Sorting on High Dimensional Reconfiguring Arrays

by

A. Schuster and G. Shemesh

Technical Report #744
June 1992

TECHNION – ISRAEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

HAIFA, ISRAEL
Sorting on high dimensional reconfiguring arrays

Assaf Schuster       Galit Shemesh

June 22, 1992

Abstract

We show how \( n^{k-1} \) items may be sorted in \( O(1)^k \) steps on a \( n \times n \times \cdots \times n \) (\( k \)-dimensional) reconfigurable cube. These upper bounds, which for constant \( k \)'s match corresponding Area - Volume lower bounds, extend a previous algorithm sorting \( n \) items in \( O(1) \) time on an \( n \times n \) dynamically reconfiguring mesh [BPRS91].

1 Introduction

One of the most important and frequently used operations is sorting. It was shown in [WCL90] and in [BS90] that sorting of \( n \) items can be done in constant time on an \( n \times n \times n \) reconfiguring 3-dimensional array. [WCL90] posed the question whether the Area-Time lower bound for sorting [Tho83, Lei84] could be matched, by reducing the number of required switches to \( n^2 \). A significant progress was achieved with a constant time sorting algorithm of \( n \) items on an \( n \times n \times \log n \) reconfiguring mesh [NMT91]. Finally (and independently), the question was positively answered with a constant time sorting algorithm of \( n \) items on a network construction with \( O(1) \) layers of size \( n \times n \) each. It was observed that only one of these layers was active at each step and so the algorithm could be emulated by an \( n \times n \) reconfiguring mesh in \( O(1) \) steps [NPS92]. Moreover, it was shown that the Area-Time lower bound (\( AT^2 \geq n^2 \) in the word model) is matched by the reconfiguring mesh for any \( T \) [PJ91].

Theorem 1.1 [NPS92] Sorting \( n \) items placed at the top row of an \( n \times n \) reconfigurable mesh can be done in constant time. The corresponding algorithm is called MESH-SORT.

In this work we show how for all \( k \geq 2 \) and \( n \), \( n^{k-1} \) items can be sorted in

\[ 31^{k-2}c_1 + \left( (31^{k-2} - 1)/30 \right)c_2 \]

steps on a \( n \times n \times \cdots \times n \) (\( k \)-dimensional) reconfigurable cube, where \( c_1 \) denotes the number of steps of the MESH-SORT algorithm and \( c_2 \) denotes the total number of steps required for routing certain permutations on sub-cubes (See sub-algorithms 1.4, 1.3, 1.2 of section 2 and sub-algorithms 2.4, 2.3, 2.2 of section 3). For constant (bounded) \( k \)'s these upper bounds match corresponding Volume - Time lower bounds in the word model, i.e

\[ V^{k-1}T^k \geq N^k \]
where $N$ is the size of the input (not to be confused with $n$, the number of coordinate values in each dimension of the cube).

This paper is organized as follows. In sub-section 1.1 we describe the reconfigurable model. In sub-section 1.2 we give notations and terminology used throughout the paper. In section 2 we give the sorting algorithm for the 3D reconfigurable cube. This is a special case of the general sorting algorithm in section 3. It is given separately (with no proof or analysis) as an easier-to-understand example. In section 3 we give the sorting algorithm for the kD reconfigurable cube followed by a run-time analysis and a proof of its validity.

1.1 The reconfigurable model

The processors residing at the nodes of a reconfigurable network perform the same program synchronously, taking local decisions and calculations according to the input and locally stored data. A single node of the network may consist of a computing unit, a small buffer and a switch with reconnection capability. The buffer holds either an input item or a partial result. In the sequel, we interchange the notions of switch, processor, cube node and a network node.

A single time step of an RN computation is composed of the following substeps.

Substep 1: The network selects a configuration $H$ of the buses, and reconfigures itself to $H$. This is done by local decisions taken at each switch individually. In each switch, the local decisions are taken according to the two facts: The switch has an I/O port for each neighbor. Connections within the switch are such that the number of ports connected together is two at the most, i.e a port can be connected to one other port or not connected at all.

Substep 2: One processor connected by a bus may transmit a message on the bus. It is called the speaker of the bus. The message length is bounded by the bandwidth parameter of the network. For our sorting algorithm, massages issued on a bus are items or of the form (item, id) where id is the first two coordinates values of a processor id.

Substep 3: Some of the processors connected by the bus read the message transmitted on the bus by the speaker. These processors are referred to as the readers of the bus.

Substep 4: A constant time local computation is taken by every processor. For example two input items may be compared.

At each time step, a bus may take one of the two states. Idle: no processor transmits, Speak: there is a single speaker.

An example of a reconfigurable network is the $n \times n$ reconfigurable mesh. This mesh consists of an $n \times n$ array of processors. Each processor has four I/O ports (L,R,U,D) for its Left, Right, Up and Down neighbors. There are ten possible internal connections within each switch: (L-R,U-D), (L-D,R-U), (L-U,R-D), (U-D), (L-R), (R-D), (L-U), (R-U), (L-D) and ($\phi$).
1.2 Higher dimensional arrays: notations and terminology

- A p-dimensional reconfigurable array \( \underbrace{n \times \cdots \times n}_p \) is called a p-cube. Whenever this simplifies notation, the p-cube is denoted \( n^p \).

- A p-sub-cube is a sub-cube of the p-cube. For example, the \( \sqrt{n} \times n^3 \) is a 4-sub-cube of the 4-cube, for which there are \( \sqrt{n} \) consequent values for the first (the x) coordinate.

- Forming a bus within a k-sub-cube along an \( x_i \) axis between two \( x_i \) borders \( x_i = a \) and \( x_i = b \):
  In a p-sub-cube with coordinates \( x_1, x_2, \ldots, x_p \) forming such a bus means that:
  
  Each sub-cube node \((j_1, j_2, \ldots, j_i, \ldots, j_p)\) with \( a < j_i < b \) is connected to the sub-cube nodes 
  \((j_1, j_2, \ldots, j_i+1, \ldots, j_p)\) and 
  \((j_1, j_2, \ldots, j_i-1, \ldots, j_p)\).

- \( x_1 x_2 \ldots x_p \) order on the items that are sorted in the nodes of a p-sub-cube:
  
  For each coordinate the default is: Smaller items are in sub-cube nodes with lower coordinate values (form 1). Changing the default is done as follows: \( a \) before the coordinate means smaller items are in sub-cube nodes with lower coordinate values (form 1) and \( b \) before the coordinate means larger items are in sub-cube nodes with lower coordinate values (form 2).
  More formally, let \( x_i \) denote the \( i^{th} \) coordinate, having \( n_i \) possible values. Within the sub-cube in which \( \forall j < i \ x_j \) has a fixed value, the \( x_1 x_2 \ldots x_p \) order has one of the following forms:

  Form 1: For any \( l, m \) such that \( 1 \leq l < m \leq n_i \), any node for which \( x_i = l \) contains an item smaller or equal to any item in a node for which \( x_i = m \).

  Form 2: For any \( l, m \) such that \( 1 \leq l < m \leq n_i \), any node for which \( x_i = l \) contains an item larger or equal to any item in a node for which \( x_i = m \).

- Operations of the form “compare the items placed at processor \( i \) and processor \( j \), place the smaller in processor \( i \) and the larger in processor \( j \)” will be called prespecified comparison-exchange operations.

- Operations which at a given time \( t \) and for any processor \( i \), processor \( i \) receives an item from processor \( j \) and sends an item to processor \( k \) (Independent of the items' values). will be called prespecified routes.

2 Constant time sorting on a 3-cube

In this section it is shown how sorting \( n^2 \) items placed one item in each processor at the \( x = 0 \) plane of a 3-cube with \( x, y, z \) coordinates can be done in \( 31c_1 + c_2 \) steps, where \( c_1 \) denotes the number
of steps of the MESH-SORT algorithm and \( c_2 \) denotes the total number of steps required for routing certain permutations on sub-cubes (See sub-algorithms 1.4, 1.3, 1.2 of this section).

Definitions:

- \( x \)-major order (column-major order):
  The \( xy \) order on the nodes of a 2-sub-cube.
- \( y \)-major order (row-major order):
  The \( yz \) order on the nodes of a 2-sub-cube.
- Block: A block is a 3-sub-cube of the size \( \sqrt{n} \times n \times n \). There are \( \sqrt{n} \) such blocks in the original cube.
- Sub-block: A sub-block is a 3-sub-cube of the size \( \sqrt{n} \times \sqrt{n} \times n \). There are \( \sqrt{n} \) such sub-blocks in each block.

In order to sort the \( n^2 \) input items in the 3-cube, we use one main algorithm, four sub-algorithms and two applications of the MESH-SORT: One is the MESH-SORT as described in [NPS92], the other one uses a \( \sqrt{n} \times \sqrt{n} \times n \) cube in order to sort \( n \) items given on a \( \sqrt{n} \times \sqrt{n} \) mesh in constant time. This is done by folding an \( n \times n \) mesh into a \( \sqrt{n} \times \sqrt{n} \times n \) cube and emulating MESH-SORT.

Execution begins in the main algorithm and uses the other algorithms as subroutines.

The main algorithm and its four sub-algorithms are presented in increasing order of importance: 1.4, 1.3, 1.2, 1.1, and 1.

Sub-algorithm 1.4: Permutations of items between a pair of sub-blocks \( k \) and \( k + 1 \) of a block \( j \) so that each processor in sub-block \( k \) receives an item from a processor in sub-block \( k + 1 \) and vice-versa. Due to the way sub-algorithm 1.1 sorts the sub-blocks \( k \) and \( k + 1 \) before sending them to sub-algorithm 1.4, sub-algorithm 1.4 switches items between the processor in sub-block \( k \) containing the \( i \)'th smallest item within the sub-block, and the processor in sub-block \( k + 1 \) containing the \( i \)'th largest item within the sub-block (\( 1 \leq i \leq n \)).

- Input: One item in each processor on the \( z = 0 \) plane of the \( k \) and \( k + 1 \) sub-blocks (for some \( k \) such that \( 1 \leq k \leq \sqrt{n} - 1 \)) of a block \( j \) (for some \( j \) such that \( 1 \leq j \leq \sqrt{n} \)).

- Output: A pair of items in each processor on the \( z = 0 \) plane of the sub-blocks \( k \) and \( k + 1 \): Processor \((i_1, i_2, 0)\) of sub-block \( k \) (It is processor \((j - 1)\sqrt{n} + i_1, (k - 1)\sqrt{n} + i_2, 0)\) of the original cube, for all \( 1 \leq i_1, i_2 \leq \sqrt{n} \) receives an item from processor \((i_1, \sqrt{n} - i_2 + 1, 0)\) of sub-block \( k + 1 \) and vice-versa. Finally, each processor contains both its input item and the item received during execution.

- Routing steps from processors in the \( k \)'th sub-block to processors in the \( k + 1 \)'th sub-block (Routing steps from processors in the \( k + 1 \)'th sub-block to processors in the \( k \)'th sub-block are done in a similar way):
Sub-algorithm 1.3: Permutations of items between a pair of blocks $k$ and $k+1$ so that each processor in block $k$ receives an item from a processor in block $k+1$ and vice-versa: Due to the way algorithm 1 sorts the blocks $k$ and $k+1$ before sending them to sub-algorithm 1.3, sub-algorithm 1.3 switches items between the processor in block $k$ containing the $i$'th smallest item within the block, and the processor in block $k+1$ containing the $i$'th largest item within the block ($1 \leq i \leq n^{\frac{3}{2}}$).

- **Input:** One item in each processor on the $z = 0$ plane of the $k$ and $k+1$ blocks (for some $k$ such that $1 \leq k \leq \sqrt{n} - 1$).

- **Output:** A pair of items in each processor on the $z = 0$ plane of the blocks $k$ and $k+1$: Processor $(i_1, i_2, 0)$ of block $k$ (It is processor $((k - 1)\sqrt{n} + i_1, i_2, 0)$ of the original cube, for all $1 \leq i_1 \leq \sqrt{n}$ and $1 \leq i_2 \leq n$), receives an item from processor $(2k\sqrt{n} - i_1 + 1, i_2, 0)$ of block $k+1$ and vice-versa. Finally, each processor contains both its input item and the item received during execution.

- **Routing steps from processors in the $k$'th block to processors in the $k+1$'th block** (Routings from processors in the $k+1$'th block to processors in the $k$'th block are done in a similar way):

  1. A bus is formed within sub-block $k$ along the $z$ axis between the two $z$ borders $z = 1$ and $z = \sqrt{n}$. Each processor on the $z = 0$ plane of the $k$'th sub-block, sends $v$ (the item) on the bus it is connected to.

  A processor in block $k$ with $z$ coordinate equal to $i_3$ so that $i_3 = \sqrt{n} - i_2 + 1$, that receives an item during this step, holds $v$.

  2. A bus is formed within the original cube along the $x$ axis between the two $x$ borders $x = (k - 1)\sqrt{n} + 1$ and $x = k\sqrt{n}$. Each processor in the $k$'th block with an item $v$ received during last step, sends $v$ on the bus it is connected to.
A processor in sub-block \( k + 1 \) with equal \( x \) and \( z \) coordinate values that receives an item during this step, holds \( v \).

3. A bus is formed within sub-block \( k + 1 \) along the \( z \) axis between the two \( z \) borders \( z = 1 \) and \( z = \sqrt{n} \). Each processor in the \( k + 1 \)'th sub-block with an item \( v \) received during last step, sends \( v \) on the bus it is connected to.
A processor in the \( z = 0 \) plane of block \( k + 1 \) that receives an item \( v \) during this step, keeps it as its second item (the first one is the input item).

**Sub-algorithm 1.2 Permutations of items within the \( k \)'th block from a row-major to a column-major order.**

- **Input:** One item in each processor on the \( z = 0 \) plane of the \( k \)'th block. Items are ordered in a row-major order (\( 1 \leq k \leq \sqrt{n} \)).
- **Output:** One item in each processor on the \( z = 0 \) plane of the \( k \)'th block. Items are ordered in a column-major order.
- **Route steps:**

1. The block is divided into \( n \) parts: each part has a distinct \( y \) value, is of the size \( \sqrt{n} \times 1 \times n \) and has \( \sqrt{n} \) items.
A bus is formed within the block along the \( z \) axis between the two \( z \) borders \( z = 1 \) and \( z = \sqrt{n} \). Each processor in the \( k \)'th block sends \((v,d)\) on the bus it is connected to. (Where \( v \) is the item and \( d \) is the \( x \) and \( y \) coordinates values of the destination).
A processor in the \( k \)'th block with equal \( y \) and \( z \) coordinate values, that receives an item during this step, holds \((v,d)\).

2. A bus is formed within the block along the \( y \) axis between the two \( y \) borders \( y = 1 \) and \( y = n \). Each processor in the \( k \)'th block with a value \((v,d)\) received during last step sends \((v,d)\) on the bus it is connected to.
A processor in the \( k \)'th block with a \( y \) coordinate value equal to the \( y \) coordinate value in \( d \) that receives an item during this step, holds \((v,d)\).

3. A bus is formed within the block along the \( z \) axis between the two \( z \) borders \( z = 1 \) and \( z = \sqrt{n} \). Each processor in the \( k \)'th block with a value \((v,d)\) received during last step, sends \((v,d)\) on the bus it is connected to.
A processor in the \( k \)'th block with an \( z \) coordinate value equal to the \( z \) coordinate value in \( d \) that receives an item during this step, holds \((v,d)\).

4. A bus is formed within the block along the \( z \) axis between the two \( z \) borders \( z = 1 \) and \( z = n \). Each processor in the \( k \)'th block with a value \((v,d)\) from last step, sends \( v \) on the bus it is connected to.
A processor on the \( z = 0 \) cube that receives an item \( v \) during this step, keeps it as its new item.
Sub-algorithm 1.1: Sorting $\sqrt{n} \times n$ items in a $\sqrt{n} \times n^2$ block in a row-major order.

- **Input:** $\sqrt{n} \times n$ items, one item in each processor on the $z = 0$ plane of the block.

- **Output:** All items in block sorted in a row-major order, one item in each processor on the $z = 0$ plane of the block (Here items are placed in the default raw-major order which is a $sY - sX$ order. Other row-major orders than the default are immediately derived).

- **Sorting steps:**

1. Sorting sub-blocks:
   Sort each sub-block separately in a row-major order. All items in each sub-block start and end on the $z = 0$ plane.
   (In this step we sort $n$ items placed at the $z = 0$ plane of a $\sqrt{n} \times \sqrt{n} \times n$ cube. Sorting a sub-block is done by an application of the MESH-SORT).

2. Sorting columns:
   Sort each column in the $z = 0$ plane on an $n \times n$ mesh. Column $i$ is sorted on the $z = i$ plane. The sorted items are placed in the $z = 0$ plane.
   (In this step we sort $n$ items placed at one row of a 2-cube. This sort is done by an application of the MESH-SORT).

3. Sorting pairs of sub-blocks in a row-major order:
   For each pair of sub-blocks do the following:
   (a) Sort each sub-block in a row-major order with odd numbered sub-blocks sorted in an $s$-row-$s$-column order and even numbered sub-blocks sorted in an $s$-row-$b$-column order.
      (Sorting a sub-block is done by an application of the MESH-SORT).
   (b) Route appropriate items between the sub-blocks: the $i$'th smallest item in the sub-block with lower $y$ coordinate values is switched with the $i$'th largest one in the other sub-block.
      (The route is performed by sub-algorithm 1.4).
   (c) Compare and change (A processor in the sub-block with lower $y$ coordinate values keeps the smaller item from the two it holds, a processor in the second sub-block keeps the larger item from the two it holds).
   (d) Sort each sub-block in a row-major order.
      (Sorting a sub-block is done by an application of the MESH-SORT).
4. Sorting consequent pairs of sub-blocks:
This step is executed the same as the preceding step with consequent numbered sub-blocks: Instead of the pairs (1,2),(3,4),... of step 4, now the pairs are (2,3),(4,5),..., only this time in sub-step a, odd numbered sub-blocks are sorted as the even numbered sub-blocks in step 3 and the even numbered sub-blocks as the odd numbered sub-blocks in step 3.

Algorithm 1: Sorting \(n^2\) items placed at the \(z = 0\) plane of a 3-cube in a column-major order:

- **Input:** \(n^2\) items, one item in each processor on the \(z = 0\) plane of the cube.
- **Output:** All items sorted in a column-major order on the \(z = 0\) plane of the cube.
- **Sorting steps:**

1. **Sorting blocks:**
   Sort each block separately in a row-major order. All the items in each block start and end in the \(z = 0\) plane.
   (Sorting a block is done by sub-algorithm 1.1).
2. **Route:**
   Route the items in each block from a row-major order to a column-major order. All the items in each block start and end in the \(z = 0\) plane.
   (The route is performed by sub-algorithm 1.2).
3. **Sorting rows:**
   Sort each row in the \(z = 0\) plane on an \(n \times 1 \times n\) cube. Row \(i\) is sorted on the \(y = i\) plane. The sorted items are placed in the \(z = 0\) plane smaller items in lower \(y\) coordinate values.
   (In this step we sort \(n\) items placed in a row in a 2-cube).
4. **Sorting pairs of blocks in a column-major order:**
   For each pair of blocks do the following:
   (a) Sort each block in a row-major order with odd numbered blocks sorted in an \(a\)-row- \(a\)-column order and even numbered blocks sorted in a \(b\)-row- \(a\)-column order.
   (Sorting a block is done by sub-algorithm 1.1).
   (b) Route appropriate items between the blocks: The \(i\)'th smallest item in the block with lower \(x\) coordinate values is switched with the \(i\)'th largest one in the other block.
   (The route is performed by sub-algorithm 1.3).
   (c) Compare and change (A processor in the sub-block with lower \(y\) coordinate values keeps the smaller item from the two it holds, a processor in the second sub-block keeps the larger item from the two it holds).
(d) Sort each block in a row-major order.
   (Sorting a block is done by sub-algorithm 1.1).
(e) Route items in each block to be sorted in a column-major order.
   (The route is performed by sub-algorithm 1.2).

5. Sorting consequent pairs of blocks:
   This step is executed the same as the preceding step with consequent numbered subblocks: Instead of the pairs (1,2),(3,4),... of step 4, now the pairs are (2,3),(4,5),... only this time in sub-step a, odd numbered blocks are sorted as the even numbered blocks in step 3 and the even numbered blocks as the odd numbered blocks in step 3.

3 Sorting on a k-cube in $O(1)^k$ steps

In this section it is shown how for any $k \geq 3$ and $n$, sorting $n^{k-1}$ items placed one item in each processor on the $w_{k-3} = 0$ sub-cube of a $k$-cube with $x,y,z,w_1,w_2,...,w_{k-3}$ coordinates, can be done in $31^{k-2}c_1 + (31^{k-2} - 1)/30)c_2$ steps, where $c_1$ denotes the number of steps of the MESH-SORT algorithm and $c_2$ denotes the total number of steps required for routing certain permutations on sub-cubes (See sub-algorithms 2.4, 2.3, 2.2 of this section).

Definitions:
- $x$-major order
  This is the order $xyzw_1...w_{k-4}$ on a $(k-1)$-sub-cube when the $y$ coordinate takes part or the order $zxw_1...w_{k-4}$ on a $(k-2)$-sub-cube otherwise.
- $y$-major order
  This is the order $yxzw_1...w_{k-4}$ on a $(k-1)$-sub-cube when the $x$ coordinate takes part or the order $yzw_1...w_{k-4}$ on a $(k-2)$-sub-cube otherwise.
- Block:
  A block is a $k$-sub-cube of the size $\sqrt{n} \times n^{k-1}$. There are $\sqrt{n}$ such blocks in the original cube.
- Sub-block:
  A sub-block is a $k$-sub-cube of the size $\sqrt{n} \times \sqrt{n} \times n^{k-2}$. There are $\sqrt{n}$ such sub-blocks in each block.

3.1 The algorithm

In order to sort the $n^{k-1}$ input items in the $k$-cube, we use one main algorithm, four sub-algorithms, one algorithm that uses a $\sqrt{n} \times \sqrt{n} \times n^{k-1}$ cube in order to sort $n^{k-1}$ items given on a $\sqrt{n} \times \sqrt{n} \times n^{k-2}$ sub-cube (This is done by folding a $k$-cube into a $\sqrt{n} \times \sqrt{n} \times n^{k-1}$ cube and emulating the main algorithm) and two applications of the MESH-SORT: One is the MESH-SORT as described in [NPS92], the other one uses a $\sqrt{n} \times \sqrt{n} \times n$ cube in order to sort $n$ items given on a $\sqrt{n} \times \sqrt{n}$ mesh in constant time (This is done by folding an $n \times n$ mesh into a $\sqrt{n} \times \sqrt{n} \times n$ cube and emulating MESH-SORT).
Execution begins in main algorithm and uses the other algorithms as subroutines.

The main algorithm and its four sub-algorithms are presented in increasing order of importance: 2.4, 2.3, 2.2, 2.1, and 2.

Sub-algorithm 2.4: Permutations of items between a pair of sub-blocks $k$ and $k+1$ of a block $j$ so that each processor in sub-block $k$ receives an item from a processor in sub-block $k+1$ and vice-versa. Due to the way sub-algorithm 2.1 sorts the sub-blocks $k$ and $k+1$ before sending them to sub-algorithm 2.4, sub-algorithm 2.4 switches items between the processor in sub-block $k$ containing the $i$'th smallest item within the sub-block, and the processor in sub-block $k+1$ containing the $i$'th largest item within the sub-block ($1 \leq i \leq n^{k-2}$).

- **Input**: One item in each processor on the $w_{k-3} = 0$ sub-cube of the $k$ and $k+1$ sub-blocks (for some $k$ such that $1 \leq k \leq \sqrt{n} - 1$) of a block $j$ (for some $j$ such that $1 \leq j \leq \sqrt{n}$).

- **Output**: A pair of items in each processor on the $w_{k-3} = 0$ sub-cube of the sub-blocks $k$ and $k+1$: Processor $(i_1, i_2, i_3, \ldots, i_{k-1}, 0)$ of sub-block $k$ (It is processor $(j-1)\sqrt{n}+i_1,(k-1)\sqrt{n}+i_2,i_3,\ldots,i_{k-1},0)$ of the original cube, for all $1 \leq i_1, i_2 \leq \sqrt{n}$ and $1 \leq i_3, \ldots, i_{k-1} \leq n$) receives an item from processor $(i_1, \sqrt{n} - i_2 + 1, i_3, \ldots, i_{k-1}, 0)$ of sub-block $k+1$ and vice-versa. Finally, each processor contains both its input item and the item received during execution.

- **Routing steps from processors in the $k$'th sub-block to processors in the $k+1$'th sub-block** (Routing steps from processors in the $k+1$'th sub-block to processors in the $k$'th sub-block are done in a similar way):

  1. A bus is formed within sub-block $k$ along the $w_{k-3}$ axis between the two $w_{k-3}$ borders $w_{k-3} = 1$ and $w_{k-3} = \sqrt{n}$. Each processor on the $w_{k-3} = 0$ sub-cube of the $k$'th sub-block, sends $v$ (the item) on the bus it is connected to.

     A processor in sub-block $k$ with $y$ and $w_{k-3}$ coordinates equal to $i_2$ and $i_k$ so that $i_k = \sqrt{n} - i_2 + 1$, that receives an item during this step, holds $v$.

  2. A bus is formed within block $j$ along the $y$ axis between the two $y$ borders $y = (k - 1)\sqrt{n} + 1$ and $y = k\sqrt{n}$. Each processor in the $k$'th sub-block with an item $v$ received during last step, sends $v$ on the bus it is connected to.

     A processor in sub-block $k+1$ with equal $y$ and $w_{k-3}$ coordinate values, that receives an item during this step, holds $v$.

  3. A bus is formed within sub-block $k+1$ along the $w_{k-3}$ axis between the two $w_{k-3}$ borders $w_{k-3} = 1$ and $w_{k-3} = \sqrt{n}$. Each processor in the $k+1$'th sub-block with an item $v$ received during last step, sends $v$ on the bus it is connected to.
A processor in the \( w_{k-3} = 0 \) sub-cube of sub-block \( k + 1 \) that receives an item \( v \) during this step, keeps it as its second item (the first one is the input item).

**Sub-algorithm 2.3 : Permutations of items between a pair of blocks \( k \) and \( k + 1 \) so that each processor in block \( k \) receives an item from a processor in block \( k + 1 \) and vice-versa:** Due to the way algorithm 2 sorts the blocks \( k \) and \( k + 1 \) before sending them to sub-algorithm 2.3, sub-algorithm 2.3 switches items between the processor in block \( k \) containing the \( i \)'th smallest item within the block, and the processor in block \( k + 1 \) containing the \( i \)'th largest item within the block (\( 1 \leq i \leq \sqrt{n}^{k-2} \))

- **Input:** One item in each processor on the \( w_{k-3} = 0 \) sub-cube of the \( k \) and \( k + 1 \) blocks (for some \( k \) such that \( 1 \leq k \leq \sqrt{n} - 1 \)).

- **Output:** A pair of items in each processor on the \( w_{k-3} = 0 \) sub-cube of the blocks \( k \) and \( k + 1 \): Processor \((i_1, i_2, \ldots, i_{k-1}, 0)\) of block \( k \) (It is processor \( ((k-1)\sqrt{n} + i_1, i_2, \ldots, i_{k-1}, 0) \) of the original cube, for all \( 1 \leq i_1 \leq \sqrt{n} \) and \( 1 \leq i_2, \ldots, i_{k-1} \leq n \)), receives an item from processor \((2k\sqrt{n} - i_1 + 1, i_2, \ldots, i_{k-1}, 0)\) of block \( k + 1 \) and vice-versa. Finally, each processor contains both its input item and the item received during execution.

- **Routing steps from processors in the \( k \)'th block to processors in the \( k + 1 \)'th block (Routing from processors in the \( k + 1 \)'th block to processors in the \( k \)'th block are done in a similar way):

  1. A bus is formed within sub-block \( k \) along the \( w_{k-3} \) axis between the two \( w_{k-3} \) borders \( w_{k-3} = 1 \) and \( w_{k-3} = \sqrt{n} \). Each processor on the \( w_{k-3} = 0 \) plane of the \( k \)'th block, sends \( v \) (the item) on the bus it is connected to.

     A processor in block \( k \) with \( x \) and \( w_{k-3} \) coordinate equal to \( i_1 \) and \( i_k \) so that \( i_k = \sqrt{n} - i_1 + 1 \), that receives an item during this step, holds \( v \).

  2. A bus is formed within the original cube along the \( x \) axis between the two \( x \) borders \( x = (k - 1)\sqrt{n} + 1 \) and \( x = k\sqrt{n} \). Each processor in the \( k \)'th block with an item \( v \) received during last step, sends \( v \) on the bus it is connected to.

     A processor in sub-block \( k + 1 \) with equal \( x \) and \( w_{k-3} \) coordinate values that receives an item during this step, holds \( v \).

  3. A bus is formed within sub-block \( k + 1 \) along the \( w_{k-3} \) axis between the two \( w_{k-3} \) borders \( w_{k-3} = 1 \) and \( w_{k-3} = \sqrt{n} \). Each processor in the \( k + 1 \)'th sub-block with an item \( v \) received during last step, sends \( v \) on the bus it is connected to.
A processor in the $w_{k-3} = 0$ sub-cube of block $k+1$ that receives an item $v$ during this step, keeps it as its second item (the first one is the input item).

Sub-algorithm 2.2 Permutations of items within the $k$th block from a $y$-major to an $z$-major order.

- Input: One item in each processor on the $w_{k-3} = 0$ cube of the $k$th block. Items are ordered in a $y$-major order ($1 \leq k \leq \sqrt{n}$).
- Output: One item in each processor on the $w_{k-3} = 0$ cube of the $k$th block. Items are ordered in an $z$-major order.
- Route steps:

1. The block is divided into $n$ parts: each part has a distinct $y$ value, is of the size $\sqrt{n} \times 1 \times n^{k-3}$ and has $\sqrt{n}n^{k-3}$ items.
   A bus is formed within the block along the $w_{k-3}$ axis between the two $w_{k-3}$ borders $w_{k-3} = 1$ and $w_{k-3} = n$. Each processor in the $k$th block sends $(v, d)$ on the bus in is connected to (Where $v$ is the item and $d$ is the the $z$ and $y$ coordinates values of the destination).
   A processor in the $k$th block with equal $y$ and $w_{k-3}$ coordinate values, that receives an item during this step, holds $(v, d)$.

2. A bus is formed within the block along the $y$ axis between the two $y$ borders $y = 1$ and $y = n$. Each processor in the $k$th block with a value $(v, d)$ received during last step sends $(v, d)$ on the bus it is connected to.
   A processor in the $k$th block with a $y$ coordinate value equal to the $y$ coordinate value in $d$ that receives an item during this step, holds $(v, d)$.

3. A bus is formed within the block along the $z$ axis between the two $z$ borders $z = 1$ and $z = \sqrt{n}$. Each processor in the $k$th block with a value $(v, d)$ received during last step, sends $(v, d)$ on the bus it is connected to.
   A processor in the $k$th block with an $z$ coordinate value equal to the $z$ coordinate value in $d$ that receives an item during this step, holds $(v, d)$.

4. A bus is formed within the block along the $w_{k-3}$ axis between the two $w_{k-3}$ borders $w_{k-3} = 1$ and $w_{k-3} = n$. Each processor in the $k$th block with a value $(v, d)$ from last step, sends $v$ on the bus it is connected to.
A processor on the $w_{k-3} = 0$ sub-cube that receives an item $v$ during this step, keeps it as its new item.

**Sub-algorithm 2.1**: Sorting $\sqrt{n} \times n^{k-2}$ items in a $\sqrt{n} \times n^{k-1}$ block in an $x$-major order.

- **Input**: $\sqrt{n} \times n^{k-2}$ items, one item in each processor on the $w_{k-3} = 0$ sub-cube of the block.
- **Output**: All items in block sorted in a $y$-major order, one item in each processor on the $w_{k-3} = 0$ sub-cube of the block (Here items are placed in the default raw-major order which is an $sY - sX - \ldots - sW_{k-4}$ order. Other row-major orders than the default are immediately derived).
- **Sorting steps**:
  1. **Sorting sub-blocks**:
     Sort each sub-block separately in a $y$-major order. All items in the sub-block start and end on the $w_{k-3} = 0$ sub-cube.
     (In this step $n^{k-2}$ items are sorted on a $\sqrt{n} \times \sqrt{n} \times n^{k-2}$ cube).
  2. **Sorting $z$ equal cubes**:
     Sort each $z$ equal cube in a $y$-major order. All items start and end at the $w_{k-3} = 0$ sub-cube.
     (In this step $n^{k-2}$ items are sorted on a $(k-1)$-cube).
  3. **Sorting pairs of sub-blocks in a $y$-major order**:
     For each pair of sub-blocks do the following:
     (a) Sort each sub-block in a $y$-major order with the odd numbered sub-blocks sorted as the default and the even numbered sub-blocks in a $sY - bX - bZ - \ldots - bW_{k-4}$ order.
        (This step is performed in a similar way to step 1. of this sub-algorithm).
     (b) Route appropriate items between the sub-blocks: The $i$'th smallest item in the sub-block with lower $y$ coordinate values is switched with the $i$'th largest one in the other sub-block.
        (The route is performed by sub-algorithm 2.4).
     (c) Compare and change (A processor in the sub-block with lower $y$ coordinate values keeps the smaller item from the two it holds, a processor in the second sub-block keeps the larger item from the two it holds).
     (d) Sort each sub-block in a $y$-major order.
        (This step is performed as step 1. of this sub-algorithm).
4. Sorting consequent pairs of sub-blocks:
This step is executed the same as the preceding step with consequent numbered sub-blocks: Instead of the pairs (1,2),(3,4),... of step 3, now the pairs are (2,3),(4,5),..., only this time in sub-step a, odd numbered sub-blocks are sorted as the even numbered sub-blocks in step 3 and the even numbered sub-blocks as the odd numbered sub-blocks in step 3.

algorithm 2 : Sorting $n^{k-1}$ items placed at the $w_{k-3} = 0$ sub-cube of an $n^k$ cube in an $x$-major order.

- Input: $n^{k-1}$ items, one item in each processor on the $w_{k-3} = 0$ sub-cube.
- Output: All items sorted in an $x$-order on the $w_{k-3} = 0$ sub-cube.
- Sorting steps:
  1. Sorting blocks:
     Sort each block separately in a $y$-major order. All items in the block start and end on the $w_{k-3} = 0$ sub-cube.
     (Sorting a block is performed by sub-algorithm 2.1).
  2. Route:
     Route the items in each block from a $y$-major order to an $x$-major order. All items in the block start and end on the $w_{k-3} = 0$ sub-cube.
     (The route is performed by sub-algorithm 2.2).
  3. Sorting $y$ equal cubes:
     Sort each $y$ equal cube in an $x$-major order. Here $\forall i \in \{1...n\}$, $n^{k-2}$ items placed at the $y = i, w_{k-3} = 0$ sub-cube, are sorted on the $y = i$ sub-cube of the size $n^{k-1}$, $\forall i \in \{1...n\}$. The sorted items are placed at the $w_{k-3} = 0$ cube.
     (In this step $n^{k-2}$ items are sorted on a $(k-1)$-cube).
  4. Sorting pairs of blocks in an $x$-major order:
     For each pair of blocks do the following:
     (a) Sort each block in a $y$-major order with odd numbered blocks sorted as the default and the even blocks in a $bY - sX - bZ - bW_1 - ... - bW_{k-4}$ order.
        (Sorting a block is done by sub-algorithm 2.1).
     (b) Route appropriate items between the blocks: The $i$'th smallest item in the block with lower $x$ coordinate values is switched with the $i$'th largest one in the other block.
        (The route is performed by sub-algorithm 2.3).
     (c) Compare and change (A processor in the sub-block with lower $y$ coordinate values keeps the smaller item from the two it holds, a processor in the second sub-block keeps the larger item from the two it holds).
(d) Sort each block in a y-major order
(Sorting a block is done by sub-algorithm 2.1).

(e) Route items in each block to be sorted in an x-major order.
(The route is performed by sub-algorithm 2.2).

5. Sorting consequent pairs of blocks:
This step is executed the same as the preceding step with consequent numbered sub-blocks: instead of the pairs (1,2),(3,4),... of step 4, now the pairs are (2,3),(4,5),..., only this time in sub-step a, odd numbered blocks are sorted as the even blocks in step 3 and the even numbered blocks as the odd blocks in step 3.

3.2 Analysis and validity of the sorting algorithm

Claim 3.1 For all $k \geq 2$ and $n$, sorting $n^{k-1}$ items placed in a $\sqrt{n} \times \sqrt{n} \times n^{k-2}$ sub-cube on a $\sqrt{n} \times \sqrt{n} \times n^{k-1}$ cube, can be done by simulating with no slowdown a sort of $n^{k-1}$ items placed in an $n^{k-1}$ sub-cube on a k-cube.

Proof: The simulation is done by folding the k-cube into a $\sqrt{n} \times \sqrt{n} \times n^{k-1}$ cube. In the simulating cube, every $n$ processors in the sub-cube with last $k-1$ coordinate values fixed, will simulate the $n$ processors in the sub-cube with same last $k-1$ coordinate values, in the simulated cube. The only connections taking part at every $\sqrt{n} \times \sqrt{n}$ mesh in the simulating cube, will be those forming a snake-like folded line. Since every processor simulates only one processor of the simulated cube, and since for every processor $p$ in the simulated cube, the simulating processor is connected to all the processors that simulate the neighbors of $p$, the above simulation can be done with no slowdown.

Remark: The sorted items will be by the end of the simulation ordered in each mesh in a snake-like manner. Turning the order into a row-major or column-major order can be done in a constant number of steps.

Lemma 3.1 For any $k \geq 2$ and $n$, the sorting algorithms used for sorting $n^{k-1}$ items on an $n^k$ reconfigurable cube as declared and presented in this section (starting with algorithm 2 for $k \geq 3$), terminate after $3!^{k-2}c_1 + ((3!^{k-2} - 1)/30)c_2$ steps, where $c_1$ denotes the number of steps of the MESH-SORT algorithm and $c_2$ denotes the total number of steps required for routing certain permutations on sub-cubes.

Proof: Let us define $T(n,k)$ as the time to sort $n^{k-1}$ items with $n^k$ processors.
The proof is given by induction on $k$:

Base:

$k = 2$:
The only algorithm used is the MESH-SORT algorithm. Therefore $T(n,2) = c_1$ satisfying the claim for the base of induction.
Induction Step:
Assume the lemma holds for all $2 \leq k \leq m$. That is, for all $2 \leq k \leq m$, sorting $n^{k-1}$ items on a $k$-cube by using the sorting algorithms declared and presented in this section, takes $31^{k-2}c_1 + ((31^{k-2} - 1)/30)c_2$ steps (with $c_1$ and $c_2$ as defined in the claim). It remains to prove that the lemma holds for $k = m + 1$ as well:

Let us look at the execution of the sorting algorithms, at the highest level of the recursion (with $m + 1$ dimensions):

Sub-algorithms 2.4, 2.3, 2.4 take constant time each.
A sorting algorithm of $n^{m-1}$ items on an ($m$-cube takes $T(n, m) = 31^{m-2}c_1 + ((31^{m-2} - 1)/30)c_2$ steps (by induction's recursion).

A sorting algorithm of $\sqrt{n} \times \sqrt{n} \times n^{m-2}$ items on a $\sqrt{n} \times \sqrt{n} \times n^{m-1}$ cube takes $T(n, m) = 31^{m-2}c_1 + ((31^{m-2} - 1)/30)c_2$ steps (by the induction's recursion and claim 3.1).

Sub-algorithm 2.1 for $m + 1$ dimensions includes: Two compare and change operations, two calls to sub-algorithm 2.4, and six calls to a sorting algorithm of $n^{m-1}$ items in an $n^{m-1}$ sub-cube on an $n^m$ reconfigurable cube (or in an $\sqrt{n} \times \sqrt{n} \times n^{m-2}$ sub-cube on an $\sqrt{n} \times \sqrt{n} \times n^{m-1}$ sub-cube).

Algorithm 2 for $m + 1$ dimensions includes: One compare and change operation, two calls to sub-algorithm 2.3, three calls to sub-algorithm 2.2, five calls to sub-algorithm 2.1 and one call to a sorting algorithm for sorting $n^{m-1}$ items in an $n^{m-1}$ sub-cube on an $m$-cube.

$\Rightarrow T(n, m + 1) = 31T(n, m) + c_2$ for $c_2$ a constant value denoting the total number of steps required for routing permutations operations (All these operations are executed in sub-algorithms 2.4, 2.3 and 2.2).

$\Rightarrow T(n, m + 1) = 31^{m-1}c_1 + c_2 \sum_{i=0}^{m-2} 31^i = 31^{m-1}c_1 + ((31^{m-1} - 1)/30)c_2$

Claim 3.2 Algorithm 2 and its sub-algorithms include only:

- Prespecified comparison-exchange operations
- Prespecified routes
- Calls with sub-sets of the original set of items to sorting routines. These routines sort correctly input sets with arbitrary values, and return the set's items to the same set of processors they started at.

Proof: Algorithm 2 and its sub-algorithms include only: prespecified comparison-exchange operations (steps 3.c, 4.c of sub-algorithm 2.1 and steps 4.c, 5.c of algorithm 2), prespecified routes (sub-algorithms 2.4, 2.3, 2.2) and calls at the base of the recursion to applications of the MESH-SORT. These applications can sort correctly $n$ items according to theorem 1.1 and claim 3.1 and they return the sorted items to the same processors they started at (steps 1, 2, 3.a, 3.d, 4.a, 4.d of sub-algorithm 2.1 and step 3 of algorithm 2 with $k = 3$).
The (0-1)* sorting lemma:
If an algorithm which includes only the three kinds of operations described at claim 3.2, sorts all input sets consisting solely of 0s and 1s, then it sorts all input sets with arbitrary values.

The proof is essentially an extension of the proof in [Lei91].

Proof: The proof for the (0-1)* lemma:

The proof is by contradiction. Assume that such an algorithm fails to correctly sort some set of input items \(x_1, x_2, \ldots, x_n\). Let \(\Pi\) be a permutation such that

\[ x_{\Pi(1)} \leq x_{\Pi(2)} \leq \cdots \leq x_{\Pi(n)} \]

and let \(\sigma\) be a permutation such that the output of the sorting algorithm is \(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}\). Let \(k\) be the smallest item such that \(x_{\sigma(k)} \neq x_{\Pi(k)}\). (There exists such an item of \(k\) since we have assumed that the sorting algorithm fails on \(x_1, x_2, \ldots, x_n\).

By definition this means that \(x_{\sigma(i)} = x_{\Pi(i)}\) for \(1 \leq i < k\) and therefore that \(x_{\sigma(k)} > x_{\Pi(k)}\). Hence, there must be a value of \(r > k\) such that \(x_{\sigma(r)} = x_{\Pi(r)}\).

Define

\[ z_i^* = \begin{cases} 0 & \text{if } x_i \leq x_{\Pi(k)} \\ 1 & \text{if } x_i > x_{\Pi(k)} \end{cases} \]

and examine possible actions of the algorithm on the input set obtained by replacing \(x_i\) with \(z_i^*\) for \(1 \leq i \leq n\):

- A comparison-exchange operation:

  Since

  \[ x_i \geq x_j \Rightarrow z_i^* \geq z_j^* \]

  for every \(i\) and \(j\), the algorithm performs the same comparison-exchange operation.

- A route operation: Since the route operations the algorithm use depend only on the identity of the processors and the time, the algorithm performs the same route operation.

- Calls to a sorting procedure as defined in the claim: Suppose we send the procedure \(x_{\Pi_1}, x_{\Pi_2}, \ldots, x_{\Pi_k}\). The same procedure in the 0-1 case will be sent \(x_{\Pi_1}^*, x_{\Pi_2}^*, \ldots, x_{\Pi_k}^*\). Since

\[ x_{m_1} \geq x_{m_2} \geq \cdots \geq x_{m_k} \Rightarrow x_{m_1}^* \geq x_{m_2}^* \geq \cdots \geq x_{m_k}^* \]

the sorting algorithm performs the same sorting on the \(z^*\) input as it does on the \(x_i\) input.

\[ \Rightarrow \text{The algorithm performs the same on the } z^* \text{ input as it does on the } x_i \text{ input. Hence, the output of the algorithm on the 0-1 values will be} \]

\[ x_{\sigma(1)}^*, x_{\sigma(2)}^*, \ldots, x_{\sigma(k-1)}^*, x_{\sigma(k)}^*, \ldots = 0, 0, \ldots, 0, 1, \ldots, 0, \ldots \]

Which is incorrect. This contradicts the assumption that the algorithm correctly sorts all 0-1 input sets, and thus the lemma is proved.

The proof of the validity of the sorting algorithm will rely on the validity of the following Midsection algorithm.
The Midsection algorithm for sorting a group $A$ with an even number of items:

- **Input:** Items of group $A$ placed one item in a processor ($|A| = 2l$).
- **Output:** Group $A$ sorted on the same processors.
- **Sorting steps:**
  1. Sort: Divide $A$ into two groups $N$ and $M$ of processors with equal number of items and sort each group separately.
  2. Route: Route the $i$'th smallest item in $N$ to the processor with the $i$'th largest item in $M$. Route the $i$'th largest item in $M$ to the processor with the $i$'th smallest item in $M$.
  3. Compare-exchange: Processors in $N$ compare the input item with the item they received at the preceding step and hold the smaller one. Processors in $M$ compare the input item with the item they received at the preceding step and hold the larger one.

**Corollary 3.1** When the only input items are 0s and 1s, after step 3 of the Midsection algorithm either $N$ is all 0s or $M$ is all 1s.

**Proof:**
- In case before start of execution of the algorithm, group $N$ contained only 0s or group $M$ contained only 1s, the only time in $N$ or $M$ contains a change could erase, is during step 3. Because in this step items in $N$ are changed only into smaller ones, and items in $M$ are changed only into larger ones, no change could erase during the step.
- Otherwise, $N$ has at least one 1 and $M$ has at least one 0 before start of execution. Let us name the processor's index in each group after step 1, starting with 1 for the processor with the smallest item and ending with $l$ for the processor with the largest item ($|N| = |M| = l$), and refer to the item in the $i$'th index in $N$ (or $M$) by $N(i)$ (or $M(i)$). Now let us look in group $N$ before step 3 at the processor with the smallest index that contains a 1 value and call its index $i$ ($1 \leq i \leq m$).

**If** after step 3 $N(i) = 0$, then $N$ is all 0s because:
- For all $j < i$, $N(j) = 0$ from the start.
- $N(i)$ has changed to 0.
- For all $j > i$, because $N(i)$ is compared with $M(l - i + 1)$ (a 0 value), and $N(j)$ is compared with $M(l - i + 1)$ where $M(l - i + 1) < M(l - i + 1)$ then $M(l - i + 1)$ is certainly a 0 value before the exchange and so after step 3 $N(j)$ has a 0 value.

**else** after step 3 $M$ is all 1s because:
- For all $j$ so that $m - j + 1 \geq m - i + 1$, $M(j) = 1$ from the start.
- For all $j$ so that $m - j + 1 < m - i + 1$, $M(j)$ is compared with $N(j)$. Since $N(j) > N(i)$ and $N(i) = 1$, after step 3 $M(l - j + 1)$ has a 1 value.

□
Claim 3.3 Assume the sortings in steps 1, 4 of the Midsection algorithm sort correctly and preserve the (0-1)* conditions, then the Midsection algorithm sorts correctly group A in the upward order $N, M$.

Proof: Because of the assumption in claim, the algorithm preserve the (0-1)* conditions, we can prove the claim only on 0-1 values and the proof for arbitrary values will follow from the (0-1)* lemma. By corollary 3.1 and step 4 of the algorithm the proof is completed. \qed

In the following claims we will consider only input sets consisting solely of 0s and 1s. We will use some definitions:

- $C_k$ - 
The $k$-cube.
- $B_i$ - 
The $i$'th block $\forall i \in \{1, \ldots, \sqrt{n}\}$.
- $SB_i$ - 
The $i$'th sub-block $\forall i \in \{1, \ldots, \sqrt{n}\}$.
- $X(condition)$ - 
  All the items in $X$ which fulfill the condition. $X$ can be any sub-cube within $C_k$.
- $\triangle(E, F)$ - 
  $|\text{number of 1s in } E - \text{number of 1s in } F|$. $E$ and $F$ can be any sub-cubes within $C_k$.
- Dirty cube - 
  A cube that contains both 0s and 1s.
- 0-1 input items - All input sets consisting solely of 0s and 1s.

Claim 3.4 For all $k \geq 3$, sub-algorithm 2.2 routes items in each block from a $y$-major order to an $x$-major order. All items in each block start and end in the $w_{k-3}$ sub-cube.

Proof: Let us define a free axis $z_i$ as: Among all processors that their coordinate's values differ only in the $z_i$ coordinate, only one contains an item.

Since it is only needed to change the order between the $z$ and $y$ coordinates, at the beginning of the algorithm, the position of each item is correct except for its $z, y$ position.

Step 1 uses the free $w_{k-3}$ axis to move items so that at the end of the step all items with same $y$ position are newly located at the same $w_{k-3}$ cube.

Step 2 uses the free $y$ axis to move each item to its correct $y$ position. Items in processors with distinct $z$ position are lead to distinct $y$ positions.
step 3 uses the free z axis to move each item to its correct z position.

After step 3 it only remains to bring each item back to the \( w_{k-3} = 0 \) sub-cube and this is done by step 4.

Claim 3.5 For 0-1 input items and for all \( k \geq 3 \), all steps in sub-algorithm 2.1 do can be performed correctly.

Proof:
step 1:
In this step \( \sqrt{n} \) sub-blocks have to be sorted in parallel. In each sub-block \( \sqrt{n} \times \sqrt{n} \times n^{k-3} \) items placed at the \( w_{k-3} = 0 \) sub-cube of an \( \sqrt{n} \times \sqrt{n} \times n^{k-2} \) sub-cube have to be sorted in a y-major order. This can be done by a simulation with no slowdown of sorting \( n^{k-2} \) items placed at the \( w_{k-2} = 0 \) sub-cube of an \( n^{k-1} \) cube in an x-major order (claim 3.1). The last sorting can be done correctly due to the induction's assumption.

step 2:
In this step \( \forall i \in \{1, \ldots, n\}, n^{k-2} \) items placed at the \( C_k(x = i, w_{k-2} = 0) \) sub-cube have to be sorted in a y-major order (The x coordinate doesn't take part) on a \( (k-1) \)-cube. This sorting can be done correctly due to the induction's assumption.

step 3:
In this step pairs of sub-blocks are sorted in a y-major order. The algorithm the four sub-steps are based on is the Midsection algorithm and is therefore correct (Since the sortings used in these steps sort correctly (due to induction's assumption) and don't intrude the \((0-1)^*\) lemma conditions (claim 3.2), the assumption for algorithm correctness is not intruded).

Sub-steps a and d are performed as step 1 and are therefore correct.

Sub-step c is a simple compare-exchange operation.

Sub-step b is executed by sub-algorithm 2.4. This sub-algorithm routes between the items in processor \((i_1, i_2, \ldots, i_{k-4}, 0)\) of the \( k \)-th sub-block and processor \((i_1, 2k\sqrt{n} - i_2 + 1, \ldots, i_{k-4}, 0)\) of the \( k+1 \)-th sub-block. Because of the special way we sorted the sub-blocks in sub-step a, it happens that the route is between the \( i \)-th smallest item in sub-block \( k \) and the \( i \)-th largest item in sub-block \( k+1 \).

step 4:
This step is performed exactly as step 3 with consequent pairs of sub-blocks.
Claim 3.6 For 0-1 input items and for all \( k \geq 3 \), at the end of step 1 of sub-algorithm 2.1 for sorting a block \( B \):

\[
\Delta(B(x = i), B(x = j)) \leq \sqrt{n} \times n^{k-3}
\]

for all \( i, j \) within the two \( x \) borders of block \( B \).

Proof: In \( B \) there are \( \sqrt{n} \) sub-blocks \( SB_s \) with \( s \in \{1, \ldots, \sqrt{n}\} \) which at the end of step 1 are all sorted in a \( y \)-major order (claim 3.5). This means in each sub-block \( SB_s \) there is at most one index \( l \) for \( l \in \{(s-1)\sqrt{n} + 1, \ldots, s\sqrt{n}\} \) so that the sub-cube \( SB_s(y = l) \) is dirty. Therefore, with regard to the two sub-cubes \( SB_s(x = i) \) and \( SB_s(x = j) \), we get the following inequality:

\[
\Delta(SB_s(x = i), SB_s(x = j)) = \Delta(SB_s(x = i, y = l), SB_s(x = j, y = l)) \leq n^{k-3}
\]

Since there are \( \sqrt{n} \) such sub-blocks,

\[
\Delta(B(x = i), B(x = j)) \leq \sum_{s=1}^{\sqrt{n}} \Delta(SB_s(x = i), SB_s(x = j)) \leq \sqrt{n} \times n^{k-3}
\]

\[
\square
\]

Claim 3.7 For 0-1 input values and for all \( k \geq 3 \), at the end of step 2 of sub-algorithm 2.1 the whole block \( B \) is sorted in a \( y \)-major order except for maybe two consequent sub-blocks.

Proof: At the end of step 2 for all \( s \) between the two \( x \) borders of the block, every \( B(x = s) \) is sorted in a \( y \)-major order (claim 3.5). This means for all \( s \) between the two \( x \) borders of the block there is at most one index \( m \in \{1, \ldots, n\} \) so that \( B(x = s, y = m) \) is dirty.

Let us define \( m_s \) as:

\[
m_s = \begin{cases} 
\max \{m \mid B(x = s, y = m) \text{ contains a zero} \} & \text{if } B(x = s) \text{ is dirty} \\
n & \text{if } B(x = s) \equiv 0 \\
0 & \text{if } B(x = s) \equiv 1 
\end{cases}
\]

If we assume without loss of generality that \( m_i \geq m_j \), we get the following inequality:

\[
\Delta(B(x = i), B(x = j)) > (m_i - m_j - 1)n^{k-3}.
\]

From claim 3.6 we have:

\[
\Delta(B(x = i), B(x = j)) \leq \sqrt{n} \times n^{k-3}
\]

\[
\implies m_i - m_j \leq \sqrt{n}
\]

\[
\square
\]

Claim 3.8 For 0-1 input items and for all \( k \geq 3 \), sub-algorithm 2.1 sorts a block \( B \) in a \( y \)-major order on the \( w_{k-3} = 0 \) cube.

Proof: By claim 3.7, after step 2 of the sub-algorithm it only remains to sort two consequent sub-blocks. By claim 3.5 steps 3 and 4 sort all consequent sub-blocks, particularly the two we needed.
Claim 3.9  For 0-1 input items and for all $k \geq 3$, all steps in algorithm 2 can be performed correctly.

Proof: step 1:
- sorting blocks is done correctly (claim 3.8).

step 2:
- route from a $y$-major order to an $x$-major order is done correctly (claim 3.4).

step 3:
- Sorting $n^{k-2}$ items on an $n^{k-1}$ cube in an $x$-major order is done correctly by induction's assumption.

step 4:
- In this step pairs of blocks are sorted in an $x$-major order. The algorithm the four sub-steps are based on is the Midsection algorithm and therefore correct (Since the sortings used in these steps sort correctly (induction's assumption) and don't intrude the (0-1)* lemma conditions (claim 3.2), the assumption of the algorithm is not intruded).
- sub-steps $a$ and $d$ are performed as step 1 and are therefore correct.
- sub-step $c$ is a simple compare-exchange operation.
- Sub-step $b$ is executed by sub-algorithm 2.3. This sub-algorithm routes between the items in processor $(i_1, i_2, \ldots, i_{k-4}, 0)$ of the $k$'th block and processor $(2k\sqrt{n} - i_1 + 1, i_2, \ldots, i_{k-4}, 0)$ of the $k+1$'th block. Because of the special way we sorted the sub-blocks in sub-step $a$, it happens that the route is between the $i$'th smallest item in block $k$ and the $i$'th largest item in sub-block $k+1$.
- sub-step $e$ inverses the operation in step 2.

step 5:
- This step is performed exactly as step 4 with consequent pairs of blocks.

Claim 3.10  For 0-1 input items and for all $k \geq 3$, at the end of step 2 of algorithm 2:

$$\Delta(C_k(y = i), C_k(y = j)) \leq \sqrt{n} \times n^{k-3}$$

for all $i, j \in \{1, \ldots, n\}$.

Proof: In $C_k$ there are $\sqrt{n}$ blocks $B_s$ with $s \in \{1, \ldots, \sqrt{n}\}$ which at the end of step 2 are all sorted in a $y$-major order (claim 3.9). This means in each block $B_s$ there is at most one index $l$ for $l \in \{(s - 1)\sqrt{n} + 1, \ldots, s\sqrt{n}\}$ so that the sub-cube $B_s(x = l)$ is dirty. Therefore, with regard to the two sub-cubes $B_s(y = i)$ and $B_s(y = j)$, we get the following inequality:

$$\Delta(B_s(y = i), B_s(y = j)) = \Delta(B_s(y = i, x = l), B_s(y = j, x = l)) \leq n^{k-3}$$

Since there are $\sqrt{n}$ such blocks,

$$\Delta(C_k(y = i), C_k(y = j)) \leq \sum_{s=1}^{\sqrt{n}} \Delta(B_s(y = i), B_s(y = j)) \leq \sqrt{n} \times n^{k-3}$$

\[\square\]
Claim 3.11 For 0-1 input items and for all $k \geq 3$, at the end of step 3 of algorithm 2 the whole cube $C_k$ is sorted in an x-major order except for maybe two consequent blocks.

Proof: At the end of step 3 for all $s \in \{1, \ldots, n\}$ every $C_k(y = s)$ is sorted in an x-major order (claim 3.9). This means for every $s \in \{1, \ldots, n\}$ there is at most one index $m \in \{1, \ldots, n\}$ so that $C_k(y = s, x = m)$ is dirty.

Let us define $m_s$ as:

$$m_s = \begin{cases} \max \{ m \mid C_k(y = s, x = m) \text{ contains a zero} \} & \text{if } C_k(y = s) \text{ is dirty} \\ n & \text{if } C_k(y = s) = 0 \\ 0 & \text{if } C_k(y = s) = 1 \end{cases}$$

If we assume without loss of generality that $m_i \geq m_j$, then the following inequality holds:

$$\Delta(C_k(y = i), C_k(y = j)) > (m_i - m_j - 1)n^{k-3}$$

From claim 3.10 we have:

$$\Delta(C_k(y = i), C_k(y = j)) \geq \sqrt{n} \times n^{k-3}$$

$$\Rightarrow m_i - m_j \leq \sqrt{n} \quad \square$$

Claim 3.12 For 0-1 input items and for all $k \geq 3$, algorithm 2 sorts the cube $C_k$ in an x-major order on the $w_{k-3} = 0$ cube.

Proof: By claim 3.11, after step 3 of the algorithm 2, it only remains to sort two consequent blocks. By claim 3.9, steps 4 and 5 sort all consequent blocks, particularly the two we needed. \(\square\)

Lemma 3.2 For input sets with arbitrary items and for any $k \geq 2$ and $n$, when using the sorting algorithms declared and presented in this section (starting with algorithm 2 for $k \geq 3$), with $n^{k-1}$ items placed on the $w_{k-3} = 0$ sub-cube of a $k$-cube, all items are sorted correctly in an x-major order and are finally located in the $w^{k-3} = 0$ cube.

Proof: We will prove lemma 3.2 only for 0-1 input items. The proof for input sets with arbitrary items will follow from the (0-1)* sorting lemma.

The proof is given by induction on $k$:

base:

$k = 2$: The mesh-sort sorts correctly $n$ items placed in a row on an 2-cube (theorem 1.1).

induction step:

Assume for all $2 \leq k \leq m$ the lemma holds. It remains to prove the lemma holds for $k = m + 1$, that is for $k = m + 1$ algorithm 2 together with the rest of the algorithms declared and presented in this section, sorts $n^{k-1}$ items (0,1 values) on a $k$-cube in an x-major order. Claim 3.12 terminates the induction step by proving that for all $k \geq 3$ algorithm 2 sorts the $C_k$ cube in an x-major order on the $w_{k-3} = 0$ cube. Therefore the proof of claim 3.2 has ended. \(\square\)
Theorem 3.1 For any $k \geq 2$ and $n$, $n^{k-1}$ items can be sorted on the $k$-cube in $31^{k-2}c_1 + ((31^{k-2} - 1)/30)c_2$ steps, where $c_1$ denotes the number of steps of the MESH-SORT algorithm and $c_2$ denotes the total number of steps required for routing certain permutations on sub-cubes (See sub-algorithms 2.4, 2.3, 2.2 of this section).

Proof: By lemma 3.1 and lemma 3.2, the proof of theorem 3.1 has ended. \hfill \Box

References


