Sort a n-length rows takes n steps. The algorithm performs $\log n$ iteration, so it requires $n \log n$ steps for sorting the rows.

Shearsort Analysis (2)

Sort a n-length rows takes n steps. The algorithm performs $\log n$ iteration, so it requires $n \log n$ steps for sorting the rows.

In each iteration, the height of the unsorted zone decreases by a factor of 2.
This means that the columns contain an unsorted zone of decreasing length. So, the number of steps to sort the column is

$$n + \frac{n}{2} + \frac{n}{4} + \ldots + 2 = 2n - 2$$

Shearsort Analysis (3)

Sort a n-length rows takes n steps. The algorithm performs $\log n$ iteration, so it requires $n \log n$ steps for sorting the rows.

In each iteration, the height of the unsorted zone decreases by a factor of 2.
This means that the columns contain an unsorted zone of decreasing length. So, the number of steps to sort the column is

$$n + \frac{n}{2} + \frac{n}{4} + \ldots + 2 = 2n - 2$$

In the additional last step this row is sorted, and the whole array is sorted.
So, the final complexity is $n \log n + 2n - 2 + n = n(\log n + 3) - 2$.

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. **Shearsort**
   - Algorithm
   - Correctness
   - Analysis
   - **Example**
8. Conclusion

Shearsort Example

Algorithm 66 Shearsort Algorithm

Input:
$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix
Output:
$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
function SHEARSORT(M)
    for k ← 0 to log n − 1 do
        for i ← 0 to n − 1 do
            M′[i, :] ← SORT(M[i, :], "ASC")
        end for
        for j ← 0 to n − 1 do
            M′[:, j] ← SORT(M′[:, j], "ASC")
        end for
        k ← k + 1
    end for
    for i ← 0 to n − 1 do
        M′[i, :] ← SORT(M′[i, :], "ASC")
    end for
end function
```

Initial situation

$$\begin{bmatrix} 0 & 1 & 7 & 12 \\ 4 & 5 & 9 & 8 \\ 13 & 6 & 15 & 14 \\ 3 & 2 & 11 & 10 \end{bmatrix}$$

Shearsort Example (2)

Algorithm 67 Shearsort Algorithm

Input:

$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix

Output:

$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

1 **function** $\textsc{Shearsort}(\mathrm{M})$
2   **for** $k \leftarrow 0$ **to** $\log n - 1$ **do**
3   **for** $i \leftarrow 0$ **to** $n - 1$ **do**
4             $\mathrm{M}'[i, :] \leftarrow \textsc{sort}(\mathrm{M}[i, :], \text{"ASC"})$
5   **end for**
6   **for** $j \leftarrow 0$ **to** $n - 1$ **do**
7             $\mathrm{M}'[:, j] \leftarrow \textsc{sort}(\mathrm{M}'[:, j], \text{"ASC"})$
8   **end for**
9         $k \leftarrow k + 1$
10   **end for**
11   **for** $i \leftarrow 0$ **to** $n - 1$ **do**
12         $\mathrm{M}'[i, :] \leftarrow \textsc{sort}(\mathrm{M}'[i, :], \text{"ASC"})$
13   **end for**
14 **end function**

After sort rows (line 5)

$$\begin{bmatrix} 0 & 1 & 7 & 12 \\ 9 & 8 & 5 & 4 \\ 6 & 13 & 14 & 15 \\ 11 & 10 & 3 & 2 \end{bmatrix}$$

After sort columns (line 10)

$$\begin{bmatrix} 0 & 1 & 3 & 2 \\ 6 & 8 & 5 & 4 \\ 9 & 10 & 7 & 12 \\ 11 & 13 & 14 & 15 \end{bmatrix}$$

Shearsort Example (3)

Algorithm 68 Shearsort Algorithm

Input:

$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix

Output:

$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
 1 function SHEARSORT(M)
 2     for k ← 0 to log n − 1 do
 3         for i ← 0 to n − 1 do
 4             M′[i, :] ← SORT(M[i, :], "ASC")
 5         end for
 6         for j ← 0 to n − 1 do
 7             M′[:, j] ← SORT(M′[:, j], "ASC")
 8         end for
 9         k ← k + 1
10     end for
11     for i ← 0 to n − 1 do
12         M′[i, :] ← SORT(M′[i, :], "ASC")
13     end for
14 end function
```

After sort rows (line 5)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 8 & 6 & 5 & 4 \\ 7 & 9 & 10 & 12 \\ 15 & 14 & 13 & 11 \end{bmatrix}$$

After sort columns (line 10)

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

Shearsort Example (4)

Algorithm 69 Shearsort Algorithm

Input:

$\mathrm{M} \in \mathbb{Z}^{n \times n}$: unsorted matrix

Output:

$\mathrm{M}' \in \mathbb{Z}^{n \times n}$: snake-sorted matrix

```
1  function SHEARSORT(M)
2      for k ← 0 to log n − 1 do
3          for i ← 0 to n − 1 do
4              M′[i, :] ← SORT(M[i, :], "ASC")
5          end for
6          for j ← 0 to n − 1 do
7              M′[:, j] ← SORT(M′[:, j], "ASC")
8          end for
9          k ← k + 1
10     end for
11     for i ← 0 to n − 1 do
12         M′[i, :] ← SORT(M′[i, :], "ASC")
13     end for
14 end function
```

After last sort rows (line 13

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 7 & 6 & 5 & 4 \\ 8 & 9 & 10 & 11 \\ 15 & 14 & 13 & 12 \end{bmatrix}$$

1. Introduction
2. LS3-Sort
   - Algorithms
   - Correctness
   - Analysis
   - Example
3. 4-Way Mergesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
4. Rotatesort
   - Algorithms
   - Correctness
   - Analysis
   - Example
5. 3n-Sort of Schnorr and Shamir
   - Algorithm
   - Correctness
   - Analysis
   - Example
6. 2D Odd-Even Transposition Sort
   - Algorithm
   - Correctness
   - Analysis
   - Example
7. Shearsort
   - Algorithm
   - Correctness
   - Analysis
   - Example
8. **Conclusion**

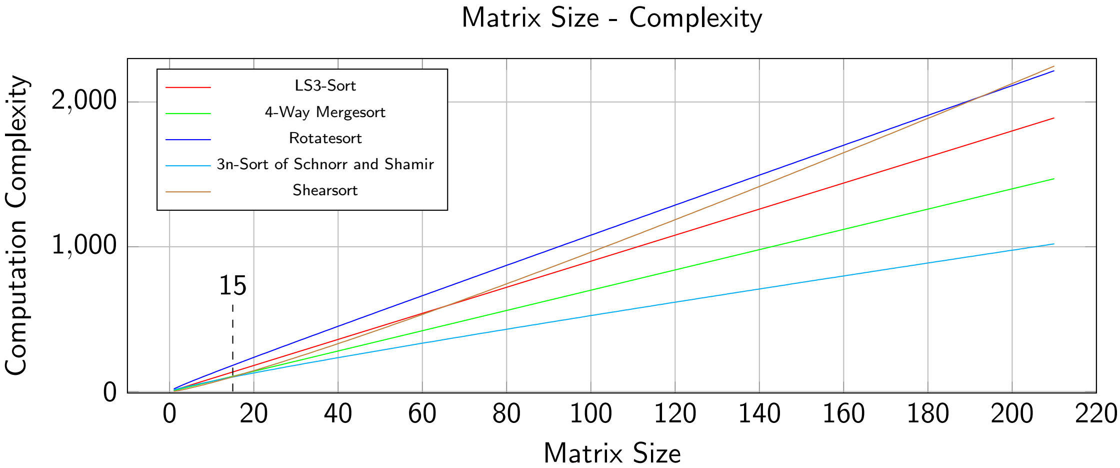
Summary of Complexity

Let us consider having a two-dimensional array $\mathrm{M} \in \mathbb{Z}^{n \times n}$

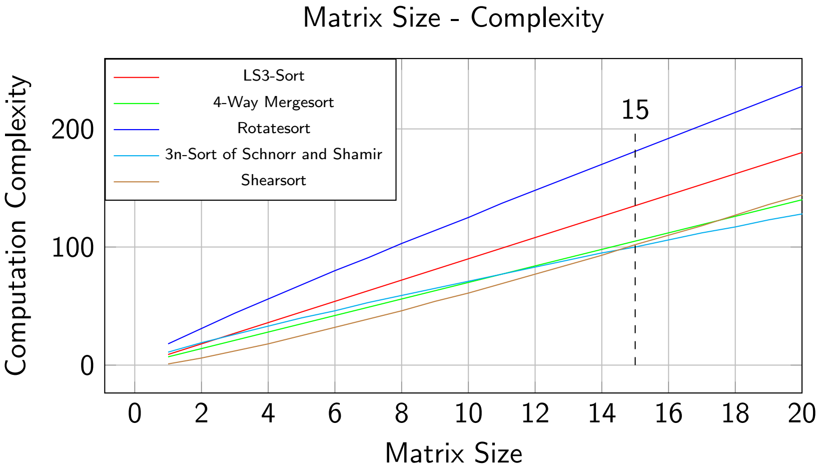
Sorting Network	Complexity
LS3-Sort	$\leq 9n$
4-Way Mergesort	$\leq 7n$
Rotatesort	$\leq 10n + 8\sqrt{n}$
3n-Sort of Schnorr and Shamir	$3n + 7n^{\frac{3}{4}} + n^{\frac{1}{4}}$
2D Odd-Even Transposition Sort	$\Theta\left(n^2\right)$
Shearsort	$n(\log n + 3) - 2$

Table: Summary of Sorting Network Complexity

Complexity Diagram



Complexity Diagram (2)



It can be deduced from the graph that for:

- $n < 15$ the best sorting network is **Shearsort**;
- $n \geq 15$ the best sorting network is **3n-Sort of Schnorr and Shamir**

Best Algorithm by Size

Which algorithm is better to use?

Shearsort	$n < 193$
Rotatesort	$n \geq 193$
Shearsort	$n < 66$
LS3-Sort	$n \geq 66$
Shearsort	$n < 18$
4-Way Mergesort	$n \geq 18$
Shearsort	$n < 15$
3n-Sort of Schnorr and Shamir	$n \geq 15$
4-Way Mergesort	$n < 12$
3n-Sort of Schnorr and Shamir	$n \geq 12$