ON MERGING NETWORKS

Tamir Levy
ON MERGING NETWORKS

Research Thesis

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Tamir Levy

Submitted to the Senate of The Technion - Israel Institute of Technology

Adar, 5771 Haifa February 2011
The Research Thesis Was Done Under
The Supervision of
Professor Ami Litman
in the Faculty of Computer Science

The Generous Financial Help Of The Technion Is Gratefully Acknowledged
# Table of Contents

Table of Contents iv

List of Tables vi

List of Figures vii

1 Introduction 1

2 Optimal Conclusive Sets for Comparator Networks 6
   2.1 Introduction ................................................. 6
   2.2 Comparator Networks .................................... 9
      2.2.1 Functionality ....................................... 10
   2.3 The 0-1 Principle ....................................... 11
   2.4 Smaller Conclusive Sets ................................ 12
      2.4.1 Sandwiches for merging .............................. 13
      2.4.2 Unitonic vectors for Bitonic sorting .............. 14
      2.4.3 Balanced vectors for halving ...................... 14
      2.4.4 Conclusive sets for sorting ..................... 16
   2.5 Lower Bounds for Conclusive Sets .................... 17

3 Accelerating certain outputs of merging and sorting networks 21
   3.1 Introduction ............................................. 22
   3.2 Preliminaries ........................................... 24
   3.3 The Asymmetric Tri-section Merging Technique .......... 26
   3.4 The Symmetric Tri-section Merging Technique ........... 32
   3.5 A counterexample ....................................... 34
   3.6 Accelerating Sorting Networks ........................... 35
4 The Strongest Model of Computation Obeying 0-1 Principles 37
4.1 Introduction .................................................. 37
4.2 Related work .................................................. 41
4.3 The Min-Max Programming Model .......................... 43
4.4 The 0-1 Principle and the min-max model ................ 44
4.5 The min-max model is stronger than the comparator model .. 46
4.6 The Bouricious Principles ................................. 47
4.7 The 0-1 Comparing Principle. ............................. 50
4.8 The Robustness Principle ................................ 52

5 Lower Bounds On Bitonic Sorting Networks 54
5.1 Introduction .................................................. 54
5.2 Preliminaries .................................................. 55
5.3 Related work .................................................. 56
5.4 Lower Bound of Bitonic sorters ............................. 56

6 Bitonic sorters of minimal depth 62
6.1 Introduction .................................................. 62
  6.1.1 Related work .............................................. 64
  6.1.2 Preliminaries ............................................. 66
6.2 Symmetric Oblivious Algorithms ........................... 67
6.3 Certain Bitonic Sorters ..................................... 76
  6.3.1 Sorting 1-heavy words ................................... 76
  6.3.2 Sorting words of even width ........................... 82
6.4 Graphical representation of Networks. ..................... 83
6.5 Bitonic sorters of odd width ............................... 86

7 Conclusion 89
7.1 The min-max model ......................................... 89
7.2 Conclusive sets .............................................. 90
7.3 Symmetry of comparator networks ........................ 91
7.4 Merging networks ............................................ 91
7.5 Bitonic sorters ............................................... 92
7.6 Further research ............................................. 93

Bibliography 96
List of Tables

2.1 Summary of results: sizes of conclusive sets for various functionalities. . . 7
## List of Figures

2.1 A merging network of width 4 and depth 2. .......................... 9
2.2 A network that sorts all 0-1 vectors except $v$. ......................... 18

3.1 A merging network of width 4 and depth 2. .......................... 24
3.2 The network $T^{5,11}$. ...................................................... 26
3.3 A comparator receiving $a_0$ and $b_4$ and producing $y_0$ and $x_4$. 26
3.4 The network $T^{5,11}$ processing the sandwich with $a_0 = 2$. .......... 29
3.5 An ascending-descending sorter. ........................................... 35

4.1 A min-max network ............................................................ 38
4.2 A comparator network ....................................................... 39
4.3 An extension as per Lemma 4.5.3. ......................................... 47

5.1 A Bitonic sorter of width 5 and depth 4 ................................. 55
5.2 Three circular vectors. ....................................................... 57

6.1 (a) Batcher’s Bitonic sorter; (b) Batcher’s odd-even merging network. 66
6.2 A min-max network of width 6 and depth 3. ............................ 67
6.3 Batcher’s Bitonic sorter of width 8 and its dual network. ............... 74
6.4 Batcher’s odd-even merging network of width 8 and its dual network. 75
6.5 Graphical representation of the networks $B(2k)$ and $C(2k - 1)$. .... 84
Chapter 1

Introduction

This work studies comparator networks - a well known model of parallel computation. This model is used extensively for keys arrangement tasks such as sorting and selection. This work investigates several aspects of comparator networks.

It starts with presenting handy tools for analysis of comparator networks in the form of conclusive sets - non-binary vectors that verify a specific functionality. The 0-1 principle introduced by Knuth [14] states that a comparator network is a sorting network if and only if it sorts all binary inputs. Hence, it points out a certain binary conclusive set. Clearly, sorting is not the only functionality that comparator networks are useful for nor the 0-1 Principle. Additional functionalities may include merging, separation, sorting restricted sets of vectors, etc. For each of these functionalities, some variant or another of the 0-1 principle [14] was used for proving the correctness of the networks in question and always using binary vectors. This work investigates non-binary conclusive sets and proposes for the above functionalities conclusive sets of minimal size. That is, we state the minimal size of a conclusive set (which is dependent of the functionality in question and of the number of keys to be processed) and propose a conclusive set of that size.

We next use some of these conclusive sets to study comparator networks in which several of the outputs are accelerated. That is, these outputs are generated much faster than the other outputs, and this without hindering the other outputs. Namely, for every $0 < k \leq n$, we present a merging network of minimal depth that merges two sorted sequences of length $n$ into a single sorted sequence. This merging network produces either the lowest $k$ keys or the highest $k$ keys$^1$ after a delay of $\lceil \log(k) \rceil + 1$ comparators. Building on that, we construct, for every $0 < k < n$, an $n$-key sorting network that accelerates its $k$ lowest or its $k$ highest outputs. This sorting network is a merge-sort network$^2$ and has a minimal depth among these networks. Namely, its depth is $\lceil \log(n) \rceil \cdot \lceil \log(2n) \rceil \div 2$, the same depth as the Batcher

---

$^1$When $n$ is a power of two, both the lowest $k$ keys and the highest $k$ keys can be accelerated.

$^2$A merge-sort network is a sorting network which operates as follows. The input is arbitrarily divided into
merge-sort networks [1]. However, in contrast to the Batcher merge-sort networks which may accelerate only the first and last outputs, our merge-sort networks accelerates either the $k$ lowest keys or $k$ highest keys to a delay of less than $\lceil \log(n) \rceil \cdot \lceil \log 2k \rceil$ comparators.

This acceleration is based on a new merging technique, the Tri-section technique, that separates, by a depth one network, two sorted sequences into three sets, such that every key in one set is smaller or equal to any key in the following set. After this separation, each of these sets can be sorted separately and this leads to the desired acceleration. The idea of separating the input into two sets is known and is used, for example, in the Bitonic sorter of Batcher [1]; however, to the best of our knowledge, separation into three sets as above is novel.

To put our results in context, let us compare the acceleration of our networks with the acceleration of other well-known merging networks – The Bitonic sorter and the odd-even merging network, both of Batcher [1]. The Bitonic sorter has no accelerated outputs at all; all outputs have exactly the same delay. On the other hand, the odd-even merging network has only two accelerated outputs, the first and the last ones whose delay is exactly one. All other outputs have the same delay.

To the best of our knowledge, the idea of accelerating certain outputs was never addressed. The only prior work which is somewhat similar to our work concerns selectors. A $(k, n)$-selector is a network that separates a set of $n$ keys into the lowest $k$ and the other keys. Fast selection leads to a sorting network that accelerate certain outputs, as follows: First, the $k$ lowest keys are separated from the other keys. Afterwards, each set is sorted separately. Yao presented a $(k, n)$-selector which is efficient when $k$ is constant and $n$ is very large. This selector can be extended into a sorting network that accelerates its lowest $k$ outputs; however, the depth of the resulting sorting network exceeds the minimal depth of a merge-sort network. Our network accelerates the lowest $k$ outputs while its depth is minimal among merge-sort networks.

The second additional contribution concerns Batcher’s merging techniques. Batcher’s odd-even merging technique [1] works as follows: Each of the input sequences is partitioned into its even part and its odd part. The even part of one sequence is merged with the even part of the other sequence recursively and similarly, the odd parts are merged. Finally, the two resulting sequences are merged into a single sorted sequence by a depth one network.

A slight variant of this method, due to Knuth [14, pp 231] and Leighton [15, pp 623], recursively merges the even part of each input sequence with the odd part of the other sequence. Again, the resulting two sorted sequences can be merged by a depth one network. We refer to the family of networks produced by allowing each of the above two variants anywhere in the recursion process as Batcher merging networks. It was shown in [16] that all published merging networks, whose width is a power of two, are members of this family. All these merging networks are of minimal depth and have no degenerate comparators. (A two sets of (almost) equal size and each set is recursively sorted; the two sorted sequences are then merged.)
degenerate comparator has a fixed incoming edge whose value is always greater or equal to the value on the other incoming edge, for every valid input of the network.) The above fact arise the following question:

**Question 1.** Are the Batcher merging networks the only merging networks with the following properties:

1. Their width, $2n$, is a power of two.
2. Their depth is minimal – $\log(2n)$.
3. They have no degenerate comparators.

The Tri-section technique provides a negative answer to this question.

We next consider a model of computation, called the min-max model, which is slightly different then the acceptable comparator model. The main difference between them is a certain fanout restriction which exists only in the comparator model. This difference makes the min-max model to be somewhat ‘stronger’ then the comparator model. It is stronger both in aspects of computability (some functions can be computed only in the min-max model) and complexity (some functions can be computed faster in the min-max model). As shown by Knuth, inserting a single key into a sorted sequence of $n$ keys can be performed by a constant depth min-max model while in the comparator model a network of depth $\lceil \log(n) \rceil$ is required. We show that in some cases sorting Bitonic sequences can be performed faster by a min-max model (however, in this case the depth difference is at most one.).

We compare these two models by considering several 0-1-like principles and show that the min-max model is the ‘strongest’ model of computation which obeys our principles. That is, if a function is computable in a model of computation in which any of these principles holds, this function can be computed by a min-max network.

This work also studies the concept of symmetry of comparator networks. We distinguish between two types of such a symmetry. The first type, called *strong symmetry*, was studied in [29] and it concerns the structure of a comparator network. We propose a second type of symmetry, called *weak symmetry* and is more of a “black-box” approach; that is, this symmetry concerns only the input-to-output mapping of the network in question. In this work we use only the weak symmetry and the strong symmetry is discussed only for completeness. Formality and details of these symmetries is discussed in Section 6.2.

The weak symmetry allows us to extend the well-known 0-1 Principle [14] as follows. A binary sequence of keys is 1-heavy if at least half of its keys are 1. Similarly, it is 0-heavy if at least half of its keys are 0. (A binary sequence with an even number of keys may be 0-heavy and 1-heavy simultaneously). As said, Section 6.2 provides a precise definition of ‘weakly symmetric comparator network’ and establishes the following extension of the 0-1 Principle:
Theorem 2. Let $S$ be a weakly symmetric comparator network. Then the following four statements are equivalent:

- $S$ sorts every input sequence.
- $S$ sorts every binary input.
- $S$ sorts every binary 0-heavy input.
- $S$ sorts every binary 1-heavy input.

Finally we consider Bitonic sorters - comparator networks that sort Bitonic sequences of a certain width. Building on previous works, it establishes that the depth of an $n$-keys Bitonic sorter is at least $2 \left\lceil \log(n) \right\rceil - \left\lfloor \log(n) \right\rfloor$. It then layout, for every $n$, an $n$-keys Bitonic sorter of such a depth.

When $n$ is a power of two, $2 \left\lceil \log(n) \right\rceil - \left\lfloor \log(n) \right\rfloor = \log(n)$. The fact that, in this case, $\log(n)$ is the minimal depth is due to the seminal work of Batcher [1]. However, the minimal depth of Bitonic sorters, in the general case, was unknown. Let $T(n)$ denote the minimal depth of a Bitonic sorter of $n$ keys. We now summarize what was previously known on the function $T$. By a straightforward reachability argument, for every $n$,

$$\left\lceil \log(n) \right\rceil \leq T(n). \quad (1.1)$$

Due to the constructions of Batcher [1]:

$$T(2n) \leq T(n) + 1. \quad (1.2)$$

This and Inequality (1.1) imply that:

$$T(2^j) = j. \quad (1.3)$$

Nakatani et al. [25] introduced an elegant manner to construct a Bitonic sorter. This technique allows to produce wider Bitonic sorters from smaller ones. Namely, let $B1$ and $B2$ be two Bitonic sorters of width $w1$ and $w2$ and of depth $d1$ and $d2$, respectively. They construct a new Bitonic sorter of width $w1 \cdot w2$ and of depth $d1 + d2$. This implies that:

$$T(i \cdot j) \leq T(i) + T(j). \quad (1.4)$$

The only prior technique that constructs Bitonic sorters of any width is due to Batcher and Liszka [22]. They show that:

$$T(n) \leq \max \left( T\left( \left\lceil \frac{n}{2} \right\rceil \right), T\left( \left\lfloor \frac{n}{2} \right\rfloor \right) \right) + 2.$$
This and a straightforward induction imply:

\[ T(n) \leq 2 \lceil \log(n) \rceil - 1. \]  
\[ \text{(1.5)} \]

The first, non-trivial, lower bound on \( T(n) \) is due to Levy and Litman [18]. They showed that for every \( n \) that is not a power of two:

\[ \lceil \log(n) \rceil + 1 \leq T(n). \]  
\[ \text{(1.6)} \]

This result, combined with Equality 1.3, yields the surprising corollary that \( T \) is not monotonic. For example, \( T(15) \geq 5 > 4 = T(16) \).

As said, our result is the exact value of \( T(n) \). Namely, for every \( n \):

\[ T(n) = 2 \lceil \log(n) \rceil - \lfloor \log(n) \rfloor. \]  
\[ \text{(1.7)} \]

In other words,

\[ T(n) = \begin{cases} 
\log(n), & \text{when } n \text{ is a power of two} \\
\lceil \log(n) \rceil + 1, & \text{otherwise}
\end{cases} \]

As said, sorting Bitonic sequences (whose width is not a power of two) is an example of a ‘natural’ functionality for which the min-max model is faster than the comparator model. Let \( T'(n) \) denote the minimal depth of a min-max network that sorts all Bitonic sequences of \( n \) keys.

The same reachability argument imply Inequality (1.1) also for min-max networks; therefore:

\[ \lceil \log(n) \rceil \leq T'(n) \]  
\[ \text{(1.8)} \]

Since every comparator network can be translated to a min-max network of the same depth, it follows that for every \( n \):

\[ T'(n) \leq T(n) \]  
\[ \text{(1.9)} \]

There are certain cases in which the exact value of \( T'(n) \) is known, as listed below. The exact value of \( T'(n) \) for other cases is yet unknown.

- \( T'(n) = \log(n) \) when \( n \) is a power of two. This follows from Inequalities (1.3,1.8) and (1.9).
- \( T'(n) = \lceil \log(n) \rceil + 1 \), for every odd \( n \). Levy and Litman [18] established that \( \lceil \log(n) \rceil + 1 \leq T'(n) \), for every odd \( n \). Inequality (1.9) and Theorem 21 provide the matching upper bound.
- \( T'(n) = \lceil \log(n) \rceil \) for \( n \in (10 \cdot 2^N) \). This was established in [18].
- \( T'(n) = \lceil \log(n) \rceil \) for \( n \in (6 \cdot 2^N) \), as shown in the next paragraph.

Due to this work, the exact value of \( T'(n) \) is almost known. That is,

\[ \lceil \log(n) \rceil \leq T'(n) \leq \lceil \log(n) \rceil + 1. \]
Chapter 2

Optimal Conclusive Sets for Comparator Networks

Dr. Guy Even, School of Electrical Engineering, Tel-Aviv University
Tamir Levi, Technion, Haifa
Prof’ Ami Litman, Technion, Haifa

A set of input vectors $S$ is conclusive for a certain functionality if, for every comparator network, correct functionality for all input vectors is implied by correct functionality for all vectors in $S$. We consider four functionalities of comparator networks: sorting, merging, sorting of Bitonic vectors, and halving. For each of these functionalities, we present two conclusive sets of minimal cardinality. The members of the first set are restricted to be binary, while the members of the second set are unrestricted. For all the above functionalities, except halving, the unrestricted conclusive set is much smaller than the binary one.

2.1 Introduction

The 0-1 principle introduced by Knuth [14] states that a comparator network is a sorting network if and only if it sorts all binary inputs. Sorting is not the only functionality that comparator networks are useful for. Additional functionalities include merging two sorted vectors, halving (i.e., separating $2^j$ keys into the $j$ lowest keys and the $j$ highest keys), and sorting restricted sets of vectors. For each of these functionalities, some variant or another of the 0-1 principle [14] was used for proving the correctness of the networks in question.

By the 0-1 Principle and its many variants, comparator networks “work properly” for all valid vectors if and only if they “work properly” for all binary valid vectors. For example,
to verify the functionality of a sorting network with \( n \) inputs and outputs, it is suffice to test all \( 2^n \) binary vectors. Clearly, there is no need to test constant vectors (vectors, all of whose keys are equal); hence, sorting can be verified by \( 2^n - 2 \) vectors. Can sorting be verified by fewer binary vectors? A second question lifts the binary restriction and asks: Can sorting be verified by even fewer vectors? Similar questions can be asked for other functionalities.

This paper discusses four functionalities: Sorting, merging, bitonic\(^1\) sorting and halving. The first three functionalities are similar – they all require the output to be sorted. The last functionality, halving, is significantly different from the former ones and this reflects in our proofs and in our results.

We refer to a set of vectors that verifies a specific functionality as a conclusive set. So far, only binary vectors were considered for conclusive sets. We introduce the usage of unrestricted vectors (e.g., vectors of natural numbers) for conclusive sets. Interestingly, our main result is that smaller conclusive sets are possible if unrestricted vectors are allowed. In addition, we prove lower bounds on the size of conclusive sets that imply the optimality of our constructions.

Table 2.1 summarizes our results. The first column in the table lists the four functionalities in question. (See Sec. 2.2.1 for formal definitions.) The second column lists the minimal sizes of binary conclusive sets. The third column lists the minimal sizes of unrestricted conclusive sets and the forth column lists the type of vectors used in our unrestricted conclusive sets of minimal size. In this table, as well as in the rest of this paper, \( n \) denotes the width of the network in question – the number of keys processed by the network. In the cases of merging and halving, \( n \) is required to be even.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Minimal size of binary conclusive set</th>
<th>Minimal Size of unrestricted conclusive set</th>
<th>Type of vectors in our minimal unrestricted conclusive set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorting</td>
<td>( 2^n - 2^{[14]} )</td>
<td>( \binom{n}{(n/2)} )</td>
<td>covering vectors</td>
</tr>
<tr>
<td>Merging</td>
<td>( \binom{n/2}{2} - 2^{[24]} )</td>
<td>( n/2 + 1 )</td>
<td>sandwiches</td>
</tr>
<tr>
<td>Bitonic Sorting</td>
<td>( (n - 1) \cdot n^{[25]} )</td>
<td>( n )</td>
<td>Unitonic permutations</td>
</tr>
<tr>
<td>Halving</td>
<td>( \binom{n}{(n/2)} )</td>
<td>( \binom{n}{(n/2)} )</td>
<td>balanced vectors</td>
</tr>
</tbody>
</table>

Table 2.1: Summary of results: sizes of conclusive sets for various functionalities.

Consider the second column of Table 2.1. For the first three rows, the corresponding conclusive sets are simply all the valid binary vectors, except the constant ones. The fact that these sets are conclusive is already known and references are given in the table.

\(^1\) ‘Bitonic sorting’ means sorting all the Bitonic vectors of a certain width. The term ‘Bitonic’ is defined in Section 2.2.1.
However, the fact that these sets are of minimal size is a contribution of this paper. Note that the last entry in this column, concerning halving, is significantly different from the first three entries. It is much smaller than the set of all binary vectors which are valid and non-constant. All the above binary conclusive sets share the following property. They are minimal in a very strong sense – each of them is a subset of any binary conclusive set for the same functionality.

Note that, unlike the binary case, the unrestricted conclusive sets of minimal cardinality are not unique; it will be apparent that the same functionality may have several conclusive sets of minimal cardinality that are substantially different.

Consider the ratio between the minimal size of binary conclusive set and the minimal size of unrestricted conclusive set. As discussed in Section 2.5, this ratio cannot exceed \( n - 1 \), for all functionalities addressed in this paper. We point to the two extreme cases of Bitonic sorting and halving. In the case of Bitonic sorting, this ratio is \( n - 1 \). On the other hand, for the functionality of halving, this ratio is 1; that is, no improvement is achieved by using unrestricted conclusive sets.

The main motivation for compact and elegant conclusive sets is simplification of the design and analysis of comparator networks. We know of two examples in which conclusive sets were used to construct new networks with useful properties.

The first example concerns fast Bitonic sorters of arbitrary width (not a power of two) whose depth is at most \( \lceil \log(n) \rceil + 3 \). Such networks are constructed in [17] using the conclusive set of Uniconclusive permutations introduced in Section 2.4.

The second example we know of concerns merging networks of minimal depth in which several of the outputs are accelerated. That is, they are generated much faster than the other outputs. Such networks are constructed in [21] using the elegant and compact conclusive set of sandwiches, presented in Section 2.4.1. A key lemma of this construction, which is proved using sandwiches, is the following lemma:

**Lemma 2.1.1.** Let \( a = \langle a_0, a_1, \ldots, a_{j-1} \rangle \) and \( b = \langle b_0, b_1, \ldots, b_{j-1} \rangle \) be two sorted sequences of length \( j \); let \( 0 < k < j \) and let

\[
c = \langle \max(a_0, b_{k-1}), \ldots, \max(a_{k-1}, b_0), \min(a_k, b_{j-1}), \min(a_{k+1}, b_{j-2}), \ldots, \min(a_{j-1}, b_k) \rangle
\]

. Then the sequence \( c \) is Bitonic.

This lemma can be proved using (a variant of) the 0-1 Principle but this leads to many special cases which need to be verified. On the other hand, as demonstrated in [21], sandwiches provide a compact proof, having only two symmetric cases.

An additional benefit of small conclusive sets concerns Black Box testing of the functionality of a given network. In such a test, all members of a conclusive set are fed into the network and the resulting vectors are examined. In this context, the conclusive sets should be as small as possible.
Previous work. The main application of the 0-1 principle is to simplify the design and proof of correctness of sorting and merging networks. We review some of the applications of the 0-1 principle from the literature. Miltersen et. al. [24] and Liszka and Batcher [3] used some variant of the 0-1 principle to prove the correctness of a certain merging network. Bender and Williamson [5] used it to prove structure theorems for recursively constructed merging networks. Batcher and Lee [2] used it to prove the correctness of a $k$-merger network whose input consists of $k$ sorted vectors of equal length. Nakatani et. al. [25] used it to prove the correctness of a Bitonic sorter.

Organization. This paper is organized as follows. In Section 2.2, comparator networks are formally defined and various functionalities of comparator networks are presented. In Section 2.3 the well known 0-1 principle for sorting networks is presented along with some variants. These variants enable extending the 0-1 principle to the functionalities presented in Section 2.2.1. In Section 2.4 we present smaller conclusive sets for each of these functionalities. In Section 2.5 we prove lower bounds on the sizes of binary and unrestricted conclusive sets. These lower bounds match the upper bounds presented in Section 2.4.

2.2 Comparator Networks

A comparator is a combinational device that sorts two keys. Namely, it has two incoming edges and it receives a key from each one of them. It has two outgoing edges of distinct types; a min edge and a max edge. It transmits the minimal key on the min edge and the maximal key on the max edge.

A comparator network (a.k.a. a network) is an acyclic network of these devices. See Figure 3.1. In this figure, comparators are denoted by circles, a Min edge is indicated by a hollow arrowhead, a Max edge is indicated by a solid arrowhead and an input edge is.
denoted by an open arrowhead. Such a network receives a vector (sequence of keys) via its input edges and produces a vector on its output edges.

Clearly, a network has the same number of input edges and output edges. To specify how an input sequence should be fed into the network, the input edges are arranged as a sequence. Similarly, the output edges of a network are arranged as a sequence to specify how the network’s output is interpreted as a sequence. Sometimes, the order of the input edges and the order of the output edges are implied from the drawing (e.g., from top to bottom).

These networks are useful for performing operations such as merging or sorting on keys – members of a certain ordered set $\mathcal{K}$. For most applications, the exact nature of the keys and their number is not important, but this paper is an exception. For most functionalities in question, a minimal conclusive set, for a network processing $n$ keys, requires $|\mathcal{K}| \geq n$. A similar phenomena is shown in [19]. That is, there are some interesting results which are sensitive to the size of $\mathcal{K}$. For definiteness, we henceforth assume that $\mathcal{K} = \mathbb{Q}$ where $\mathbb{Q}$ is the set of rational numbers.

### 2.2.1 Functionality

A **vector** $v = \langle v_0, v_1, \ldots, v_{n-1} \rangle$ is a sequence of keys. The **width** of $v$ is its length, $n$, and is denoted by $|v|$. Such a vector can be fed into a network if it is of the appropriate width. As said, the input edges of a network are arranged as a sequence; this specifies how a vector is fed into the network. Similarly, the output edges of a network are also arranged as a sequence; this specifies how the network’s output is interpreted as a sequence.

This work discusses four functionalities and each of them is associated with a certain set of valid input vectors. To this end, we define the following type of vectors. A vector $w = \langle w_0, \ldots, w_{n-1} \rangle$ is **ascending** (a.k.a. sorted) if $w_i \leq w_j$ whenever $i \leq j$. Similarly, $w$ is **descending** if $w_i \geq w_j$ whenever $i \leq j$. A vector is **ascending-descending** if it is a concatenation of an ascending vector and a descending vector. A vector is **Bitonic**\(^2\) if it is a rotation of an ascending-descending vector.

Note that valid inputs to merging networks are different from valid inputs of other types of networks; namely, they naturally consist of two sequences rather than a single one. Hence, we extend the concept of a vector. A vector may be not only a single sequence but also an ordered pair of sequences. A vector $v = \langle a, b \rangle$ is **Bisorted** if the two sequences, $a$ and $b$, are of equal width and are sorted. Note that the width of $v$ equals $|a| + |b|$. Bisorted vectors (of the appropriate width) are the valid input vectors of a merging network. By our definition, the input of a comparator network is a single sequence rather than two sequences. To overcome this, a Bisorted vector is combined, in a fixed manner, into a sequence.

---

\(^2\)The term ‘Bitonic’ was coined by Batcher [1] and we follow his terminology. We caution the reader that some authors use the same term with other meanings.
single sequence so it can serve as an input of a comparator network; this can be done, for example, by concatenation.

The four functionalities discussed in this paper are as follows:

**sorting:** A *sorting network* is a network that sorts all its input vectors.

**Bitonic sorting:** A *Bitonic sorter* is a network that sorts all its Bitonic input vectors.

**merging:** A *merging network* is a network that sorts every Bisorted vector.

**halving:** A *halver* receives an even number of keys, and separates them into two sets of equal size. One set contains the lowest keys and the other set contains the highest keys. To this end, the output edges are divided into two (equal size) sets. For example, this division may conform to the order of the output edges as follows. The initial half of the output edges transmit the lowest keys, while the second half of the output edges transmit the highest keys. Following this convention, we say that a vector of even width \( n \) is *halved* if every key in the first \( n/2 \) positions is lower or equal to any key in the last \( n/2 \) positions.

### 2.3 The 0-1 Principle

A 0-1 vector (binary vector) contains only the keys 0 and 1. The classical 0-1 Principle concerns sorting networks and is as follows:

*Theorem 3* (The 0-1 Principle, [14]). A comparator network is a sorting network if and only if it sorts every 0-1 vector (of the appropriate width).

The next lemma (Lemma 2.3.1) is a stronger variant of the 0-1 Principle that concerns a single vector rather than all the vectors. Namely, it says that a network sorts a vector \( v \) if and only if it sorts a certain set of 0-1 vectors associated with \( v \). The lemma uses the following notation.

**Definition 2.3.1.**

- A vector \( y \) is an image of a vector \( x \) (or \( x \) covers \( y \)) if \( |y| = |x| \) and \( x_i \leq x_j \) implies that \( y_i \leq y_j \), for every two indexes \( i \) and \( j \).

- A set of vectors \( X \) covers a set of vectors \( Y \) if every member of \( Y \) is covered by some member of \( X \).

For example, consider the vectors: \( a = \langle 1, 9, 5 \rangle, b = \langle 2, 9, 2 \rangle, c = \langle 4, 8, 4 \rangle \) and \( d = \langle 7, 7, 7 \rangle \). Each of these vectors covers all the following vectors. Among these vectors there is exactly one pair \((b\ and\ c)\) in which every vector covers the other vector.

The binary relation “\( y \) is an image of \( x \)” is reflexive and transitive but is neither symmetric nor antisymmetric, as demonstrated in the above examples. A vector \( y \) is a 0-1 image
of \( x \) if it is binary and an image of \( x \). Straightforward 0-1 arguments imply the following two lemmas.

**Lemma 2.3.1.** A comparator network sorts a vector if and only if it sorts all its 0-1 images.

**Lemma 2.3.2.** A comparator network halves a vector if and only if it halves all its 0-1 images.

We now state 0-1 principles for merging networks, Bitonic sorters, and halvers. These results are known, can be proved by straightforward 0-1 arguments and are used, for example, in [24],[25] and [14].

**Theorem 4.**

1. A network is a merging network if and only if it sorts every 0-1 bisorted vector.
2. A network is a Bitonic sorter if and only if it sorts every 0-1 Bitonic vector.
3. A network is a halver if and only if it halves every 0-1 vector.

Theorem 4 is already known; it follows from Lemmas 2.3.1, 2.3.2 and the following lemma.

**Lemma 2.3.3.** A vector is sorted/bitonic/halved if and only if all its 0-1 images are sorted/bitonic/halved.

**Proof.** The hardest case is the Bitonic one and we address only this case. The left to right implication is trivial. To prove the converse direction, we show that every non-Bitonic vector \( v \) has a non-Bitonic 0-1 image. It is not hard to see that \( v \) has a subsequence \( v' \) of length 4 that is not Bitonic. (For example, the minimal key and the maximal key of \( v \) are members of \( v' \); they divide \( v \) into two parts which are supposed to be ascending and descending. The other two keys of \( v' \) demonstrate that one of these parts is not in the correct order.) Next, pick a rational key \( q \) which is greater than two members of \( v' \) and is smaller than the other two.

Using \( q \) as a threshold, project the rational numbers into the set \( \{0, 1\} \) as follows. Keys lower than \( q \) are mapped to 0 and other keys are mapped to 1. This projection produces a 0-1 image of \( v \) which has a subsequence of length 4 that is not Bitonic; therefore, this 0-1 image of \( v \) is not Bitonic.

\[\square\]

### 2.4 Smaller Conclusive Sets

As said, each functionality is associated with a set of valid input vectors. For example, the Bitonic vectors are the valid inputs of Bitonic sorters.
Definition 2.4.1. A set of vectors $C$ is conclusive for sorting/merging/bitonic sorting/halving if every network that “works properly” for every input vector of $C$, “works properly” for every valid input vector.

By definition, for every functionality, the set of all valid input vectors is a conclusive set (e.g., the set of all Bitonic vectors is conclusive for Bitonic sorting). Our goal is to present minimal conclusive sets for these functionalities. We now define several sets of binary vectors, one for each of the first three functionalities.

- Let $B^{\text{sort}}$ be the set of all 0-1 vectors which are non-constant.
- Let $B^{\text{bitonic}} \subset B^{\text{sort}}$ be the set of all 0-1 vectors which are non-constant and Bitonic.
- Let $B^{\text{merge}}$ be the set of all non-constant 0-1 Bisorted vectors.

In other words, each of the sets $B^{\text{sort}}$, $B^{\text{bitonic}}$ and $B^{\text{merge}}$ is the set of all binary non-constant vectors that are valid for the corresponding functionality. Each of these sets is conclusive for the appropriate functionality. Section 2.4.3 presents a binary conclusive set for halving whose definition is substantially different from the above conclusive sets. In Section 2.5, we show that each of these four conclusive sets is minimal in a very strong sense – it is a subset of every binary conclusive set for the corresponding functionality. Lemma 2.3.1, and the fact that the above sets are conclusive, imply the following lemma which is our main tool for constructing even smaller conclusive sets.

Lemma 2.4.1. Any set of vectors that covers $B^{\text{sort}}/B^{\text{merge}}/B^{\text{bitonic}}$ is conclusive for sorting/merging/bitonic sorting.

2.4.1 Sandwiches for merging

In order to describe a small conclusive set for merging we use the following notation. A non-repeating vector is a vector in which each key appears at most once. A permutation is a non-repeating vector of length $n$ containing all the keys in the set $\{0, 1, \ldots, n-1\}$. Recall that a Bisorted vector is an ordered pair of ascending sequences of equal width. We now define Bisorted vectors of a special form called sandwiches.

Definition 2.4.2. A sandwich is a Bisorted vector $(x, y)$ which is a permutation and in which the range of the $x$ sequence is an interval.

For example the vector $((\langle 1, 2, 3, 4 \rangle, \langle 0, 5, 6, 7 \rangle)$ is a sandwich. The term “sandwich” follows from the fact that the vector can be sorted by inserting the first sequence consecutively in a certain place in the second sequence. Clearly, there are exactly $n/2 + 1$ sandwiches of width $n$. The following lemma is a straightforward observation.
Lemma 2.4.2. The set of sandwiches covers $B^{merge}$.

Lemmas 2.4.1 and 2.4.2 imply the following result:

Lemma 2.4.3 (The sandwich lemma). The set of sandwiches is conclusive for merging.

2.4.2 Unitonic vectors for Bitonic sorting

Recall that a Bitonic sequence is a rotation of an ascending-descending sequence. Similarly, we have the following definition:

Definition 2.4.3. A Unitonic vector is a rotation of an ascending sequence.

Clearly, for every $n$ there are $n$ Unitonic permutations of width $n$. The following lemma is a straightforward observation.

Lemma 2.4.4. The set of Unitonic permutations covers $B^{bitonic}$.

Lemma 2.4.1 and 2.4.4 imply the following result:

Lemma 2.4.5. The set of Unitonic permutations is conclusive for Bitonic sorting.

2.4.3 Balanced vectors for halving

Recall that the binary sets $B^{sort}$, $B^{merge}$ and $B^{bitonic}$, were shown to be conclusive for sorting, merging and Bitonic sorting. In this section we present the appropriate binary conclusive set for halving. To this end, we say that a 0-1 vector is balanced if it contains the same number of zeros and ones.

- Let $B^{half}$ be the set of balanced 0-1 vectors.

This section proves that $B^{half}$ is conclusive for halving. Later on (Section 2.5), $B^{half}$ is shown to be the minimal binary conclusive set for halving.

Agreeing vectors

To address the issue of halving we need an additional tool which was not needed for the other functionalities. Namely, the concept of agreement.

Definition 2.4.4. Two vectors, $x$ and $y$, agree if $|x| = |y|$ and there are no indexes $i$ and $j$ such that $x_i < x_j$ and $y_i > y_j$. 
For example, consider the vectors: $a = \langle 3, 3, 3 \rangle$, $b = \langle 2, 5, 5 \rangle$, $c = \langle 4, 4, 8 \rangle$ and $d = \langle 6, 7, 6 \rangle$. Every two of these vectors agree except $c$ and $d$. Note that the binary relation “$x$ and $y$ agree” is symmetric and reflexive. By the above examples, this relation is not transitive. Clearly, if one vector is an image of another vector then the two vectors agree. The inverse implication is not necessarily true. (E.g., the vectors $b$ and $c$ agree but neither one is an image of the other.) We have the following lemma.

**Lemma 2.4.6.** Suppose two vectors $x$ and $y$ agree. Let $N$ be a network of width $|x|$ and let $x'$ and $y'$ be the vectors produced by $N$ when receiving $x$ and $y$, respectively. Then $x'$ and $y'$ agree.

**Proof.** It is not hard to see that the lemma holds when $N$ has a single comparator. By induction on the number of comparators, the lemma holds for any network $N$.

**A conclusive set for halvers**

In order to construct a small conclusive set for halving, we use the following lemmas whose proofs are straightforward and, therefore, omitted.

**Lemma 2.4.7.** Every vector of even width agrees with some member of $B_{half}$.

For any $j$, let $0^j1^j$ be the vector composed of $j$ zeros followed by $j$ ones.

**Lemma 2.4.8.** A vector $x$, of length $n$, is halved if and only if $x$ and $0^n1^n$ agree.

The following theorem is the main result of this section.

**Theorem 5.** The set $B_{half}$ is conclusive for halving.

**Proof.** Let $N$ be a network that halves all members of $B_{half}$ of width $n$. Let $x$ be a vector of width $n$ and let $x'$ denote the output produced by $N$, given input $x$. By Lemma 2.4.7, $x$ agrees with some vector $y \in B_{half}$. The network $N$ halves all members of $B_{half}$; in particular, it halves $y$, producing the output $0^n1^n$. Lemma 2.4.6 imply that $x'$ and $0^n1^n$ agree. By Lemma 2.4.8, $x'$ is halved, and the lemma follows.

Theorem 5 and Lemma 2.3.2 imply the following result:

**Lemma 2.4.9.** Any set of vectors that covers $B_{half}$ is conclusive for halving.
2.4.4 Conclusive sets for sorting

This section proves the existence of a conclusive set of size \( \left\lceil \frac{n}{2} \right\rceil \) for sorting \( n \) keys. However, it does not actually constructs such a set. In fact, we do not know of a canonical and elegant conclusive set, of minimal size, for sorting.

Surprisingly, this section is based on the theory of partially ordered sets and on the seminal theorems of Dilworth \[8\] and Sperner \[23\]. We use the following notations. Let \( \mathbb{P} = (F, \preceq) \) denote a partially ordered set (poset). Two elements, \( a \) and \( b \) of \( F \), are comparable if \( a \preceq b \) or \( b \preceq a \). A chain (antichain) of \( \mathbb{P} \) is a subset \( Y \subset F \) such that any two distinct elements of \( Y \) are comparable (not comparable).

**Theorem 6** (Dilworth’s Theorem \[8\]). Let \( \mathbb{P} \) be a finite partially ordered set. Let \( K \) be the cardinality of the largest antichain of \( \mathbb{P} \) and let \( M \) be the minimal number of chains that cover \( \mathbb{P} \). Then \( K = M \).

Recall that, in the context of merging, a vector is an ordered pair of sequences. However, in this section, a vector is always a single sequence; namely, \( v = \langle v_0, v_1, \ldots, v_{n-1} \rangle \). A 0-1 vector of width \( n \) and a subset of \( \{0, 1, \ldots, n-1\} \) are two aspects of essentially the same object. In this section we do not distinguish between these aspects. That is, for a vector \( v \), the two phrases \( "v_i = 1" \) and \( "i \in v" \) are equivalent. Similarly, for two 0-1 vectors of the same width, \( u \) and \( v \), the phrases \( "u \subset v" \) and \( "u_i \leq v_i \) for every \( i \)”, are equivalent.

Let \( n \in \mathbb{N} \) be fixed in the following discussion. We focus on the poset \( \mathbb{P}^n = (\{0, 1\}^n, \subset) \). Let \( V^n \) be the set of unrestricted vectors of width \( n \). We next show that every chain is covered by some vector of \( V^n \). To this end, we use the following notations. Let \( X \) be a subset of \( \{0, 1\}^n \). For an index \( i \), let \( X[i] \) be the subset of \( X \) defined by \( X[i] = \{v \mid i \in v \in X\} \). Let \( \hat{X} \in V^n \) be defined by \( \hat{X}_i = |X[i]| \) for every \( i \). For example, let \( n = 5 \) and let \( C = \{\{3\}, \{0, 3, 4\}, \{0, 1, 3, 4\}\} \) be a chain of \( \mathbb{P}^n \). Then \( \hat{C} = \langle 2, 1, 0, 3, 2 \rangle \).

**Lemma 2.4.10.** Every chain \( C \) of \( \mathbb{P}^n \) is covered by \( \hat{C} \).

**Proof.** Let \( i \) and \( j \) be two indexes and consider the two sets \( C[i] \) and \( C[j] \). Since \( C \) is a chain, one of these sets contains the other. Referring to Definition 2.3.1 of cover, assume that \( \hat{C}_i \leq \hat{C}_j \). By the above argument, \( C[i] \subset C[j] \). In other words, for every \( v \in C, v_i = 1 \) implies \( v_j = 1 \). That is, \( v_i \leq v_j \); hence, \( \hat{C} \) covers \( v \). Since this holds for every \( v \in \hat{C} \), it follows that \( \hat{C} \) covers \( C \). \qed

Our construction is based on Sperner’s famous theorem.

**Theorem 7** (Sperner’s Theorem \[23\]). The largest antichain of \( \mathbb{P}^n \) is of size \( \left( \frac{n}{2} \right) \).

The following lemma is the main result of this section.

**Theorem 8.** There is a conclusive set of size \( \left( \frac{n}{2} \right) \) for sorting vectors of width \( n \).
Proof. By the theorems of Dilworth and Sperner, there are \( \binom{n}{\lceil n/2 \rceil} \) chains that cover \( \mathbb{P}^n \). By Lemma 2.4.10, each of these chain is covered by a single vector; hence, \( \{0, 1\}^n \) is covered by a set of \( \binom{n}{\lfloor n/2 \rfloor} \) vectors. By Lemma 2.4.1, this set is conclusive for sorting.

\[ \square \]

2.5 Lower Bounds for Conclusive Sets

This section shows that our binary conclusive sets and unrestricted conclusive sets are of minimal sizes. Moreover, it shows that our binary conclusive sets are minimal in a stronger sense – each of these sets is a subset of any binary conclusive set for the same functionality. Our main tool is the next lemma which provides, for any 0-1 vector, a network that identifies this vector in the following sense.

Lemma 2.5.1 (The identification lemma). For every 0-1 vector \( v \) which is not constant, there is a network that sorts all the 0-1 vectors of the appropriate width, except \( v \).

Proof. The desired network, \( N \), is depicted in Figure 2.2. All squares represent sorting networks of various width. The main idea behind the construction is as follows. The input vector, let us call it \( z \), is partitioned into two vectors, \( d \) and \( u \), such that the following holds. For most 0-1 vectors, the pair \( \langle d, u \rangle \) is a separation of \( z \) – namely, any key of \( d \) is smaller from or equal to any key of \( u \). In particular, for a 0-1 vector \( z \), the pair \( \langle d, u \rangle \) is not a separation of \( z \) exactly when \( z = v \). We refer to this functionality as ‘conditional separation’. A network that performs this ‘conditional separation’ can easily be extended to the desired network by sorting the vector \( d \), to produce the lowest keys, and sorting \( u \) to produce the highest keys.

The network \( N \) works as follows. Let \( L \) and \( H \) be the sets of indexes where \( v \) equals zero and one, respectively. Namely, \( L = \{ i \mid v_i = 0 \} \) and \( H = \{ i \mid v_i = 1 \} \). Let \( x \) and \( y \) be the subvectors of \( z \) composed of the keys in the positions of \( L \) and \( H \), respectively. First, \( N \) partitions \( z \) into the vectors \( x \) and \( y \). (Clearly, this is done without any comparators.) The vector \( x \) is separated (by a sorting network) into its highest key, denoted \( \max(x) \), and all the other keys, denoted \( \hat{x} \). Similarly, \( y \) is separated into its lowest key, denoted \( \min(y) \), and all the other keys, denoted \( \hat{y} \).

Next, the \( n-2 \) keys of \( \hat{x} \) and \( \hat{y} \) are combined into a single vector; this vector is separated (by a sorting network) into \( \langle d', u' \rangle \) where \( |d'| = |L| - 1 \) and \( |u'| = |H| - 1 \). Let \( d \) be the combined vector of \( d' \) and \( \min(y) \); similarly, let \( u \) be the combined vector of \( u' \) and \( \max(x) \). It remains to show that the partition of \( z \) into \( d \) and \( u \) is a ‘conditional separation’; that is, \( \langle d, u \rangle \) is a separation of \( z \) if and only if \( z \neq v \). We consider four cases according to the values of \( \max(x) \) and \( \min(y) \). It is not hard to see that the case of \( \max(x) = 0 \) and \( \min(y) = 1 \) holds exactly when \( z = v \).

- \( \max(x) = 0, \min(y) = 1 \) : By our construction, \( \max(x) \) is a member of \( u \) and \( \min(y) \) is a member of \( d \). This implies that \( \langle d, u \rangle \) is not a separation.
• \( \max(x) = 1, \min(y) = 0 \): These values, combined with the fact that \( \langle d', u' \rangle \) is a separation, imply that \( \langle d, u \rangle \) is a separation of \( z \).

• \( \max(x) = 1, \min(y) = 1 \): Since \( \min(y) = 1 \), it follows that \( \hat{y} \) is all ones and, therefore, \( u' \) is all ones. Since \( \max(x) = 1 \), it follows that \( u \) is all ones, implying that \( \langle d, u \rangle \) is a separation.

• \( \max(x) = 0, \min(y) = 0 \): This case is similar to the previous one.

Lemma 3.3.3 was sometimes mistaken [26] as proven by Rice [27]. Rice’s result does not refer to comparator networks but to a certain set of functions defined by topological means. As shown in [16, Section 3.1], Rice’s result is strictly weaker than Lemma 3.3.3.

Recall that, by our convention, a halved vector of width \( n \) has the \( \frac{n}{2} \) lowest keys in the first positions and the \( \frac{n}{2} \) highest keys in the last ones. Therefore, every sorted vector is halved. Furthermore, a 0-1 balanced vector is halved if and only if it is sorted. Hence, Lemma 3.3.3 has the following corollary.

\textbf{Lemma 2.5.2.} For every 0-1 balanced vector \( v \), there is a network that halves every 0-1 vector except \( v \).

Sections 2.3 and 2.4.3 show that the sets \( B^{\text{sort}} / B^{\text{merge}} / B^{\text{bitonic}} / B^{\text{half}} \) are conclusive for the corresponding functionality. Lemmas 3.3.3 and 2.5.2 imply that they are minimal in a very strong sense as follows:

\textbf{Lemma 2.5.3.} A set of 0-1 vectors is conclusive for sorting/merging/bitonic sorting/halving if and only if it contains \( B^{\text{sort}} / B^{\text{merge}} / B^{\text{bitonic}} / B^{\text{half}} \).
We now consider unrestricted conclusive sets. The following lemma states necessary and sufficient conditions for a set to be conclusive for each of the considered functionalities.

**Lemma 2.5.4.** A set is conclusive for sorting/merging/bitonic sorting/halving if and only if it covers $B_{\text{sort}}/B_{\text{merge}}/B_{\text{bitonic}}/B_{\text{half}}$.

**Proof.** Lemmas 2.4.1 and 2.4.9 provide the left to right implication. Consider the other direction. We focus on the Bitonic functionality and the proofs for the other functionalities are similar.

Assume that some 0-1 vector $z \in B_{\text{bitonic}}$ is not covered by a set of vectors $C$. By Lemma 3.3.3, there exists a network, $N$, that sorts all the 0-1 vector except $z$. Hence, it sorts all 0-1 vectors covered by $C$. By Lemma 2.3.1, $N$ sorts all vectors in $C$; however, $N$ does not sort $z$ and therefore, $N$ is not a Bitonic sorter. This implies that $C$ is not a conclusive set for Bitonic sorting.

The proof for halving is similar and is based on Lemma 2.5.2 rather than Lemma 3.3.3.

Clearly, any vector of length $n$ covers at most $n - 1$ non-constant 0-1 vectors. This fact and Lemma 2.5.4 imply a trivial lower bound on the size of an unrestricted conclusive set. However, these lower bounds are usually not tight, as shown shortly.

We next apply Lemma 2.5.4 to show that the unrestricted conclusive sets presented in Section 2.4 are of minimal size. Clearly, a vector can cover a number of 0-1 vectors; however, there is a class of 0-1 vectors such that every vector covers at most one member of this class. To this end, we extend the term ‘balanced’ to vectors of odd width as follows.

A 0-1 vector of odd width, $v$, is **balanced** if it has exactly $\lceil |v|/2 \rceil$ ones. Clearly, every vector (of odd or even width) covers at most one 0-1 balanced vector. It is not hard to see that:

**Lemma 2.5.5.** For every appropriate $n$, the sets $B_{\text{sort}}, B_{\text{merge}}, B_{\text{bitonic}}$ and $B_{\text{half}}$ have $\binom{n}{\lceil n/2 \rceil}, n/2 + 1, n$ and $\binom{n}{\lfloor n/2 \rfloor}$ balanced vectors of width $n$, respectively.

Section 2.4 presents, for every functionality and for every width $n$, a conclusive set as follows.

- **Sorting:** a set of $\binom{n}{\lfloor n/2 \rfloor}$ vectors.
- **Merging:** the set of $n/2 + 1$ sandwiches.
- **Bitonic sorting:** the set of $n$ Unitonic permutations.
- **Halving:** the set of $\binom{n}{\lceil n/2 \rceil}$ balanced 0-1 vectors.

---

3For the functionalities of merging and halving, $n$ is required to be even.
Note that, in the case of sorting, a conclusive set was not constructed; only its existence was proven. In the case of halving the unrestricted conclusive set is, in fact, binary.

Lemmas 2.5.4 and 2.5.5, and the fact that any vector covers at most one balanced 0-1 vector, imply the following theorem:

**Theorem 9.** Each of the unrestricted conclusive sets in the above list, for the functionalities of sorting/merging/bitonic sorting/halving, is of minimal cardinality.

By Lemma 2.5.3, all the functionalities considered in this work have unique 0-1 conclusive sets of minimal cardinality. However, this is not the case for unrestricted conclusive sets. Namely, the same functionality may have several unrestricted conclusive sets of minimal cardinality that are substantially different. For example, consider the functionality of Bitonic sorting. As said, the set of Unitonic permutations is conclusive for this functionality. Next consider the set of rotations of descending permutations. This set covers $B_{bitonic}$ and, by Lemma 2.4.1, it is conclusive for Bitonic sorting.
Chapter 3

Accelerating certain outputs of merging and sorting networks

Tamir Levi, Technion, Haifa
Prof’ Ami Litman, Technion, Haifa

This work studies comparator networks in which several of the outputs are accelerated. That is, they are generated much faster than the other outputs, and this without hindering the other outputs. We study this acceleration in the context of merging networks and sorting networks.

The paper presents a new merging technique, the Tri-section technique, that separates, by a depth one network, two sorted sequences into three sets, such that every key in one set is smaller or equal to any key in the following set. After this separation, each of these sets can be sorted separately, causing the above acceleration of certain outputs.

An additional contribution of this paper concerns the well-known 0-1 Principle [14]. This principle is a powerful tool that simplifies the construction and analysis of comparator networks. The paper demonstrates that, in some cases, there is a better tool to achieve the same goal. In the case at hand, this new tool simplifies one of our proofs by having fewer special cases than the classical 0-1 Principle.

A second additional contribution concerns Batcher’s merging techniques. It was shown in [16] that all published merging networks, whose width is a power of two, are a natural generalization of Batcher’s odd-even merging network. All these published merging networks are of minimal depth and have no degenerate comparators. This raises the following question. Is there a merging network, having the above properties, that is not a natural generalization of Batcher’s odd-even
merging network? The Tri-section technique provides a positive answer to this question.

### 3.1 Introduction

We study comparator networks in which several of the outputs are accelerated. That is, they are generated much faster than the other outputs, and this without hindering the other outputs. Namely, for every $0 < k \leq n$, we present a merging network of minimal depth that merges two sorted sequences of length $n$ into a single sorted sequence. This merging network produces either the lowest $k$ keys or the highest $k$ keys after a delay of $\lceil \log(k) \rceil + 1$ comparators. Building on that, we construct, for every $0 < k < n$, an $n$-key sorting network that accelerates its $k$ lowest or its $k$ highest outputs. This sorting network is a merge-sort network and has a minimal depth among these networks. Namely, its depth is $\lceil \log(n) \rceil \cdot \lceil \log(2n) \rceil$, the same depth as the Batcher merge-sort networks [1]. However, in contrast to the Batcher merge-sort networks which may accelerate only the first and last outputs, our merge-sort networks accelerates either the $k$ lowest keys or $k$ highest keys to a delay of less than $\lceil \log(n) \rceil \cdot \lceil \log 2k \rceil$ comparators.

The paper presents a new merging technique, the Tri-section technique, that separates, by a depth one network, two sorted sequences into three sets, such that every key in one set is smaller or equal to any key in the following set. After this separation, each of these sets can be sorted separately and this leads to the desired acceleration. The idea of separating the input into two sets is known and is used, for example, in the Bitonic sorter of Batcher [1]; however, to the best of our knowledge, separation into three sets as above is novel.

To put our results in context, let us compare the acceleration of our networks with the acceleration of other well-known merging networks – The Bitonic sorter and the odd-even merging network, both of Batcher [1]. The Bitonic sorter has no accelerated outputs at all; all outputs have exactly the same delay. On the other hand, the odd-even merging network has only two accelerated outputs, the first and the last ones whose delay is exactly one. All other outputs have the same delay.

To the best of our knowledge, the idea of accelerating certain outputs was never addressed. The only prior work which is somewhat similar to our work concerns selectors. A $(k, n)$-selector is a network that separates a set of $n$ keys into the lowest $k$ and the other keys. Fast selection leads to a sorting network that accelerate certain outputs, as follows: First, the $k$ lowest keys are separated from the other keys. Afterwards, each set is sorted separately. Yao presented a $(k, n)$-selector which is efficient when $k$ is constant and $n$ is very large. This selector can be extended into a sorting network that accelerates its lowest

---

1 When $n$ is a power of two, both the lowest $k$ keys and the highest $k$ keys can be accelerated.

2 A merge-sort network is a sorting network which operates as follows. The input is arbitrarily divided into two sets of (almost) equal size and each set is recursively sorted; the two sorted sequences are then merged.
outputs; however, the depth of the resulting sorting network exceeds the minimal depth of a merge-sort network. Our network accelerates the lowest \( k \) outputs while its depth is minimal among merge-sort networks.

Our paper has several additional contributions. The first one concerns the well-known 0-1 Principle [14]. This principle is a powerful tool that simplifies the construction and analysis of comparator networks. The paper demonstrates that, in some cases, there is a more convenient tool to achieve the same goal. In the context of merging, we use a small and elegant set of vectors which constitute a **conclusive set** [10]; namely, a network is a merging network if and only if it sorts this set. This tool simplifies our proof by having fewer special cases than the classical 0-1 Principle.

The second additional contribution concerns Batcher’s merging techniques. Batcher’s odd-even merging technique [1] works as follows: Each of the input sequences is partitioned into its even part and its odd part. The even part of one sequence is merged with the even part of the other sequence recursively and similarly, the odd parts are merged. Finally, the two resulting sequences are merged into a single sorted sequence by a depth one network.

A slight variant of this method, due to Knuth [14, pp 231] and Leighton [15, pp 623], recursively merges the even part of each input sequence with the odd part of the other sequence. Again, the resulting two sorted sequences can be merged by a depth one network. We refer to the family of networks produced by allowing each of the above two variants anywhere in the recursion process as **Batcher merging networks**. It was shown in [16] that all published merging networks, whose width is a power of two, are members of this family. All these merging networks are of minimal depth and have no degenerate comparators. (A degenerate comparator has a fixed incoming edge whose value is always greater or equal to the value on the other incoming edge, for every valid input of the network.) The above fact arise the following question :

**Question 10.** Are the Batcher merging networks the only merging networks with the following properties:

1. Their width, \( 2n \), is a power of two.
2. Their depth is minimal – \( \log(2n) \).
3. They have no degenerate comparators.

---

\(^3\)A degenerate comparator is a special case of a **redundant** comparator, studied in [16]. A comparator or a set of comparators is **redundant** if it can be removed from the network without disturbing the network’s functionality. Clearly, a degenerate comparator is redundant but not the other way around. The above reference to a redundant set of comparators, rather than to a single redundant comparator, is crucial due to the following result of [16]. A merging network or a sorting network may have a set of redundant comparators while any single comparator can not be removed without disturbing the network’s functionality.
The Tri-section technique provides a negative answer to this question, as shown in Section 3.5.

Another question, which remains open, concerns accelerating all the outputs of a merging network, each to a delay that is close to the trivial reachability bound of this output. This bound is due to the fact that the $j$ lowest (or highest) output may come from each of certain $2j$ input edges. Therefore, our question is:

**Question 11.** For any $n$ (or arbitrary large $n$), is there a merging network of width $2n$ that, for every $j < n$, accelerates the $j$ lowest output and the $j$ highest output to a delay of $\log(j) + o(\log(j))$?

### 3.2 Preliminaries

The concept of comparator networks is well-known and an example is depicted in Figure 3.1. A comparator (represented by a circle) receives two keys via its two incoming edges. The comparator sorts these keys; it transmits the minimal one on the outgoing Min edge (indicated by a hollow arrowhead) and the maximal key on the outgoing Max edge (indicated by the solid arrowhead). The network’s input edges are indicated by an open arrowhead.

The network of Figure 3.1 is in fact a merging network. Its input are two sorted sequences, each of width 2 and its output is a sorted sequence of width 4. Keys enter the network through its input edges and exit the network through its output edges. We name the input edges to denote how the input is fed into the network. In such a network, one input sequence enters the edges $\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_{n-1}$ and the other input sequence enters the edges $\hat{b}_0, \hat{b}_1, \ldots, \hat{b}_{n-1}$. Similarly, output edges are named $\hat{o}_0, \hat{o}_1, \ldots, \hat{o}_{2n-1}$ to denote how the output keys are assembled into a sequence. Namely, the output sequence $\hat{o} = (o_0, o_1, \ldots, o_{2n-1})$ is composed of the values on these edges, in that order.

![Figure 3.1: A merging network of width 4 and depth 2.](image)

The *width* of a network $N$ is the number of its input edges which clearly equals the number of its output edges. Let $e$ be an edge of $N$. The *depth* of $e$, denoted $d(e)$, is the
length of the longest path that ends in the tail of \( e \) (i.e., the path does not include \( e \)). Hence \( d(e) = 0 \), for every input edge \( e \). The depth of \( N \), denoted \( d(N) \), is the maximal depth of the edges of \( N \). In the network, \( M \), of Figure 3.1, \( d(M) = d(\hat{o}_1) = d(\hat{o}_2) = 2 \) and \( d(\hat{o}_0) = d(\hat{o}_3) = 1 \).

A sequence of keys \( \langle x_0, x_1, \ldots, x_{n-1} \rangle \) is denoted by \( \vec{x} \); the width of this \( \vec{x} \), denoted \( |\vec{x}| \), is \( n \). A bisequenced vector is a pair of sequences of equal width and is denoted by \( \langle \vec{a}, \vec{b} \rangle \). Naturally, \( |\langle \vec{a}, \vec{b} \rangle| = 2|\vec{a}| = 2|\vec{b}| \). A bisorted vector is a bisequenced vector \( \langle \vec{a}, \vec{b} \rangle \) in which both \( \vec{a} \) and \( \vec{b} \) are sorted. Such a vector is a valid input to a merging network of the appropriate width.

Let \( \mathcal{K} \) be the set of optional keys. Usually the cardinality of \( \mathcal{K} \) is insignificant, as long as it is greater than one, and this paper is no exception. This fact is reflected by the 0-1 principle. Surprisingly, there are some properties of networks which depend on the size of \( \mathcal{K} \) as shown in [19].
3.3 The Asymmetric Tri-section Merging Technique

This paper presents two Tri-section merging techniques. As said, they are based on separating, by a depth one network, a Bisorted vector of width $2n$ into three sequences, $\vec{x}, \vec{y}$ and $\vec{z}$, such that every key in one set is smaller or equal to any key in the next set. This allows us to sort each of these sequences separately. In all our techniques the resulting merging network is of a minimal depth. Furthermore, the sequences $\vec{x}, \vec{y}$ and $\vec{z}$ are sorted by networks of depth $\lceil \log(|\vec{x}|) \rceil$, $\lceil \log(|\vec{y}|) \rceil$ and $\lceil \log(|\vec{z}|) \rceil$, respectively.

In this section we present the Asymmetric Tri-section technique in which $|\vec{x}| = k$, $|\vec{y}| = n$ and $|\vec{z}| = n - k$, where $k$ is an arbitrary number smaller or equal to $n$. The technique is called “Asymmetric” in contrast to the “Symmetric” variant in which $|\vec{x}| = |\vec{z}|$.

The depth one network performing the Asymmetric Tri-section, with these parameters, is called $T^{k,n}$. Figure 3.2 presents the network $T^{5,11}$. In this figure, a comparator is denoted as in Figure 3.3. Namely, it contains two horizontal edges: a $\text{Min}$ edge and a $\text{Max}$ edge, connected by a diagonal line. The name of the edges entering this comparator are written on the diagonal line while the name of the edges coming out of it are written on the edges. (See Figure 3.3.) The general network, $T^{k,n}$, naturally follows the format of Figure 3.2 and a formal definition is omitted. Note that the network $T^{0,n}$ is identical to the first stage of Batchet’s Bitonic sorter [1]. Hence, in some sense, the Tri-section technique is a generalization of Batchet’s technique.

![Figure 3.2: The network $T^{5,11}$.](image)

![Figure 3.3: A comparator receiving $a_0$ and $b_4$ and producing $y_0$ and $x_4$.](image)
Let $\bar{T}^{k,n}$ denote the mapping performed by the network $T^{k,n}$. That is, $\bar{T}^{k,n}((\vec{a}, \vec{b})) = (\vec{x}, \vec{y}, \vec{z})$, where $\vec{x}$, $\vec{y}$ and $\vec{z}$ are the sequences generated by $T^{k,n}$ when it receives the input vector $(\vec{a}, \vec{b})$. To study $T^{k,n}$ we name several types of vectors. A sequence $\vec{x}$ is sorted (ascending) if $x_i \leq x_j$ whenever $i \leq j$. Similarly, $\vec{x}$ is descending if $x_i \geq x_j$ whenever $i \leq j$. A sequence is ascending-descending if it is a concatenation of an ascending sequence followed by a descending sequence. Similarly, a concatenation of a descending sequence followed by an ascending sequence is called descending-ascending. Note that either of the sequences may be empty; therefore, ascending sequences and descending sequences are both ascending-descending and descending-ascending. A sequence is Bitonic if it is a rotation of an ascending-descending sequence. A comparator network is an ascending-descending sorter if it sorts all ascending-descending sequences. Similarly, we define descending-ascending sorter and Bitonic sorter.

A powerful tool to study merging networks, similar to the 0-1 Principle, is the set of Sandwich vectors, presented in [10]. As demonstrated in the proof of Lemma 3.3.2, their simple and elegant form simplifies the analysis of merging networks and leads to fewer special cases than the traditional 0-1 Principle. For the sake of sandwiches we assume that $K = N$. A sandwich of width $2n$ is a Bisorted vector $(\vec{a}, \vec{b})$ in which every member of the interval $[0, 2n)$ appears exactly once and the range of the $\vec{a}$ sequence is an interval. The term “sandwich” follows from the fact that the vector can be sorted by inserting the sequence $\vec{a}$ consecutively in a certain place in the sequence $\vec{b}$. The sandwich technique is based on the following lemma:

**Lemma 3.3.1** (The sandwich Lemma [10]). A network is a merging network if and only if it sorts all sandwiches.

The following lemma is the keystone of the Asymmetric Tri-section technique.

**Lemma 3.3.2.** Let $(\vec{a}, \vec{b})$ be a Bisorted vector of width $2n$, let $k \in [0, n)$ and let $(\vec{x}, \vec{y}, \vec{z}) = \bar{T}^{k,n}((\vec{a}, \vec{b}))$. Then

1. $|\vec{x}| = k$, $|\vec{y}| = n$, $|\vec{z}| = n - k$.

2. Every key in $\vec{x}$ is smaller or equal to any key in $\vec{y}$ and every key in $\vec{y}$ is smaller or equal to any key in $\vec{z}$.

3. $\vec{x}$ is ascending-descending, $\vec{z}$ is descending-ascending.

4. $\vec{y}$ is Bitonic.

---

4This term was coined by Batcher [1] and we follow his terminology. We caution the reader that some authors use the term “Bitonic” with other meanings.
Proof. Statement (1) follows directly from our construction. Statement (2) is straightforward and its proof omitted. The fact that \( \vec{x} \) and \( \vec{z} \) are Bitonic is proved by Batcher [1, Appendix B], in a slightly different context. In order to show that \( \vec{x} \) is ascending-descending it suffices to show that the minimal key of \( \vec{x} \) is either in the first position or last position. This clearly follows from the fact that the key in question is either \( a_0 \) or \( b_0 \). The fact that \( \vec{z} \) is descending-ascending is proved analogously and Statement (3) follows.

The hard part of this proof is Statement (4). To use the 0-1 Principle or the Sandwich Lemma we need the network in question to be a merging network. To this end, we extend the network \( T^{k,n} \) into a network \( M \) as follows: The sequence \( \vec{x} \) enters an arbitrary ascending-descending sorter, the sequence \( \vec{y} \) enters an arbitrary Bitonic sorter and the sequence \( \vec{z} \) enters an arbitrary descending-ascending sorter. We now prove that \( M \) is a merging network. This fact can be proved using (a variant of) the 0-1 Principle but this leads to many special cases which need to be verified. On the other hand, sandwiches lead to a proof having only two symmetric cases. Therefore, we next assume that the input \( \langle \vec{a}, \vec{b} \rangle \) is a sandwich and show that in this case the sequence \( \vec{y} \) is Bitonic.
Note that a sandwich vector $\langle \vec{a}, \vec{b} \rangle$ is determined by the key $a_0$. There are two (overlapping) cases; either $a_0 \leq k$ or $a_0 \geq k$. The two cases are similar and we focus on the first. Figure 3.4 depicts the network $T^{5,11}$ processing the sandwich with $a_0 = 2$. The initial part of $\vec{y}$, having $k - a_0$ keys, comes from $\vec{b}$ in reverse order. Hence, this initial part of $\vec{y}$ is descending. The rest of $\vec{y}$ comes from $\vec{a}$ in the natural order; hence, this part is ascending.

To summarize, in the case of $a_0 \leq k$, $\vec{y}$ is Bitonic. In the other case, where $a_0 \geq k$, the sequence $\vec{y}$ is ascending-descending; hence, $\vec{y}$ is Bitonic also in this case. This and the Sandwich Lemma imply that $M$ is a merging network.

![Figure 3.4](image-url)

Figure 3.4: The network $T^{5,11}$ processing the sandwich with $a_0 = 2$.

If we were just to prove that $M$ is a merging network, then the proof would have ended here. However, our lemma is stronger – it says that $\vec{y}$ is always Bitonic, for any Bisorted input. To prove that, assume for a contradiction that $\vec{y}$ is not Bitonic. By the following Lemma 3.3.4 (whose proof is not dependent of the current lemma), there exists a Bitonic sorter that does not sort $\vec{y}$. As said, $M$ processes $\vec{y}$ using an arbitrary Bitonic sorter. In particular, this Bitonic sorter could be the one that does not sort $\vec{y}$. This contradicts the fact that the entire network $M$ is a merging network.

Our goal now is to show that for every non-Bitonic vector there exists a Bitonic sorter which does not sort it. To this end, we use the following result of [10]. For any 0-1 vector there is a network that identifies it in the following sense.

**Lemma 3.3.3** (The identifying lemma, [10]). For every 0-1 vector $v$ which is not constant, there is a network that sorts all the 0-1 vectors of the appropriate width, except $v$. 


Lemma 3.3.4. For every non-Bitonic sequence there is a Bitonic sorter that does not sort it.

Proof. We say that a vector $\vec{u}'$ is a binary image of a vector $\vec{u}$ if and only if there exists a monotonic function $f : K \rightarrow \{0, 1\}$ such that $u'_i = f(u_i)$ for every $i$. It is not hard to see that a vector is Bitonic if and only if all its binary images are Bitonic. Let $\vec{v}'$ be a non-Bitonic vector and let $\vec{v}$ be a non-Bitonic binary image of $\vec{v}$.

Clearly, $\vec{v}'$ is not constant. By Lemma 3.3.3, there exists a network $N$ of width $|\vec{v}'|$ that sorts all binary vectors except $\vec{v}'$. By a straightforward 0-1 argument, a network sorts a vector if and only if it sorts all its binary images. Since all binary images of Bitonic vectors are Bitonic it follows that $N$ is a Bitonic sorter. Since $N$ does not sort $\vec{v}'$ it does not sort $\vec{v}$.

Returning to the Tri-section technique, recall that our goal is to construct a merging network of minimal depth in which the sequences $\vec{x}, \vec{y}$ and $\vec{z}$ are sorted by networks of depth $\lceil \log(|\vec{x}|) \rceil, \log(|\vec{y}|)$ and $\lceil \log(|\vec{z}|) \rceil$ respectively. To this end, we use pruning ([11], [31], [16, Section 4.2.2]) to reduce the width of certain comparator networks. This technique is based on the concept of a degenerate comparators, as defined in the introduction. Pruning, in the context of merging networks, is studied in [16] and can be applied as follows. Several consecutive input edges at the top of the input sequences are fed with the fictive values of $+\infty$ while the rest of the inputs are fed with real keys. Clearly, the pathes of the fictive values are fixed – they do not depend on the values of the real keys. Any comparator that is on such a path is degenerate and can be removed without effecting the network’s functionality. The resulting network, that processes no fictive values, is a merging network of a smaller width. Returning to our $\vec{x}, \vec{y}$ and $\vec{z}$, we first consider the case where $n$ is a power of two. In this case, the vector $\vec{y}$ is Bitonic and its width is a power of two. Such a vector can be sorted by Batcher’s Bitonic sorter [1], whose depth is $\log(n)$.

Concerning the sequence $\vec{x}$, recall that this sequence is not only Bitonic, but also ascending-descending. Such a sequence can be expanded into a Bitonic sequence of a desired width by adding fictive keys of value $-\infty$ at the beginning (or the end) of the sequence. This implies that any wide enough Bitonic sorter can be pruned into an ascending-descending sorter of a smaller given width. The depth of the resulting network is clearly not greater than the depth of the original one. Therefore, $\vec{x}$ can be sorted by a network of depth $\lceil \log(|\vec{x}|) \rceil$. By symmetry, the sequence $\vec{z}$ can be sorted in a similar manner.

Next consider the case where $n$ is not a power of two. Note that in this case $|\vec{y}|$ is not a power of two. One may assume that a wider Bitonic sorter can be pruned into a Bitonic sorter of the desired width; however, as shown in [18], the minimal depth of a Bitonic sorter is not monotonic in the width of the input; therefore, such pruning is impossible.
The problem is solved as follows. Let \( n' = 2^{\lceil \log(n) \rceil} \) be the first power of two following \( n \). Let \( M' \) be the merging network of width \( 2n' \) generated by the asymmetric construction in which the \( k \) lowest outputs are accelerated to a depth of \( \lceil \log(k) \rceil + 1 \). The network \( M' \) is pruned into a width \( 2n \) merging network as discussed above. This results in a merging network \( N \) of width \( 2n \) and of depth \( \lceil \log(2n) \rceil \) whose \( k \) lowest outputs are accelerated to a delay of \( \lceil \log(k) \rceil + 1 \) comparators.

Our construction possesses an additional important attribute which enables the concatenation of several such accelerating merging networks into an accelerating sorting network. This attribute relates to ‘restricted reachability’ as follows. We say that an edge \( e \) of a merging network is reachable only from the \( k \) lowest (highest) inputs if there is no path to \( e \) from an input edge which is not one the \( k \) lowest (highest) input edges of one of the input sequences. The construction of this section is summarized in the following lemma.

**Lemma 3.3.5.** For every \( k < n \) there is merging network of width \( 2n \) and of depth \( \lceil \log(2n) \rceil \) in which each of the \( k \) lowest (highest) outputs is accelerated to a depth of \( \lceil \log(k) \rceil + 1 \) comparators and is reachable only from the \( k \) lowest (highest) inputs.
3.4 The Symmetric Tri-section Merging Technique

Another Tri-section technique that accelerates certain outputs is presented in this section. As said, the Tri-section technique separates, by a depth one network, a Bisorted input into three sets, $\vec{x}$, $\vec{y}$ and $\vec{z}$ such that every key in one set is smaller or equal to any key in the following set. We first consider the case where the width of the input, $2n$, and the number of accelerated outputs, $k$, are powers of two. In this case the Symmetric Tri-section satisfies $|\vec{x}| = |\vec{z}| = k$. Hence, the network accelerates both the lowest $k$ outputs and the highest $k$ outputs to a delay of $\log(2k)$ comparators.

The symmetric technique is based on the Bitonic sorting technique of Nakatani et al. [25] which considers the keys to be arranged in a matrix. To this end, we denote a matrix of keys by $\vec{m}$. Their technique is based on the following lemma.

**Lemma 3.4.1 (The matrix technique [25]).** Let $\vec{b}$ be a Bitonic sequence of width $j \cdot k$ and $\vec{m}$ be the $j \times k$ matrix having the $\vec{b}$ sequence in a row major fashion. Then:

1) Every row in $\vec{m}$ is a Bitonic sequence.
2) Every column in $\vec{m}$ is a Bitonic sequence.

Let $\vec{m}'$ be derived from $\vec{m}$ by sorting each column separately. Then:
3) Every row in $\vec{m}'$ is a Bitonic sequence.
4) Every element of every row in $\vec{m}'$ is smaller or equal to any element of the next row.

The matrix technique of Nakatani et al. to sort Bitonic sequences of length $j \times k$ is composed of three stages:

**Stage 1:** Arrange the Bitonic sequence in a $j \times k$ matrix in a row major fashion.

**Stage 2:** Independently, sort every column by a Bitonic sorter.

**Stage 3:** Independently, sort every row by a Bitonic sorter.

Following those stages the resulting matrix is sorted in a row major fashion.

Note that the matrix technique does not require that $j$ and $k$ be powers of two; however, we use it only under this restriction. Assume that the Bitonic sorters used in stage (2) and stage (3) are of minimal depth. The depth of the entire network is $\log(j) + \log(k)$ which is minimal. As shown in [16], for every $n$, a power of two, there is a unique $n$-key Bitonic sorter of minimal depth. This implies that, when the width is a power of two, the network of Nakatani et al. is identical to Batcher’s [1] Bitonic sorter. Yet, even in this case, Nakatani’s technique sheds a new light on Batcher’s Bitonic sorter.
Note that the parameter $k$ in the technique of Nakatani et al. and the parameter $k$ of our technique refer to the same number; namely, using a $j \times k$ matrix as per Lemma 3.4.1, we accelerate the highest $k$ outputs and the lowest $k$ outputs. Since we are only interested in merging and not in Bitonic sorting, we can perform Stage (2) in a special manner. When the Bisorted input is turned into a Bitonic sequence and arranged in the above matrix, every column in the matrix $\vec{m}$ is not only Bitonic, but in-fact Bisorted; hence, it can be sorted by any merging network. There are many merging networks [16, Section 6] of minimal depth that produce the lowest key and highest key after a delay of a single comparator (e.g., Batcher’s [1] odd-even merging network); therefore, Stage (2) can be performed by $k$ such merging networks, working in parallel. Stage (3) can be performed by $j$ Batcher Bitonic sorters, working in parallel. The depth of these Bitonic sorters is minimal $- \log(k)$; therefore, this construction accelerates the lowest $k$ outputs and highest $k$ outputs to a delay of $\log(k) + 1$ comparators.

Let $\bar{M}$ be the network performing Stage (2). By Statement (4) of Lemma 3.4.1, $\bar{M}$ Tri-sections the keys into three sets: the first row, the last row and all the rest. Moreover, the very same Tri-section is performed by a subnetwork of $\bar{M}$ of depth one; hence this technique is a Tri-section technique. Our construction not only accelerates the required inputs but also has a restricted reachability as summarized in the following lemma.

**Lemma 3.4.2.** For any $k < n$, both powers of two, there is merging network of width $2n$ and of depth $\log(2n)$ in which:

- Each of the $k$ lowest outputs is accelerated to a delay of $\log(2k)$ comparators and is reachable only from the lowest $k$ inputs.

- Each of the $k$ highest outputs is accelerated to a delay of $\log(2k)$ comparators and is reachable only from the highest $k$ inputs.

Next consider the case where $k$ is not a power of two. As in the previous section, instead of accelerating $k$ outputs we accelerate $k' = 2^{\lceil \log k \rceil}$ outputs. Note that in this construction, each accelerated output is reachable from $k' = 2^{\lceil \log k \rceil}$ (rather than $k$) extreme inputs.

Finally, consider the case where the network’s width, $2n$, is not a power of two. In this case, we do not know how to accelerate both the highest $k$ and the lowest $k$ keys, simultaneously; in fact, we do not know if such acceleration is possible. We do know how to accelerate either the smallest $k$ outputs or the highest $k$ outputs. This is accomplished by pruning a network whose width is a power of two. The following Lemma (similar to Lemma 3.3.5) summarizes this case.
**Lemma 3.4.3.** For any \( k < n \) there is merging network of width \( 2n \) and of depth \( \lceil \log(2n) \rceil \) in which each of the \( k \) lowest (highest) outputs is accelerated to a delay of \( \lceil \log k \rceil + 1 \) comparators and is reachable only from the lowest (highest) \( 2^{\lceil \log k \rceil} \) inputs.

### 3.5 A counterexample

As shown in [16], all published merging networks (whose width is a power of two) are Batcher merging networks. Namely, they are constructed by a straightforward generalization of Batcher’s odd-even technique. The depth of all these merging networks is minimal. This raises the following question:

**Question 1** Are the Batcher merging networks the only merging networks with the following properties:

1. Their width, \( 2n \), is a power of two.

2. Their depth is minimal – \( \log(2n) \).

3. They have no degenerate comparators.

The answer to this question is no. The Tri-section technique can generate a counterexample based on the fact that when \( \lceil \bar{x} \rceil \) is small w.r.t. \( n \), the sequence \( \bar{x} \) can be sorted in an arbitrary manner (by a network of excessive depth) while maintaining the minimal depth of the entire merging network. We next present such a network for any \( n \geq 8 \), a power of two.
Figure 3.5: An ascending-descending sorter.

Our construction starts with the network $T^{3,n}$ that produces the three sequences $\vec{x}$, $\vec{y}$ and $\vec{z}$. The sequence $\vec{x}$ is sorted by the network depicted in Figure 3.5 which has no degenerate comparators. The sequences $\vec{y}$ and $\vec{z}$ are sorted by any minimal depth network as per Section 3.3. The resulting merging network, $M$, satisfies the three conditions of Question 10. (If $M$ has degenerate comparators, they should be removed.) The network $M$ is not a Batcher merging network since it has a comparator, $c$, with the following property. Of the two edges exiting $c$, one is the output edge $\hat{o}_2$ and the other is not an output edge. This is never the case in a Batcher merging network.

The above construction can be extended to yield a merging network of minimal depth which does not follow the “divide and conquer” paradigm. Let $k = |\vec{x}|$ be large enough and still much smaller than $n$. Then the sequence $\vec{x}$ can be sorted using a network which is clearly not of the above paradigm. Two such examples are Knuth’s bubble-sort network and Knuth’s odd-even transposition sort [14, pp 223,241]. This construction may produce degenerate comparators that can be removed without effecting the network’s functionality. This implies the existence of a minimal depth merging network that has no degenerate comparators and has an arbitrary large subnetwork lacking any recursive structure.

### 3.6 Accelerating Sorting Networks

Building on the merging networks introduced in previous section, we now utilize the classical merge sort algorithm to construct a sorting networks that accelerates certain outputs. Clearly, the depth of a merge-sort network is at least $\frac{\lceil \log(n) \rceil \cdot \lceil \log(2n) \rceil}{2}$ and due to Batcher [1], this number is an exact bound. In theory, due to the AKS construction, there are sorting networks which are much faster than merge-sort networks. However, this holds only for impractically large $n$. The merge-sort networks of Batcher [1], invented in 1968, are still the best practical sorting networks [14, Section 5.3.4] and, as said, their depth is $\frac{\lceil \log(n) \rceil \cdot \lceil \log(2n) \rceil}{2}$. We refer to the last number as the Batcher depth. This section presents a merge-sort network of a Batcher depth which accelerates the $k$ lowest outputs (or $k$ highest outputs) to a delay smaller than $\lceil \log(n) \rceil \cdot \lceil \log(2k) \rceil$ as follows.
We assume, without loss of generality, that $k$ is a power of two. By pruning, we may also assume that $n$ is a power of two. Our construction is composed of a sorting stage followed by $\log(n) - \log(k)$ merging stages. In the sorting stage the $n$ input keys are separated into sets of $k$ keys each, and each of these sets is sorted separately by any sorting network of a Batcher depth. We now follow the merge sort method. Namely, in each of the merging stages, all the sorted sequences produced in the previous stage are paired and each pair is merged into a single sorted sequence. This merge is performed by a merging network, as per Lemma 3.3.5 or Lemma 3.4.3, that accelerates its $k$ lowest outputs to a delay of $\log(2^k)$ comparators and moreover, it possesses the restricted reachability property.

Consider the delay of the lowest $k$ outputs. This delay is composed of $\frac{\log k \cdot \log 2k}{2}$ comparators in the sorting stage and $\log 2k$ comparators in each of the $\log n - \log k$ merging stages. Due to the restricted reachability property, these delays are added up; that is, in the entire network, the delay of the lowest $k$ outputs is at most $\frac{\log k \cdot \log 2k}{2} + (\log n - \log k) \cdot \log 2k$. Clearly, the depth of the entire sorting network is a Batcher depth. Our construction is summarized in the following lemma.

**Theorem 12.** For every $0 < k < n$, there is a sorting network of width $n$ and of Batcher depth that accelerates all the lowest $k$ (or highest $k$) outputs to a delay of $\frac{\log k \cdot \log 2k}{2} + (\log n - \log k) \cdot \log 2k$.

In the special case where $n$ is a power of two, we may use Lemma 3.4.2 to achieve the above acceleration both for the highest $k$ keys and the lowest $k$ keys, simultaneously.
Chapter 4

The Strongest Model of Computation
Obeying 0-1 Principles

Tamir Levi, Technion, Haifa
Prof' Ami Litman, Technion, Haifa

The 0-1 Principle of Knuth and its many variants are well-known in the context of comparator networks. However, the comparator model is not the strongest model of computation obeying such principles. This paper studies another natural model of computation, the min-max model, that obeys all known 0-1 principles. More important, it is the strongest model obeying certain variants of the 0-1 Principle.

4.1 Introduction

Most of the work on key processing networks is under the well-known comparator model. This paper studies a different model of computation – the min-max model. These two models are presented shortly. The min-max model obeys all known variants of the 0-1 Principle. This, however, is easy and non-interesting. Our main contribution is the fact that the min-max model is the strongest model of computation obeying certain variants of the 0-1 Principle. Namely, this paper considers four variants of the 0-1 Principle. For each variant, the paper shows that if $M$ is a model of computation that obeys this variant and a mapping $g$ is computable under $M$ then $g$ is computable by a min-max network.
The min-max model is a natural extension of Monotone Boolean Circuits [7]; the latter processes binary values while the former process keys – members of some ordered set \( K \). To this end, AND-gates are replaced by \( \text{MIN-gates} \) and OR-gates are replaced by \( \text{MAX-gates} \). These are the only gates in the min-max model and, to make this model comparable to the comparator model, the fan-in of gates is exactly two. A min-max network is an acyclic network of such gates; see Figure 4.1. In this figure solid triangles denote MAX-gates and hollow triangles denote MIN-gates. A min-max network receives a sequence of \( n \) keys through its \( n \) input ports and produces a sequence of \( m \) keys through its \( m \) output ports. In Figure 4.1, these ports are depicted by bullets. Naturally, such a min-max network computes a mapping \( g: K^n \to K^m \).

The second class of networks addressed by this paper are the well-known comparator networks. A comparator is a combinational device that receives two keys and sorts them. It has two outgoing edges; on one of them, called the \( \text{MIN-edge} \), it sends the minimal key and on the other outgoing edge, called the \( \text{MAX-edge} \), it sends the maximal key. A comparator network is an acyclic network of comparators. See Figure 4.2. In this figure, a solid arrowhead denotes a MAX-edge and a hollow arrowhead denotes a Min-edge. The fan-out of an input port in a comparator network is exactly one. Hence, such a network has the same number of input ports and output ports. Naturally, a comparator network computes a mapping \( g: K^n \to K^n \) where \( n \) is the number of input ports.

Every comparator network can be translated into a min-max network by replacing every comparator with two gates – a MIN-gate and a MAX-gate. The opposite translation from a min-max network to a comparator network does not work. A gate can be replaced by a comparator. However, one of the values produced by this comparator is discarded while the other may be transmitted to several destinations. The resulting network is not a valid comparator network. (However, a relaxed version of the comparator model, without the fan-out restrictions, is equivalent to the min-max model.)
A well-known tool for design and analysis of key processing networks is the 0-1 Principle and its many known variants. All these variants are of the same flavor and each is related to a certain arrangement task – e.g., sorting, merging, Bitonic sorting and halving. For each of these tasks, the relevant variant of the 0-1 Principle states that any comparator network that works properly under binary keys – works properly under arbitrary keys. The classic 0-1 Principle of Knuth [14] relates to the task of sorting and similar variants for other tasks are common knowledge – see [10, Theorem 6], [25] and [24, Fact 2.1.2]. It is not surprising that all known variants hold also in the min-max model.

As said, this paper presents four new variants of the 0-1 Principle that hold in both models. (In contrast to the known variants, our variants do not relate to arrangement tasks.) This paper shows that each of our variants characterizes the min-max model in the following way: the min-max model is the strongest model of computation obeying it.

As discussed in Section 4.5, the comparator model is strictly weaker than the min-max model. Namely, there are mappings that are computable only under the latter and not under the former. This implies that the comparator model is not the strongest model of computation that obeys these variants of the 0-1 Principle. Apparently, there is no elegant principle, of a similar nature, that characterizes the comparator model. Hence, we find the min-max model to be more natural than the comparator model. Due to these characterizations of the min-max model, we suggest to define an oblivious key processing algorithm as an algorithm that can be implemented by a min-max network.

The style of the min-max model is somewhat unusual; this model is hardware oriented and does not explicitly specify when each elementary operation is executed. An equivalent model of computation, which computes exactly the same mappings, is presented in Section 4.3. This model is software oriented and has a familiar style.

Each of our four variants of the 0-1 Principle, presented shortly, holds in the min-max model and in the comparator model. This is the easy and non-interesting part of our work. The harder and more interesting part is the fact that, for each of these principles, the min-max model is the strongest model of computation that obeys it.
For an integer \( n > 0 \), an \( n\)-\textit{function} is a function \( f: \mathbb{K}^n \rightarrow \mathbb{K} \). All our new principles do not refer to the internal of the model in question; rather, they refer to the set of \( n\)-functions computable under this model. By the common convention, a comparator network does not compute an \( n\)-function but a mapping \( g: \mathbb{K}^n \rightarrow \mathbb{K}^n \). Such a mapping naturally induces \( n\) \( n\)-functions and we consider each of them to be computable under the comparator model.

We assume that \( |\mathbb{K}| \geq 2 \), that \( \mathbb{K} \) has two distinguished keys called ‘0’ and ‘1’ and that \( 0 < 1 \). We use the following terminology. An \textit{Order-Preserving Mapping} (\textit{OPM}) is a (weakly) monotonic mapping \( \tau: \mathbb{K} \rightarrow \mathbb{K} \). (OPMs are usually not computable under our models.) A \textit{binary OPM} (a.k.a. 0-1 \textit{OPM}) is an OPM \( \tau \) such that \( \tau(k) \in \{0, 1\} \), for every \( k \in \mathbb{K} \). For example, assume \( \mathbb{K} = \mathbb{Z} \) and let \( \tau, \rho: \mathbb{Z} \rightarrow \mathbb{Z} \) be defined by:

\[
\tau(z) \triangleq z^3 + 7 \quad \rho(z) \triangleq \left\lfloor \frac{1}{3z + 7} \right\rfloor
\]

Then, \( \tau \) is an OPM and \( \rho \) is a binary OPM.

An \( n\)-function is \textit{Bouricious} if it commutes\(^1\) with every OPM. An \( n\)-function is \textit{0-1 Bouricious} if it commutes with every 0-1 OPM. As observed by Bouricious [14, pp 223], every \( n\)-function computable by a comparator network (and also by a min-max network) is Bouricious due to the following argument. The 2-functions \( \min(x, y) \) and \( \max(x, y) \) are Bouricious and the Bouricious property is preserved under composition.

We now present our 4 variants of the 0-1 Principle. Each of them is a property of a model of computation. The first two variants explicitly refer to the keys ‘0’ and ‘1’ and the last two are 0-1 Principles only by their spirit.

\textbf{1. The 0-1 Bouricious Principle:} Any \( n\)-function computable under the model in question is 0-1 Bouricious.

To describe the next principle, we define a 0-1 \textit{image of a sequence} \( \langle v_1, v_2 \ldots v_n \rangle \) as any sequence of the form \( \langle \tau(v_1), \tau(v_2), \ldots, \tau(v_n) \rangle \) for some binary OPM \( \tau \).

\textbf{2. The 0-1 Comparing Principle:} Exact definition of this principle appears in Section 2.3. The core of this principle is the following property. For every two \( n\)-functions \( e \) and \( f \), computable under the model in question, and for every \( v \in \mathbb{K}^n \), one can tell the truth value of the predicate ‘\( e(v) \leq f(v) \)’ without knowing \( e, f \) or \( v \). This truth value is determined by the following details: the set \( U = \{u | u \text{ is a 0-1 image of } v \} \) and the truth value of the predicates ‘\( e(u) \leq f(u) \)’ for all \( u \in U \).

The next two principles are in the spirit of the 0-1 Principle but they do not explicitly refer to ‘0’ and ‘1’.

\textbf{3. The General Bouricious Principle:} Any \( n\)-function computable under the model in question is Bouricious.

\(^1\)Let \( X \) be an arbitrary set, let \( g: X^n \rightarrow X \) and \( \alpha: X \rightarrow X \). We say that \( g \) and \( \alpha \) \textit{commute} if \( \alpha(g(x_1, x_2 \ldots x_n)) = g(\alpha(x_1), \alpha(x_2) \ldots \alpha(x_n)) \), for every \( x_1, x_2 \ldots, x_n \in X \).
4. The Robustness Principle: The value of an \( n \)-function computable under the model in question is invariant under ‘minor modifications’ of the arguments. A precise definition of ‘minor modifications’ is given in Section 4.8.

As said, the atomic unit of information processed under our models is a key – a member of some ordered set \( \mathbb{K} \). The case of \( |\mathbb{K}| = 1 \) is singular, trivial and not interesting. In fact, most of the results of this paper are meaningless or wrong in this case. We assume that \( |\mathbb{K}| \geq 2 \) and that \( \mathbb{K} \) has two distinguished keys called ‘0’ and ‘1’ and that \( 0 < 1 \). It is well-known that most results about comparator networks are indifferent to the cardinality of \( \mathbb{K} \), as long as \( |\mathbb{K}| \geq 2 \). This holds also for min-max networks. Our paper demonstrates, however, that certain results are sensitive to the distinction between \( |\mathbb{K}| = 2 \) and \( |\mathbb{K}| \geq 3 \).

Consider our main results that each of our four principles characterize the min-max model as above. For the 0-1 Bouricious Principle and the General Bouricious Principle these results hold only when \( |\mathbb{K}| \geq 3 \); for the 0-1 Comparing Principle and the Robustness Principle these results hold for any \( |\mathbb{K}| \geq 2 \). On the other hand, the fact that our models obey the above principles is insensitive to \( |\mathbb{K}| \). These subtleties are addressed in the paper next to the theorems in question.

4.2 Related work

Knuth [14] observed a 4-function (presented in Section 4.5) that is computable by a min-max network\(^2\) but not by a comparator network. Other related works concern the time it takes to perform certain tasks under the min-max model and under the comparator model. We measure the computation time of a network by its depth – the maximal number of gates or comparators on a directed path in the network in question. Such a measure is fair w.r.t. the two models since every computational element in both models processes two keys.

Clearly, a computation performed by a comparator network can be emulated by a min-max network of the same depth; so, min-max networks are at least as fast as comparator networks. Moreover, some tasks can be performed faster by min-max networks as follows.

Consider the task of inserting a single key into a sorted sequence of \( n - 1 \) keys. By a straightforward reachability argument, the depth of a comparator network that performs this task is at least \( \lceil \log(n) \rceil \). As observed by Knuth [14], there is such a min-max network of depth two.

\(^2\)Knuth did not refer to the min-max model but to a relaxed variant of the comparator model having no fanout restrictions. As said, these two models compute exactly the same mappings.
Other related works concern Bitonic sorting. A Bitonic sequence is a rotation of a concatenation of two sequences – an ascending sequence followed by a descending one. A Bitonic sorter of width $n$ is an acyclic network that sort any Bitonic sequence of length $n$. In fact, Figures 4.1 and 4.2 depict two Bitonic sorters of width six. The first is under the min-max model and its depth is three. The second is under the comparator model and its depth is four. The fact that these networks are Bitonic sorters can be verified using the following well-known variant of the 0-1 Principle.

The 0-1 Principle for Bitonic sorting: A (Min-Max or comparator) network is a Bitonic sorter if and only if it sorts all binary Bitonic sequences of the appropriate width.

A verification, according to this principle, requires to test all 30 binary sequences of width six that are Bitonic and non-constant. However, there is a better way to accomplish the same goal. By [10], it suffices to verify that the network in question sorts the following six sequences.

\[
\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 2, 3, 4, 5, 6, 1 \rangle, \langle 3, 4, 5, 6, 1, 2 \rangle, \langle 4, 5, 6, 1, 2, 3 \rangle, \langle 5, 6, 1, 2, 3, 4 \rangle, \langle 6, 1, 2, 3, 4, 5 \rangle
\]

Namely, by [10], every (Min-Max or comparator) network that sorts all these sequences is a Bitonic sorter. Such a set of sequences is called a conclusive set for the functionality in question (Bitonic sorting in this case).

---

3The term ‘Bitonic’ was coined by Batcher [1] and we follow his terminology. We caution the reader that some authors use the same term with other meanings.
By a straightforward reachability argument, the Min-Max Bitonic sorters of Figure 4.1 is of minimal depth. As shown in [20], the Bitonic sorter of Figure 4.2 is also of minimal depth. Namely, [20] shows that the depth of a Bitonic sorter of width \( n \) in the comparator model is at least \( \lceil \log(n) \rceil + 1 \), for every \( n \in \mathbb{N} \) that is not a power of two. Hence, the task of Bitonic sorting 6 keys can be performed faster by a min-max network than by a comparator network. Moreover, as shown in [18], there are infinitely many \( n \) such that the task of Bitonic sorting \( n \) keys can be performed faster by a min-max network than by a comparator network.

4.3 The Min-Max Programming Model

The min-max model is hardware oriented. This section presents an equivalent model of computation which is software oriented. The **Min-Max Programming Model** has only one data type – a key. These keys are stored in variables and there are only two elementary instructions: "\( z \leftarrow \min(x, y) \)" and "\( z \leftarrow \max(x, y) \)". where \( x, y \) and \( z \) stand for arbitrary (not-necessarily distinct) variables. This model does not allow for any control operations (such as branching or looping) and especially no conditional instructions of the form “if \( x < y \) then ...”; therefore, a (non-parallel) program in this model is a straight-line code of the above instructions. These instructions are executed in a serial manner one after the other.

Some of the variables are *input variables* while others are *output variables*; these variables play the role of the input and output ports of a min-max network. A Min-Max program must obey the ‘read-after-write’ restriction. Namely, a variable must be written before it is read. In this context, input variables are considered to be written at the start of the program’s execution and output variables are considered to be read at the end of the program’s execution.

There is no parallelism in a serial program as above while the main goal of this model is parallel processing of keys. To this end, a parallel program is a sequence of *macro-instructions* that are executed one after the other. Each macro-instruction is a set of elementary instructions as above. All the elementary instructions of a macro-instruction are executed simultaneously. Due to this, a parallel Program must obey, in addition to the ‘read-after-write’ restriction, the following restriction. Two instructions of the same macro-instruction do not write to the same variable. In our model a macro-instruction is executed in one time unit. Hence, the total run-time of a parallel program is the number of its macro-instructions. We refer to this number as the **depth** of the program.
A Min-Max (parallel) program has a natural translation into a min-max network that performs the same computation. Namely, the resulting min-max network is the Data Dependency Graph of the program. The resulting min-max network performs exactly the same elementary operations as the original Min-Max program. However, the depth is not necessarily preserved. The depth of the min-max network may be smaller (but never greater) than the depth of the program.

Every min-max network can be translated into a Min-Max program that performs exactly the same elementary operations while preserving the depth of the min-max network. Usually, a Min-Max program can be translated into several (substantially different) Min-Max programs as above. Every gate of the min-max network corresponds to an elementary instruction of the parallel program. However, the location of this elementary instruction is not necessarily unique.

Consider translating the min-max network of Figure 4.1 into a parallel program. Let the input variables of the program be $b_1, b_2, \ldots, b_6$ and its output variables be $o_1, o_2, \ldots, o_6$. Our min-max network can be translated into several (substantially different) programs of depth three. One of them is as follows:

1. \[
\begin{align*}
x_1 &\leftarrow \min(b_1, b_3), \quad x_2 \leftarrow \min(b_2, b_4), \quad x_3 \leftarrow \min(b_2, b_5), \quad x_4 \leftarrow \min(b_3, b_6), \\
x_5 &\leftarrow \min(b_1, b_4), \quad x_6 \leftarrow \max(b_2, b_5), \quad x_7 \leftarrow \min(b_3, b_5), \quad x_8 \leftarrow \min(b_4, b_6)
\end{align*}
\]

2. \[
\begin{align*}
y_1 &\leftarrow \min(x_1, b_5), \quad y_2 \leftarrow \min(x_2, b_6), \quad y_3 \leftarrow \min(b_1, b_4), \quad y_4 \leftarrow \max(x_3, x_4), \\
y_5 &\leftarrow \min(x_5, x_6), \quad y_6 \leftarrow \max(b_3, b_6), \quad y_7 \leftarrow \max(b_1, x_7), \quad y_8 \leftarrow \max(b_2, x_8)
\end{align*}
\]

3. \[
\begin{align*}
o_1 &\leftarrow \min(y_1, y_2), \quad o_2 \leftarrow \max(y_1, y_2), \quad o_3 \leftarrow \max(y_3, y_4), \\
o_4 &\leftarrow \min(y_5, y_6), \quad o_5 \leftarrow \min(y_7, y_8), \quad o_6 \leftarrow \max(y_7, y_8)
\end{align*}
\]

To summarize, the min-max model and the Min-Max Programming Model are equivalent in the following sense. Both models compute the same set of mappings. Moreover, each mapping computable under one model in a certain depth is computable under the other model in the same depth.

### 4.4 The 0-1 Principle and the min-max model

This section shows that the min-max model obeys the classical 0-1 Principle. To this end, a binary sequence (a.k.a. a 0-1 sequence) is a sequence of zeroes and ones. Knuth’s classical 0-1 Principle [14, pp. 224] states that:
Theorem 13. A comparator network is a sorting network if and only if it sorts all 0-1 sequences.

As said, many variants of Theorem 13 are known. All these variants are of the form of Theorem 13 in which the sorting functionality is replaced by another functionality, for example merging [2], Bitonic sorting [25] and halving [10]. This phenomena, that many functionalities enjoy such a natural extension of the 0-1 Principle was investigated in [16]. That work address the question “Which functionalities enjoy such an extension and which do not ?”.

Recall that the General Bouricious Principle states that any $n$-function computable under the model in question is Bouricious. The proof of Theorem 13 is based on the following lemma of Bouricious [14, pp. 196,224].

Lemma 4.4.1. The comparator model satisfies the General Bouricious Principle.

The General Bouricious Principle is easily extended to the context of min-max networks. Namely,

Lemma 4.4.2. The min-max model satisfies the General Bouricious Principle.

As said, the proof of Theorem 13 is based on Lemma 4.4.1. This proof can be easily extended to the context of min-max networks except of the following minor problem. With respect to sorting, a min-max network has an additional mode of failure over those of a comparator network. Namely, the former network may lose or duplicate keys. In this regard, a mapping $g: \mathbb{K}^n \rightarrow \mathbb{K}^n$ is isomeric if, for every $v \in \mathbb{K}^n$, $g(v)$ is a permutation of $v$. That is, every key appears in $v$ and in $g(v)$ with the same multiplicity. Clearly, a sorting network should compute an isomeric mapping. To tackle this mode of failure we use the following lemma which easily follows from Lemma 4.4.2. For a mapping $g$, this lemma use the notation $g^{0,1}$ to denote the restriction of $g$ to the set $\{0,1\}$.

Lemma 4.4.3. Let $g: \mathbb{K}^n \rightarrow \mathbb{K}^n$ be a mapping computable by a min-max network. Then $g$ is isomeric if and only if $g^{0,1}$ is isomeric.

Lemma 4.4.3 and standard arguments establish that the classical 0-1 Principle holds also for min-max networks. Namely:
Theorem 14. A min-max network is a sorting network if and only if it sorts all 0-1 sequences.

4.5 The min-max model is stronger than the comparator model

This section demonstrates that the min-max model is strictly stronger than the comparator model. That is, some mappings are computable only in the min-max model and not in the comparator model. Knuth observed that:

Lemma 4.5.1 ([14]). There is a 4-function that is computable by a min-max network and is not computable by a comparator network.

Proof. Every $n$-function $f$ computable by a comparator network is either a projection or it has two distinct arguments $x_i$ and $x_j$ s.t. $f$ is invariant under a transposition of the values of $x_i$ and $x_j$. The min-max model allows to compute $n$-functions that violate this property. An example is the following 4-function ([14, pp. 241]).

$$f(x_1, x_2, x_3, x_4) = \max \left( \min(x_1, x_2), \min(x_2, x_3), \min(x_3, x_4) \right)$$

This proof demonstrates a major difference between the min-max model and the comparator model. Namely, let $f_1, f_2 \ldots f_k$ be $n$-functions such that all of them are computable under the model in question. In the min-max model there is a single network that computes all these $n$-functions simultaneously. This is not the case in the comparator model, as demonstrated by the proof of Lemma 4.5.1. That is, each of the three 4-functions $\min(x_1, x_2), \min(x_2, x_3)$ and $\min(x_3, x_4)$ is computable by a comparator network but no single comparator network computes all these 4-functions simultaneously.

By the common convention, a comparator network computes a mapping $g: \mathbb{K}^n \rightarrow \mathbb{K}^n$ rather than an $n$-function. Even under this convention, the min-max model is stronger than the comparator model. A trivial example is due to the fact that a mapping computable by a comparator network is isomorphic (defined in Section 4.4) and this is not the case for min-max networks. So the interesting question concerns isomorphic mappings that are computable only by min-max networks. The next lemma shows that such mappings exist.

Lemma 4.5.2. For any $n \geq 4$ there is an isomorphic mapping $g: \mathbb{K}^n \rightarrow \mathbb{K}^n$ that is computable by a min-max network and not by a comparator network.

An example of such a $g$ is given in Figure 4.3. Lemma 4.5.2 follows from Lemma 4.5.1 combined with the following lemma.
Lemma 4.5.3. For every $n$-function $f$, computable by a min-max network, there is an isomeric mapping $g : \mathbb{K}^n \to \mathbb{K}^n$, also computable by a min-max network, such that the first key of $g(v)$ is $f(v)$, for every $v \in \mathbb{K}^n$.

For example, let $f$ be the 4-function defined in the proof of Lemma 4.5.1. A mapping $g : \mathbb{K}^4 \to \mathbb{K}^4$, as per Lemma 4.5.3, is given in Figure 4.3.

![Figure 4.3: An extension as per Lemma 4.5.3.](image)

The proof of Lemma 4.5.3 is deferred to the appendix since it relies on results which are proved ahead. However, there are no circular arguments since Lemmas 4.5.2 and 4.5.3 are not used in the rest of the paper.

4.6 The Bouricious Principles

The General Bouricious Principle (the 0-1 Bouricious Principle) states that every $n$-function, computable under the model in question, commutes with any OPM (0-1 OPM). To study these properties we use the following terminology. An $n$-function is a choice function if $f(v)$ is always a member of $v$. Clearly, min-max networks and comparator networks compute only choice functions. This property is critical to our work and is due to the fact that we disallow gates that produce constant values. Recall that ‘≤’ denotes the order relation over $\mathbb{K}$. We define a partial order over $\mathbb{K}^n$, as follows. For every two sequences $u, v \in \mathbb{K}^n$: ‘$u \preceq v$’ denotes that $u_i \leq v_i$ for every $i$. An $n$-function $f$ is monotone if $u \preceq v$ implies $f(u) \leq f(v)$. Clearly, all $n$-functions computable by comparator networks and min-max networks are monotone. For any two $n$-functions $e$ and $f$, let $e \preceq f$ denote that $e(v) \leq f(v)$, for every $v \in \mathbb{K}^n$. For a sequence $v \in \mathbb{K}^n$ and an OPM $\tau$, we define $\tau(v) \triangleq (\tau(v_1), \ldots, \tau(v_n))$.

The main result of this section is the following theorem.
Theorem 15. Let $|\mathbb{K}| \geq 3$. Then the following three conditions are equivalent, for every $n$-function $f$:

**Condition (1)** $f$ is computable by a min-max network.

**Condition (2)** $f$ is Bouricious.

**Condition (3)** $f$ is 0-1 Bouricious.

In other words, Theorem 15 says that for $|\mathbb{K}| \geq 3$:

**Statement (a)** The min-max model obeys the 0-1 Bouricious Principle and the General Bouricious Principle.

**Statement (b)** For each of these two principles, the min-max model is the strongest model of computation that obeys it.

Before proving Theorem 15 let us consider the requirement $|\mathbb{K}| \geq 3$. Statement (a) holds even when $|\mathbb{K}| = 2$. On the other hand, Statement (b) holds only for $|\mathbb{K}| \geq 3$. This is established as follows. When $|\mathbb{K}| = 2$, there are only three OPMs: the identity, the constant ‘0’, and the constant ‘1’. It is easy to verify that each of them commutes with any choice function. That is, every choice function is Bouricious. However, there are choice functions that are not monotone and therefore, are not computable by a min-max network. Such a function is, for example, the parity 3-function $p(x_1, x_2, x_3) \triangleq (x_1 + x_2 + x_3) \mod 2$.

The hard part in the proof of Theorem 15 is the derivation of Condition (1) from Condition (3). We prove it using the following lemmas which state that every 0-1 Bouricious $n$-function $f$ satisfies:

1. $f$ is a choice function (Lemma 4.6.1).
2. $f^{0-1}$ is monotone (Lemma 4.6.2).
3. $f$ is determined by $f^{0-1}$ (Lemma 4.6.3).

We start with the following lemma.

**Lemma 4.6.1.** Every 0-1 Bouricious $n$-function is a choice function.

*Proof.* Assume otherwise; let $f$ be a 0-1 Bouricious $n$-function and let $v \in \mathbb{K}^n$ such that $f(v)$ is not a member of $v$. Clearly, there are two 0-1 OPMs $\tau$ and $\tau'$ such that $\tau(f(v)) \neq \tau'(f(v))$ and $\tau(v) = \tau'(v)$. Since $f$ is 0-1 Bouricious we have $\tau(f(v)) = f(\tau(v)) = f(\tau'(v)) = \tau'(f(v))$. This contradicts the fact that $\tau(f(v)) \neq \tau'(f(v))$. □
Lemma 4.6.2. Let $|\mathbb{K}| \geq 3$ and let $f$ be 0-1 Bouricious $n$-function. Then $f^{0-1}$ is a monotone choice function.

Proof. By Lemma 4.6.1, $f$ is a choice function. Let $x, y \in \{0, 1\}^n$ be two binary sequences such that $x \leq y$. We need to show that $f(x) \leq f(y)$. Since $|\mathbb{K}| \geq 3$ there are three keys $k_0 < k_1 < k_2$. Let $v \in \mathbb{K}^n$ be the sequence defined by $v_i = k_{x_i + y_i}$. Let $\tau, \tau': \mathbb{K} \rightarrow \{0, 1\}^n$ be two 0-1 OPMs satisfying $\tau(v) = x$, $\tau'(v) = y$ and $\tau \preceq \tau'$. Since $f$ is 0-1 Bouricious, $f(x) = f(\tau(v)) = \tau(f(v)) \leq \tau'(f(v)) = f(\tau'(v)) = f(y)$.

The parity 3-function demonstrates that the requirement $|\mathbb{K}| \geq 3$ is mandatory in Lemma 4.6.2. The next lemma states that a 0-1 Bouricious $n$-function is determined by its binary restriction.

Lemma 4.6.3. Let $f$ and $g$ be 0-1 Bouricious $n$-functions and let $f^{0-1} = g^{0-1}$, then $f = g$.

Proof. Assume otherwise and let $v \in \mathbb{K}^n$ be a sequence such that $f(v) \neq g(v)$. Let $\tau$ be a 0-1 OPM such that $\tau(f(v)) \neq \tau(g(v))$. Since $f$ and $g$ are 0-1 Bouricious, $f(\tau(v)) \neq g(\tau(v))$. This contradicts the assumption that $f^{0-1} = g^{0-1}$.

As said, the min-max model is a natural generalization of the Monotone Boolean Circuits model. It is well-known that every monotone choice function over binary values is computable by a Monotone Boolean Circuit. In the context of binary values, a MIN-gate and an AND-gate are the very same gate. The same holds for OR-gates and MAX-gates. This implies the following observation.

Theorem 16. Let $|\mathbb{K}| = 2$. Then an $n$-function $f$ is computable by a min-max network if and only if $f$ is a monotone choice function.

The condition $|\mathbb{K}| = 2$ in Theorem 16 is mandatory as follows. Let $\mathbb{K} = \{0, 1, 2\}$ and let $f$ be the 2-function $f(x, y) \triangleq \max(x, y) \cdot \min(x, y, 1)$. Clearly, $f$ is a monotone choice function; however, $f$ does not commute with the OPM $\tau(k) \triangleq \lfloor k/2 \rfloor$. Since $f$ is not Bouricious, by Theorem 15, $f$ is not computable by a min-max network.

Proof of Theorem 15: By Lemma 4.4.2, Condition (1) implies Condition (2). By definition, Condition (2) implies Condition (3). It remains to show that Condition (3) implies Condition (1).

Let $f$ be a 0-1 Bouricious $n$-function. By Lemma 4.6.2, $f^{0-1}$ is a monotone choice function. By Lemma 16, $f^{0-1}$ is computable by a min-max network $N$. By Lemma 4.4.2, $N$ computes a Bouricious $n$-function which, by Lemma 4.6.3, is $f$. ■
Theorem 15 and Lemma 4.6.3 imply the following surprising result:

**Theorem 17.** For any $n$, the number of $n$-functions computable by min-max networks is independent of $|\mathbb{K}|$.

Lemma 4.6.3 implies that the number of $n$-functions computable by min-max networks and the number of monotone Boolean functions of $n$ arguments differ by exactly two. The problem of finding the latter number, for any $n$, has been extensively studied and is known as the Dedekind Problem [13].

### 4.7 The 0-1 Comparing Principle.

This section studies the 0-1 Comparing Principle and uses the following notation. For a sequence $v \in \mathbb{K}^n$, define $\widehat{v} \triangleq \{v'|v'$ is a 0-1 image of $v\}$. For $1 \leq i \leq n$, let $\pi^n_i$ denote the projection of $\mathbb{K}^n$ on its $i$'th coordinate; namely, $\pi^n_i(v_1, v_2, \ldots, v_n) \triangleq v_i$.

**Definition 4.7.1.** The 0-1 Comparing Principle is the following property of a model of computation:

1. Any $n$-function computable under the model is monotonic and is a choice function.

2. All projections are computable under the model.

3. For every two $n$-functions, $e$ and $f$ computable under the model, the truth value of the predicant $'e(v) \leq f(v)'$ is determined solely by the set $\widehat{v}$ and the truth value of the predicants $'e(v') \leq f(v')'$, for every $v' \in \widehat{v}$.

Trivially, both the min-max model and the comparator model obey Conditions (1) and (2) of Definition 4.7.1. Let $e$ and $f$ be two Bouricious $n$-functions and let $v \in \mathbb{K}^n$. Then $e(v) \leq f(v)$ if and only if $e(v') \leq f(v')$, for every $v' \in \widehat{v}$. Hence, Bouricious $n$-functions comply with Condition (3) of Definition 4.7.1. This, combined with Lemmas (4.4.1) and (4.4.2), implies the following lemma.

**Lemma 4.7.1.** The min-max model and the comparator model obey the 0-1 Comparing Principle.

The harder and more interesting result of this section is that the min-max model is the strongest model of computation that obeys the 0-1 Comparing Principle. For this result we use the following terminology. Let $f$ be an $n$-function and $v \in \mathbb{K}^n$; the *signature of $f$ and $v$*, denoted $S(f, v)$, is the restriction of $f$ to $\widehat{v}$. Namely, $S(f, v) \triangleq f |_{\widehat{v}}$. 
Theorem 18. The min-max model is the strongest model of computation that obeys the 0-1 Comparing Principle.

Proof. Let us consider only the harder case of $|K| \geq 3$. Let $M$ be a model of computation that obeys the 0-1 Comparing Principle. We show that any $n$-function computable under $M$ is 0-1 Bouricious. By Theorem 15 such an $n$-function is computable by a min-max network.

Let $e$ and $f$ be two $n$-functions computable under $M$ and let $v \in K^n$. Condition (3) of Definition 4.7.1 implies that the truth value of the predicant '$e(v) = f(v)$' is determined by the pair of signatures $(S(e, v), S(f, v))$. Hence, there is an abstract Black-Box that receives only $S(e, v)$ and $S(f, v)$ and declares the truth value of the predicant '$e(v) = f(v)$'. In fact, our proof relies on this property rather than on Condition (3) of Definition 4.7.1.

Assume this Black-Box receives a pair of signatures $(S(e, v), S(f, v))$ such that $S(e, v) = S(f, v)$. In this case, the Black-Box must declare that $e(v) = f(v)$; otherwise, it fails when $e = f$. This implies that, for any two $n$-functions $e$ and $f$ computable under $M$ and for any $v \in K^n$, if $S(e, v) = S(f, v)$ then $e(v) = f(v)$.

Let $f$ be computable under $M$ and let $v \in K^n$. Clearly, $\hat{v}$ is a chain under the relation '$\preceq$'; namely, for any two $v', v'' \in \hat{v}$, either $v' \preceq v''$ or $v'' \preceq v'$. The fact that $f^{0-1}$ is a monotonic choice function and that $\hat{v}$ is a chain implies that there is some $i$ such that $S(f, v) = S(\pi^n_i, v)$. This is the only place that Condition (1) is used in the proof; hence, this condition can be replaced by the following weaker requirement – “For any $n$-function $f$, computable under the model in question, $f^{0-1}$ is a monotonic choice function”.

We now show that $f$ is 0-1 Bouricious. Let $\tau$ be a 0-1 OPM. Recall that $S(f, v) = S(\pi^n_i, v)$. By the above Black-Box argument, $f(v) = \pi^n_i(v)$ and therefore,

$$\tau(f(v)) = \tau(\pi^n_i(v))$$

Every projection commutes with any function $\alpha: K \rightarrow K$; hence:

$$\tau(\pi^n_i(v)) = \pi^n_i(\tau(v))$$

By definition, $\tau(v) \in \hat{v}$. Therefore, the fact that $S(f, v) = S(\pi^n_i, v)$ implies that:

$$\pi^n_i(\tau(v)) = f(\tau(v))$$

Combining these equalities yields that:

$$\tau(f(v)) = f(\tau(v))$$

This holds for every 0-1 OPM $\tau$ and every $v \in K^n$, implying that $f$ is 0-1 Bouricious.
4.8 The Robustness Principle

In a nutshell, the Robustness Principle states that the value of an \( n \)-function, computable under the model in question, is invariant under “minor variations” of its arguments. To formalize this concept, let \( v, v' \in \mathbb{K}^n \) and \( f \) be an \( n \)-function. We say that \( v' \) is a near variant of \( v \) (under \( f \)) if the two following conditions hold for every \( i \):

- \( v_i \leq f(v) \Rightarrow v'_i \leq f(v) \).
- \( v_i \geq f(v) \Rightarrow v'_i \geq f(v) \).

An \( n \)-function \( f \) is robust if every near variant \( v' \) of every \( v \in \mathbb{K}^n \) satisfies \( f(v') = f(v) \).

**Definition 4.8.1.** The Robustness Principle is the following property of a model of computation: Any \( n \)-function computable under the model is robust and is a choice function.

The easy part of this section is summarized in the following lemma.

**Lemma 4.8.1.** The min-max model and the comparator model obeys the Robustness principle.

**Proof.** Let \( f \) be an \( n \)-function computable under one of these models. Clearly, \( f \) is a choice function. Let \( v' \) be a near variant of a sequence \( v \). It is not hard to see that there are two OPMs \( \tau_1 \) and \( \tau_2 \) that satisfy \( \tau_1(v) \preceq v' \preceq \tau_2(v) \) and \( \tau_1(f(v)) = \tau_2(f(v)) = f(v) \). Since \( f \) is Bouricious and monotone, \( f(v) = \tau_1(f(v)) = f(\tau_1(v)) \leq f(v') \leq f(\tau_2(v)) = \tau_2(f(v)) = f(v) \), proving that \( f(v) = f(v') \).

The harder and more interesting result of this section is the following theorem.

**Theorem 19.** The min-max model is the strongest model of computation that obeys the Robustness Principle.

**Proof.** Let \( f \) be an \( n \)-function that is robust and a choice function. We consider the two cases of \( |\mathbb{K}| = 2 \) and \( |\mathbb{K}| \geq 3 \). First, assume \( |\mathbb{K}| = 2 \). By Theorem 16, it suffices to show that \( f \) is monotone. So, let \( v, v' \in \mathbb{K}^n \) such that \( v \preceq v' \). If \( f(v) = 0 \), then trivially \( f(v) \leq f(v') \). Otherwise, \( f(v) = 1 \); then \( v' \) is a near variant of \( v \) and therefore \( f(v') = 1 \). To summarize, \( f(v) \leq f(v') \) always holds.

For the case of \( |\mathbb{K}| \geq 3 \) we use the following fact. Let \( \tilde{\tau} \) be an OPM of some finite ordered set \( \tilde{\mathbb{K}} \). Then \( \tilde{\tau} \) is a composition of OPMs \( \mu_t \circ \mu_{t-1} \cdots \circ \mu_1 \) of \( \tilde{\mathbb{K}} \) such that every \( \mu_i \) relocates exactly one member of \( \tilde{\mathbb{K}} \).
We now show that $f$ is Bouricious; that is, $f$ commutes with every OPM $\tau$ of $\mathbb{K}$ (which is not necessarily finite). Let $v \in \mathbb{K}^n$. Due to the above fact, we may assume without loss of generality, that among the keys that appear in $v$, $\tau$ relocates exactly one of them, lets call it $k$.

If $k \neq f(v)$, then $\tau(v)$ is a near variant of $v$. Therefore, $f(\tau(v)) = f(v) = \tau(f(v))$. Otherwise, $k = f(v)$. Assume, for a contradiction, that $f(\tau(v)) \neq \tau(f(v))$. Then $v$ is a near variant of $\tau(v)$. This implies that $f(\tau(v)) = f(v) = k$. But $k$ does not appear in $\tau(v)$, contradicting the fact that $f$ is a choice function.

Appendix

The proof of the following lemma is deferred from Section 4.5.

**Lemma 4.5.3** For every $n$-function $f$, computable by a min-max network, there is an isomeric mapping $g : \mathbb{K}^n \rightarrow \mathbb{K}^n$, also computable by a min-max network, such that the first key of $g(v)$ is $f(v)$, for every $v \in \mathbb{K}^n$.

**Proof.** Let $g : \mathbb{K}^n \rightarrow \mathbb{K}^n$ and $g_1, g_2 \ldots g_n : \mathbb{K}^n \rightarrow \mathbb{K}$ be the unique mapping and $n$-functions that satisfy the following conditions for every $v \in \mathbb{K}^n$:

1. $g(v) = \langle g_1(v), g_2(v), \ldots, g_n(v) \rangle$.
2. $g_1 = f$.
3. $g_2(v) \leq g_3(v) \leq \cdots \leq g_n(v)$.
4. $g$ is isomeric.

It is not hard to see that each $g_i^{0-1}$ is a monotone choice $n$-function; hence, by Theorem 16, each $g_i^{0-1}$ is computable by a min-max network. Hence, there is a min-max network $N$ which, when restricted to binary values, computes $g^{0-1}$.

Let $\bar{g} : \mathbb{K}^n \rightarrow \mathbb{K}^n$ and $\bar{g}_1, \bar{g}_2 \ldots \bar{g}_n : \mathbb{K}^n \rightarrow \mathbb{K}$ be the mapping and $n$-functions computed by $N$. We show that the mapping $\bar{g}$ and the $n$-functions $\bar{g}_i$ satisfy Conditions (1)-(4). Since there is a unique mapping that satisfy these conditions, it follows that $\bar{g} = g$.

By construction, Condition (1) holds. Since $\bar{g}_i^{0-1} = f^{0-1}$, Lemma 4.6.3 implies Condition (2). Lemma 4.7.1 implies Condition (3). By Lemma 4.4.3 and the fact that $g^{0-1}$ is isomeric, $\bar{g}$ is isomeric.

\qed
Chapter 5

Lower Bounds On Bitonic Sorting Networks

Tamir Levi, Technion, Haifa
Prof’ Ami Litman, Technion, Haifa

This paper points to an abnormal phenomena of comparator networks. For most key processing problems (such as sorting, merging or insertion) the smaller the input size the easier the problem. Surprisingly, this is not the case for Bitonic sorting. Namely, the minimal depth of a comparator network that sorts all Bitonic sequences of \( n \) keys is not monotonic in \( n \).

We show that for any \( n \), not a power of two, the depth of a Bitonic sorter of \( n \) keys is at least \( \lceil \log(n) \rceil + 1 \). This contrast with Batcher’s seminal construction of a Bitonic sorter of \( 2^j \) keys and depth \( j \). That is, in order to reduce the number of keys from \( n = 2^j \) to \( n' > 2^{j-1} \) one must increase the depth of the network.

5.1 Introduction

This paper points to an abnormal phenomena of comparator networks. For most key processing problems (such as sorting, merging or insertion) the smaller the input size the easier the problem. Surprisingly, this is not the case for Bitonic sorting. Namely, the minimal depth of a comparator network that sorts all Bitonic sequences of \( n \) keys is not monotonic in \( n \).
Let \( D(n) \) denote the minimal depth of a Bitonic sorter of \( n \) keys. This paper shows that for any \( n \), not a power of two, \( D(n) \geq \lceil \log(n) \rceil + 1 \). Due to Batcher’s seminal construction \( D(2^j) = j \). That is, \( D(n) < D(n') \) for any \( n = 2^j > n' > 2^{j-1} \).

For some values of \( n \) (e.g., 3 and 5) our bound is tight; however, we do not know if this bound is always tight. We also do not know if \( D(n) = \log(n) + O(1) \).

5.2 Preliminaries

Our work concerns the well-known concept of a comparator network [14]. To be self-contained, we provide the following definitions. A comparator is a combinational device that receives, via two incoming edges, two keys and sorts them. It has two outgoing edges; on one of them, called the MIN-edge, it sends the minimal key and on the other outgoing edge, called the MAX-edge, it sends the maximal key. A comparator network is an acyclic network of comparators. See Figure 5.1. In this figure, a solid arrowhead denotes a MAX-edge and a hollow arrowhead denotes a Min-edge. Keys enter a comparator network via its input ports and exit the network via its output ports. These ports are depicted by solid circles. The network specifies, in some form, how the input is fed to the input ports and how the output is assembled from the output ports. The fan-out of an input port in a comparator network is exactly one. Hence, a comparator network has the same number of input ports and output ports; this number is referred to as the width of the network. The depth of the network is the maximal number of comparators on a directed path in the network. For example, the network of Figure 5.1 is of width five and depth four.

![Figure 5.1: A Bitonic sorter of width 5 and depth 4](image)

As said, comparator networks processes keys which are members of some ordered set, \( K \). The exact nature of keys is usually not important but for definiteness we choose \( K = Q \). This paper focuses on comparator networks that sort Bitonic sequences. A sequence of keys is Bitonic\(^1\) if it is a rotation of a concatenation of two sequences – an ascending sequence followed by a descending one. A Bitonic sorter of width \( n \) is a comparator network that sort any Bitonic sequence of \( n \) keys. The network of Figure 5.1 is a Bitonic sorter. Verifying the last statement via a variant of the 0-1 Principle [14] requires to test all 20 binary sequences of width five that are Bitonic and non-constant. However, there is a better way to accomplish the same goal. By Lemma 5.4.1, it suffices to verify that the network in question sorts the following five sequences: \( \langle 1, 2, 3, 4, 5 \rangle, \langle 2, 3, 4, 5, 1 \rangle, \langle 3, 4, 5, 1, 2 \rangle, \langle 4, 5, 1, 2, 3 \rangle, \langle 5, 1, 2, 3, 4 \rangle. \)

\(^1\)The term ‘Bitonic’ was coined by Batcher [1] and we follow his terminology. We caution the reader that some authors use the same term with other meanings.
The Bitonic sorter of Figure 5.1 demonstrates that $D(5) \leq 4$. Our lower bound implies that $D(5) \geq \lceil \log(5) \rceil + 1 = 4$; hence, for $n = 5$ our lower bound is tight.

5.3 Related work

We summarize the little that is known on the function $D$. By a straightforward reachability argument, for every $n$,

$$D(n) \geq \lceil \log(n) \rceil$$

(Batcher’s [1] well-known Bitonic sorter is based on a network of depth one that splits every Bitonic sequence of width $2n$ into two Bitonic sequences of width $n$. The first contains the lower keys and the second contains the higher keys; therefore,

$$D(2 \cdot j) \leq D(j) + 1$$

This, the fact that $D(2) = 1$, and Inequality (1) imply:

$$D(2^i) = j$$

Nakatani et al. [25] has presented an elegant procedure to produce a larger Bitonic sorter from smaller ones as follows. Let $B'$ and $B''$ be two Bitonic sorters of width $n'$ and $n''$ and of depth $d'$ and $d''$, respectively. They have shown how to combine several copies of $B'$ and $B''$ into a Bitonic sorter of width $n' \cdot n''$ and of depth $d' + d''$. This implies that

$$D(i \cdot j) \leq D(i) + D(j)$$

The only technique that constructs Bitonic sorters for any width is due to Batcher and Liszka [22]. Their construction partitions a Bitonic sequence of length $n$ into two Bitonic subsequences of length $\lceil \frac{n}{2} \rceil$ and $\lfloor \frac{n}{2} \rfloor$ and sorts each of them recursively. The two resulting sorted sequences are then merged into a single sorted sequence. This merge requires a depth one network when $n$ is even and a depth two network when $n$ is odd. Therefore,

$$D(2n + 1) \leq \max(D(n), D(n+1)) + 2$$

This implies that for every $n$,

$$D(n) \leq 2 \lceil \log(n) \rceil - 1$$

Levy and Litman [19] studied another class of key processing networks – the min-max networks. A min-max network is built of 2-input Min-gates and Max-gates. There is no fanout restriction on the gates or the input ports of a min-max network. They construct, for every $n$, a min-max network that is a Bitonic sorter of width $n$ whose depth is at most $\lceil \log(n) \rceil + 3$. As said, it is unknown if the comparator model has a Bitonic sorter of a comparable depth – namely, of depth $\lceil \log(n) \rceil + O(1)$. They also show that the above abnormality exists (in a slightly different form) in min-max networks.

5.4 Lower Bound of Bitonic sorters

In this section we establish the main result of this paper. Namely, that:
Theorem 20. For every \( n \), not a power of two, \( D^n > \lceil \log(n) \rceil \).

To this end, we propose a new manner to organize keys. Traditionally, keys are organized as sequences. Such sequences are fed into and produced by comparator networks. However, in many cases it is preferable to consider the keys to have a different structure. For example, in the work of Nakatani [25] keys are arranged in a matrix. In the context of Bitonic sequences we prefer to consider the keys to be arranged on a circle. To study arbitrary arrangement of keys, we pick a finite set, \( X \), called the index set. A vector \( v \) over \( X \) is a function \( v: X \to \mathbb{K} \). We denote the elements of \( v \) by \( v_x \), rather than \( v(x) \), and the term ‘index’ is due to this notation. For example, a vector over \( \{1, 2, \ldots, n\} \) is the familiar sequence of \( n \) keys while a vector over \( \{1, 2, \ldots, n\} \times \{1, 2, \ldots, m\} \) is an \( n \times m \) matrix of keys.

To allow networks to receive and produce such vectors, a network \( N \) is associated with an input index set denoted \( I(N) \) and with an output index set denoted \( O(N) \). Such a network receives vectors over \( I(N) \) and produces vectors over \( O(N) \). The former vectors are called input vector of \( N \). There are two bijections that specify how the input is fed into the network and how the output is collected from the network; one is between the input ports and \( I(N) \) and the other is between the output ports and \( O(N) \).

A sequential key structure is, by nature, unsymmetrical; it has a first member and a last member. In the context of Bitonic sequences, this lack of symmetry is artificial, since Bitonic sequences (by definition) are closed under rotations. Hence, it is preferred that the keys are organized in a structure that is of a circular nature – a structure that has no first element or last element. To formalize this idea we pick, for every \( n \), some directed graph \( G^n = (C^n, E^n) \) which is a cycle of \( n \) vertices. The exact nature of the vertices is not important. The set \( C^n \) serves as the index set for Bitonic vectors. Namely, a circular vector of width \( n \) is a function \( v: C^n \to \mathbb{K} \). In contrast to a sequence, a circular vector has no first or last elements – all elements play the same role in this structure. Figure 5.2 depicts three such circular vectors. By our convention, a Bitonic sorter \( B \) receives circular vectors over \( C^n \); hence, \( I(B) = C^n \) and \( O(B) = \{1, 2, \ldots, n - 1, n\} \).

![Figure 5.2: Three circular vectors.](image)

A circular vector \( v: C^n \to \mathbb{K} \) is Bitonic if there are two vertices \( u', u'' \in C^n \) such that \( v \) is weakly increasing on the path from \( u' \) to \( u'' \) (including the two endpoints), and \( v \) is weakly decreasing on the path from \( u'' \) to \( u' \) (including the two endpoints). All three circular vectors of Figure 5.2 are Bitonic. A circular vector \( v \) is Unitonic if there is simple path that covers all of \( C^n \) and \( v \) is weakly increasing on this path. Vectors (b) and (c) of Figure 5.2 are Unitonic. A vector \( v \) with \( n \) keys is a permutation if every key of the interval \( [1, n] \) appears in \( v \); hence, every such key appears exactly once in \( v \). For example, vector (c) of Figure 5.2 is a permutation. The concept of Unitonic permutations is important due to the following lemma of [10].
Lemma 5.4.1 ([10]). A comparator network is a Bitonic sorter if and only if it sorts all Unitonic permutations of the appropriate width.

Recall that we assume that \( \mathbb{K} = \mathbb{Q} \). Let \( v' \) and \( v'' \) be two vectors over an index set \( X \). The *distance between \( v' \) and \( v'' \)* is defined by \( \delta(v', v'') = \max_{x \in X} |v'_x - v''_x| \). Let \( e \) be an edge and \( v \) be an input vector of a comparator network \( N \). We define \( v(e) \) to be the key transmitted on \( e \) when \( v \) is fed into \( N \). For a set \( V \) of input vectors of \( N \), define \( V(e) \triangleq \{ v(e) | v \in V \} \). Let \( \mathcal{P}^n \), \( \mathcal{B}^n \) and \( \mathcal{U}^n \) denote the set of permutations, Bitonic permutations and Unitonic permutations, all of width \( n \), respectively. The following lemma is due to Alekseyev [14, Section 5.3.4 Exercise 25].

**Lemma 5.4.2.** Let \( e \) be an edge of a comparator network of width \( n \). Then \( \mathcal{P}^n(e) \) is an interval.

Lemma 5.4.2 follows from the following two lemmas. The first lemma can be established by a simple induction.

**Lemma 5.4.3.** Let \( x \) and \( y \) be two input vectors of a comparator network \( N \) and let \( e \) be an edge of \( N \). Then \( |x(e) - y(e)| \leq \delta(x, y) \).

For the next lemma, we use the following terminology. Two vectors, \( v' \) and \( v'' \), are *1-neighbors* if they are over the same index set and \( \delta(v', v'') = 1 \). For a set \( V \) of vectors over the same index set, we define the binary relation ‘\( V \)-neighbor’ as the transitive closure of the ‘1-neighbor’ relation. Namely, two vectors, \( v' \) and \( v'' \), are \( V \)-neighbors if there is a sequence of vectors, all members of \( V \), in which the first is \( v' \), the last is \( v'' \) and every two consecutive vectors are 1-neighbors. Clearly, the ‘\( V \)-neighbor’ relation is an equivalent relation. The following observation is straightforward.

**Lemma 5.4.4.** Every two members of \( \mathcal{P}^n \) are \( \mathcal{P}^n \)-neighbors.

As said, Lemma 5.4.2 easily follows from Lemmas 5.4.3 and 5.4.4. We next prove a variant of Lemma 5.4.2 in which \( \mathcal{P}^n \) is replaced with \( \mathcal{B}^n \). To this end, it suffices to prove a variant of Lemma 5.4.4 in which \( \mathcal{P}^n \) is replaced with \( \mathcal{B}^n \). We use the following terminology. Let \( k' \) and \( k'' \) be two keys that appear in a permutation \( p \). Define \( p^{k' \leftrightarrow k''} \) to be the permutation derived from \( p \) by swapping the keys \( k' \) and \( k'' \). Clearly, \( \delta(p, p^{k' \leftrightarrow k''}) = |k' - k''| \). Assume \( p \) is a permutation over the index set \( C^n \). We say that \( k' \) and \( k'' \) are *adjacent in \( p \)* if they are associated with consecutive vertices of \( C^n \).
Let $b \in \mathcal{B}^n$. Then following two observations are straightforward:

**Observation 1:** $b^{1\rightarrow 2}, b^{n-1\rightarrow n} \in \mathcal{B}^n$.

**Observation 2:** Let $k \in [1, n-1]$ such that $k$ and $k + 1$ are not adjacent in $b$. Then $b^{k\rightarrow k+1} \in \mathcal{B}^n$.

The last two observations imply the following one.

**Observation 3:** For every $b \in \mathcal{B}^n$ there is a $u \in \mathcal{U}^n$ such that $b$ and $u$ are $\mathcal{B}^n$-neighbors and $b_x = u_x = 1$ for some index $x \in \mathcal{C}^n$.

Recall that ‘$\mathcal{B}^n$-neighbor’ is an equivalent relation. By Observations (1) and (3):

**Observation 4:** Every two members of $\mathcal{U}^n$ are $\mathcal{B}^n$-neighbors.

Observations (3) and (4) imply the desired variant of Lemma 5.4.4.

**Lemma 5.4.5.** Every two members of $\mathcal{B}^n$ are $\mathcal{B}^n$-neighbors.

Lemmas 5.4.3 and 5.4.5 imply the required variant of Alekseyev’s result.

**Lemma 5.4.6.** Let $N$ be a comparator network such that $\mathcal{I}(N) = \mathcal{C}^n$ and let $e$ be an edge of $N$. Then $\mathcal{B}^n(e)$ is an interval.

Note that Lemma 5.4.6 does not require the network in question to be a Bitonic sorter.

Let $e$ be an edge of a comparator network $N$. Define $\mathcal{I}(e) \subseteq \mathcal{I}(N)$ to be the set of indexes associated with input ports that have a path to $e$. Similarly, define $\mathcal{O}(e) \subseteq \mathcal{O}(N)$ to be the set of indexes associated with output ports that are reachable from $e$. The input depth of $e$, denoted $d^I(e)$, is the maximal number of comparators along a path from an input port to $e$, not including the comparator at the end of $e$ (if there is such a comparator). Similarly, the output depth of $e$, denoted $d^O(e)$, is the maximal number of comparators along a path from $e$ to an output port, not including the comparator from which $e$ exits (if there is such a comparator). Clearly,

$$|\mathcal{I}(N)| \leq 2^{d^I(e)} \quad \text{and} \quad |\mathcal{O}(N)| \leq 2^{d^O(e)}$$

Let $d(N)$ denote the depth of $N$, that is, the maximal number of comparators on a directed path in $N$. Let $c$ be a comparator of a Bitonic sorter. We say that $c$ is degenerate if its incoming edges can be named $e'$ and $e''$ such that $b'(e) \leq b''(e)$, for every two input vectors, $b'$ and $b''$, which are Bitonic.
Lemma 5.4.7. Let $N$ be a Bitonic sorter with no degenerate comparators. Then $O(e)$ is an interval, for every edge $e$.

Proof. The proof is by induction on $d^O(e)$. The case where $d^O(e) = 0$ is trivial. Assume otherwise and note the following fact. For two intervals of integers, $I'$ and $I''$, $I' \cup I''$ is an interval if and only if there exist $i' \in I'$ and $i'' \in I''$ such that $|i' - i''| \leq 1$.

Let $e'$ and $e''$ be two edges that enter a comparator $c$ and let $f'$ and $f''$ be the two outgoing edges of $c$. By Lemma 5.4.6, $B^c(e')$ and $B^c(e'')$ are intervals. Since $c$ is not degenerate, it follows that $B^c(e') \cup B^c(e'')$ is an interval. Clearly, $B^c(f') \cup B^c(f'') = B^c(e') \cup B^c(e'')$; hence, $B^c(f') \cup B^c(f'')$ is an interval. By the above fact, there are $i' \in B^c(f')$ and $i'' \in B^c(f'')$ such that $|i'' - i'| \leq 1$. Clearly, $B^c(f') \subset O(f')$ and $B^c(f'') \subset O(f'')$; that is, $i' \in O(f')$ and $i'' \in O(f'')$. By the induction hypothesis, $O(f')$ and $O(f'')$ are intervals. Again by the above fact, $O(f') \cup O(f'')$ is an interval. Clearly, $O(e') = O(e'') = O(f') \cup O(f'')$.

Note that Lemma 5.4.6 holds for every comparator network with no concern to the network’s functionality. Lemma 5.4.7, on the other hand, refers to Bitonic sorters and does not hold in general. Recall that our goal is to prove Theorem 20; that is, to show that $D^n > \lceil \log(n) \rceil$. We need Lemma 5.4.7 to this end but we use it much later.

A major tool in our analysis is the following concept of span. Let $e$ be an edge of a comparator network $N$. The span of $e$, denoted $S(e)$, is defined by $S(e) \triangleq |I(e)| \cdot |O(e)|$. The following lemma is straightforward.

Lemma 5.4.8. Let $e$ be an edge of a comparator network. Then a directed path having $\lceil \log(S(e)) \rceil$ comparators passes through $e$.

Lemma 5.4.9. Let $e$ be an edge of a Bitonic sorter of width $n$. Then $S(e) \geq n$.

Proof. Let $I = I(e)$. The set $I$ partitions the circle $C^n$ into $|I|$ disjoint segments; each segment starts at a member of $I$ and ends just before the next member of $I$. Clearly, the length of one of these segments is at least $\lceil n/|I| \rceil$. Let $s$ be such a segment.

Let $U \subseteq U^n$ be the set of Unitonic permutations in which the maximal key, $n$, is located in $s$. Every comparator which leads to $e$ performs the same routing for all members of $U$. That is, there is an index $x \in C^n$ such that $u(e) = u_x$, for every $u \in U$. This implies that $u'(e) \neq u''(e)$ for every $u'$ and $u''$, distinct members of $U$. Hence $|U^n(e)| \geq |U(e)| = |U| = |s| \geq n/|I|$. This clearly implies that $|O(e)| \geq n/|I|$. Hence, $S(e) = |O(e)| \cdot |I| \geq n$.

Lemmas 5.4.8 and 5.4.9 imply the weak inequality, $D(n) \geq \lceil \log(n) \rceil$. In order to prove the strong inequality $\lceil D(n) > \lceil \log(n) \rceil \rceil$, it remains to show the following lemma.
Lemma 5.4.10. Let $B$ be a Bitonic sorter whose width is not a power of two and assume $B$ has no degenerate comparators. Then $B$ has two edges $e'$ and $e''$ such that $S(e'') \geq 2 \cdot S(e')$.

Proof. Let $p$ be the output port indexed by 1. Consider the subgraph $G$ of $B$ composed of all comparators, edges and input ports that have a path to $p$. Since the width of $B$ is not a power of two, $G$ is not a balanced tree. Hence, $G$ has (at least one) comparator $c$ with incoming edges $e'$ and $e''$ and outgoing edges $f'$ and $f''$, for which one (or both) of the followings conditions hold:

1. $d^I(e') \neq d^I(e'')$.

2. $f'$ and $f''$ belong to $G$.

We refer to such a comparator as a bad comparator. Let $\bar{c}$ be a minimal bad comparator; that is, $\bar{c}$ is not reachable from any other bad comparator.

First assume that Condition (1) holds w.r.t. $\bar{c}$. Assume, without loss of generality, that $d^I(e'') > d^I(e')$. The minimality of $\bar{c}$ implies that $|I(e'')| = 2^{d^I(e'')} \geq 2 \cdot 2^{d^I(e')} = 2 \cdot |I(e')|$. Clearly, $|O(e')| = |O(e'')|$. This implies that $S(e'') \geq 2 \cdot S(e')$.

Next assume that Condition (1) does not hold; hence, Condition (2) holds. By Lemma 5.4.7, $O(f')$ and $O(f'')$ are intervals. By construction, $1 \in O(f')$ and $1 \in O(f'')$; hence, one interval is a subset of the other. Say, $O(f') \subseteq O(f'')$. By the minimality of $\bar{c}$ and the fact that Condition (1) does not hold, we get that $|I(f'')| = 2 \cdot |I(e'')|$. However $O(f'') = O(e'')$. This implies that $S(f'') = 2 \cdot S(e'')$.  

Clearly, a degenerate comparator of a Bitonic sorter can be removed without disturbing the network’s functionality and without increasing its depth. This fact, together with Lemmas 5.4.8,5.4.9 and 5.4.10, imply Theorem 20.
Chapter 6

Bitonic sorters of minimal depth

Tamir Levi, Technion, Haifa
Prof’ Ami Litman, Technion, Haifa

Building on previous works, this paper establishes that the minimal depth of a Bitonic sorter of $n$ keys is $2 \lceil \log(n) \rceil - \lfloor \log(n) \rfloor$.

6.1 Introduction

A Bitonic sorter is a comparator network that sorts every Bitonic input sequence. This work studies the minimal depth of such networks. Building on previous works, it establishes that:

Theorem 21. The minimal depth of a Bitonic sorter of $n$ keys is $2 \lceil \log(n) \rceil - \lfloor \log(n) \rfloor$.

When $n$ is a power of two, $2 \lceil \log(n) \rceil - \lfloor \log(n) \rfloor = \log(n)$. The fact that, in this case, $\log(n)$ is the minimal depth is due to the seminal work of Batcher [1]. However, the minimal depth of Bitonic sorters, in the general case, was unknown. This paper constructs, for any $n$, a Bitonic sorter of depth $\lceil \log(n) \rceil + 1$. By [18], these Bitonic sorters are of minimal depth, when $n$ is not a power of two.

This paper also studies the concept of symmetry of comparator networks. Such a symmetry can be manifested in two forms; one form refers to the structure of the network and the other form refers to the input-to-output mapping computed by the network. To wit, a comparator network is strongly symmetric if its structure is symmetric. This form of symmetry was studied in [29]. A comparator network is weakly symmetric if its input-to-output transformation is symmetric. These terms are precisely defined and discussed in Section 6.2. This paper uses only the weaker form of symmetry and the strong symmetry is discussed only for completeness.
The concept of symmetry allows us to extend the well-known 0-1 Principle [14] as follows. A binary sequence of keys is 1-heavy if at least half of its keys are 1. Similarly, it is 0-heavy if at least half of its keys are 0. (A binary sequence with an even number of keys may be 0-heavy and 1-heavy simultaneously). As said, Section 6.2 provides a precise definition of ‘weakly symmetric comparator network’ and establishes the following extension of the 0-1 Principle:

**Theorem 22.** Let $S$ be a weakly symmetric comparator network. Then the following four statements are equivalent:

- $S$ sorts every input sequence.
- $S$ sorts every binary input.
- $S$ sorts every binary 0-heavy input.
- $S$ sorts every binary 1-heavy input.

Replacing, in Theorem 22, the term ‘weakly’ with ‘strongly’ reduces the power of this theorem. It is not hard to see that there are many comparator networks that are weakly symmetric and are not strongly symmetric. Moreover, there are some tasks that can be performed by a weakly symmetric comparator network but not by a strongly symmetric comparator network. Such a task, for example, is the sorting of an odd number of keys, as shown in Section 6.2.

The 0-1 Principle has many variants related to other functionalities, for example, merging and Bitonic sorting. The same holds for Theorem 22 and several of its variants are presented in Section 6.2. One of these variants, related to Bitonic sorting, is used in our construction. The ability to consider only 0-heavy (or only 1-heavy) inputs significantly simplifies the analysis and design of our networks.
Another contribution of this paper is a compact and coherent graphical presentation of comparator networks, presented in Figure 6.5.

### 6.1.1 Related work

Let $T(n)$ denote the minimal depth of a Bitonic sorter of $n$ keys. We now summarize what was previously known on the function $T$. By a straightforward reachability argument, for every $n$,

$$\lceil \log(n) \rceil \leq T(n). \quad (6.1)$$

Due to the constructions of Batcher [1]:

$$T(2n) \leq T(n) + 1. \quad (6.2)$$

This and Inequality (6.1) imply that:

$$T(2^j) = j. \quad (6.3)$$

Nakatani et al. [25] established that:

$$T(i \cdot j) \leq T(i) + T(j). \quad (6.4)$$

The only prior technique that constructs Bitonic sorters of any width is due to Batcher and Liszka [22]. They show that:

$$T(n) \leq \max \left( T(\left\lceil \frac{n}{2} \right\rceil), T(\left\lfloor \frac{n}{2} \right\rfloor) \right) + 2.$$

This and a straightforward induction imply:

$$T(n) \leq 2 \lceil \log(n) \rceil - 1. \quad (6.5)$$

The first, non-trivial, lower bound on $T(n)$ is due to Levy and Litman [18]. They showed that for every $n$ that is not a power of two:

$$\lceil \log(n) \rceil + 1 \leq T(n). \quad (6.6)$$

This result, combined with Equality 6.3, yields the surprising corollary that $T$ is not monotonic. For example, $T(15) \geq 5 > 4 = T(16)$.
As said, our main result is the exact value of $T(n)$. Namely, for every $n$:

$$T(n) = 2 \lceil \log(n) \rceil - \lfloor \log(n) \rfloor .$$  \hfill (6.7)

In other words,

$$T(n) = \begin{cases} 
\log(n), & \text{when } n \text{ is a power of two} \\
\lceil \log(n) \rceil + 1, & \text{otherwise}
\end{cases}$$

Another model of oblivious computation, called min-max networks, was studied by Levy and Litman [19]. (These networks are discussed in Section 6.1.2.) Let $T'(n)$ denote the minimal depth of a min-max network that sorts all Bitonic sequences of $n$ keys. Due to our Theorem 21, the exact value of $T'(n)$ is almost known, as implied by the following arguments.

The same reachability argument imply Inequality (6.1) also for min-max networks; therefore:

$$\lfloor \log(n) \rfloor \leq T'(n)$$  \hfill (6.8)

Since every comparator network can be translated to a min-max network of the same depth, it follows that for every $n$:

$$T'(n) \leq T(n)$$  \hfill (6.9)

Inequalities (6.8,6.9) and Theorem 21 imply that for every $n$:

$$\lfloor \log(n) \rfloor \leq T'(n) \leq \lceil \log(n) \rceil + 1.$$

There are certain cases in which the exact value of $T'(n)$ is known, as listed below. The exact value of $T'(n)$ for other cases is yet unknown.

- $T'(n) = \log(n)$ when $n$ is a power of two. This follows from Inequalities (6.3,6.8) and (6.9).
- $T'(n) = \lceil \log(n) \rceil + 1$, for every odd $n$. Levy and Litman [18] established that $\lfloor \log(n) \rfloor + 1 \leq T'(n)$, for every odd $n$. Inequality (6.9) and Theorem 21 provide the matching upper bound.
- $T'(n) = \lceil \log(n) \rceil$ for $n \in (10 \cdot 2^N)$. This was established in [18].
- $T'(n) = \lceil \log(n) \rceil$ for $n \in (6 \cdot 2^N)$, as shown in the next paragraph.

Figure 6.2 depicts a min-max network, presented in [19], which is a Bitonic sorter of 6 keys and of depth 3. Hence, $T'(6) = 3$. Due to this network, $T'(6 \cdot 2^i) = 3 + i$ as follows. The techniques of Nakatani [25] and Batcher [1] are applicable also to min-max networks; hence, Inequalities (6.2) and 6.4 hold also for $T'$. Together with Inequality (6.8), we get that $T'(6 \cdot 2^i) = 3 + i = \lceil \log(6 \cdot 2^i) \rceil$. 

As discussed in [18], the above examples imply that min-max networks are sometimes strictly faster than comparator networks. Namely, there are infinitely many $n$’s with $T'(n) < T(n)$; this holds at least for any $n$ of the form $n = 6 \cdot 2^i$ or $n = 10 \cdot 2^i$.

The work of this paper can be generalized in two directions. One direction considers the same computational problem but under a different model of computation. The above discussion on $T'$ follows this direction. Another direction keeps the same model of computation but considers harder computational problems. A natural generalization of our problem is the problem of sorting “multitonic sequences”, studied by Seiferas [28].

6.1.2 Preliminaries

To be self contained, we provide the following definitions. A comparator is a combinational device that receives, via two incoming edges, two keys and sorts them. It has two outgoing edges; on one of them, called the Min-edge, it sends the minimal key and on the other outgoing edge, called the Max-edge, it sends the maximal key. A comparator network is an acyclic network of comparators. See Figure 6.1. In this figure, a solid arrowhead denotes a Max-edge and a hollow arrowhead denotes a Min-edge. Keys enter a comparator network via its input ports and exit the network via its output ports. The fan-out of input ports and the fan-in of output ports is exactly one. These ports are depicted by bullets. The network specifies, in some form, how the input is fed to the input ports and how the output is assembled from the output ports. A comparator network has the same number of input ports and output ports; this number is referred to as the width of the network. The depth of the network is the maximal number of comparators on a directed path in the network.

For example, Figure 6.1 depicts two famous networks, both of width eight and depth three. One is Batcher’s Bitonic sorter and the other is Batcher’s odd-even merging network [1].

![Figure 6.1: (a) Batcher’s Bitonic sorter; (b) Batcher’s odd-even merging network.](image)

Comparator networks processes keys which are members of some ordered set, $\mathbb{K}$. The exact nature of keys is usually not important. This paper focuses on comparator networks that sort Bitonic sequences. A sequence of keys is Bitonic if it is a rotation of a concatenation of two sequences – an ascending sequence followed by a descending one. A Bitonic sorter is a comparator network that sort any Bitonic sequence of the appropriate width. As said, the famous Batcher’s Bitonic sorter of width 8 is depicted in Figure 6.1(a).

Another model of oblivious computation, the min-max model, was studied by Levy and Litman [19]. Figure 6.2 depicts a min-max network, presented in [19], that is a Bitonic sorter of 6 keys. A min-max network is an acyclic network of MIN-gates and MAX-gates. The fan-in of these gates is exactly two. These gates compute the minimum and maximum of their two input keys, respectively. Note that there is no restriction on the fan-out of the gates and of the input ports. Graphically, MIN-gates are depicted by hollow triangles and MAX-gates are depicted by solid triangles.
The current paper concerns comparator networks rather than min-max networks; therefore, in the rest of the paper, a network means a comparator network and a Bitonic sorter means a comparator network that sorts every Bitonic sequence of the appropriate length.

This work relies heavily on the famous 0-1 Principle [14] and its variant related to Bitonic sorters. Due to that, we henceforth assume that the input to our networks is binary. We refer to a binary sequence as a word and denote the set of all words by \( \{0, 1\}^* \). A vector is a sequence of words. Usually, words are denoted by lowercase letters and vectors are denoted by uppercase letters. For any finite sequence \( s \) (e.g., a word or a vector), let \( |s| \) denote the number of members in \( s \). We refer to \( |s| \) as the width of \( s \). The members of \( s \) are denoted by \( \langle s_1, s_2 \ldots s_{|s|} \rangle \). By abuse of notation, we usually do not distinguish between a sequence of a single element and that element by itself. For every \( k \in \mathbb{N} \), a \( k \)-vector is a vector \( V \) with \( |V| = k \); let \( \{0, 1\}^*^k \) denote the set of all \( k \)-vectors. Let \( \{0, 1\}^{**} \) denote the set of all vectors.

For a vector \( V \), let \( n^0(V) \) and \( n^1(V) \) denotes the number of 0’s and 1’s in \( V \), respectively. Clearly, \( n^0(V) + n^1(V) = \sum_{i=1}^{|V|} |V_i| \).

### 6.2 Symmetric Oblivious Algorithms

This chapter defines and studies symmetric oblivious algorithms. The main results of the section are Theorems 23, 24 and 25. They state that a weakly symmetric network (a sorting network, a merging network or a Bitonic sorter) operates properly if and only if it operates properly on all the 0-heavy inputs (or on all the 1-heavy inputs).
Symmetry of oblivious algorithms can be manifested in two forms. The weaker form takes a Black Box approach; it considers only the input-output transformation defined by the algorithm and ignores its internal working. The stronger form of symmetry concerns the structure of the network associated with the algorithm. The latter form is studied in [29]. However, this paper demonstrates that it is more convenient to work with the Black Box approach.

Recall that inputs to our algorithms are restricted to be binary vectors. In fact, our concept of weak symmetry is meaningful only under this restriction. To define this concept, the following terminology is used. A signature of a vector $V$, denoted $||V||$, is defined by $||V|| \triangleq \langle |V_1|, |V_2|, \ldots, |V_{||V||}| \rangle$. Two vectors are isomeric if they have the same number of 0's and the same number of 1's. The following discussion concerns functions and expressions that are sometimes undefined. In this context, the symbols ‘$=$’ and ‘$\triangleq$’ also mean that the left side of the equation is defined if and only if the right side is defined.

The weaker concept of symmetry is based on special mappings, which we call operators, and are defined by:

**Definition 6.2.1.** An operator is a partial mapping $\alpha : \{0,1\}^* \rightarrow \{0,1\}^*$ with the following properties:

a. $\alpha$ is isomeric; that is, for a vector $V$, $\alpha(V)$ and $V$ are isomeric whenever $\alpha(V)$ is defined.

b. $||\alpha(U)|| = ||\alpha(V)||$, for every two vectors, $V$ and $U$, such that $||U|| = ||V||$.

Note that Requirement (b) of Definition 6.2.1, combined with the special meaning of ‘$=$’, implies that, for every signature $s$, either $\alpha$ is defined on all vectors with signature $s$ or $\alpha$ is defined on none of these vectors. In the former case we say that $\alpha$ is defined on the signature $s$ or that $s$ is an input signature of $\alpha$.

In this paper, a network receives and produces vectors. Under this convention, a network $N$ computes an operator. This operator has exactly one input signature and we refer to this signature as the input signature of $N$. Every vector with this signature is called an input vector of $N$. A vector produced by $N$, under some input vector, is called an output vector of $N$. Clearly, all the output vectors of a network share the same signature which is called the output signature of $N$. However, in contrast to the input vectors, not all vectors with the output signature of $N$ are output vectors of $N$.

To illustrate this terminology, let $M$ be a merging network that merges two sorted words, each of width 7, into a single sorted word of width 14. The input signature of $M$ is $(7, 7)$ and its output signature is $(14)$. That is, every vector with signature $(7, 7)$ is a meaningful input vector of $M$. Namely, by our terminology, the two words of an input vector of $M$ are not required to be sorted. Note that all the above details about $M$ do not determine the operator computed by $M$. That is, two merging networks with the same input signature may compute different operators. This contrasts sorting networks in which the input signature completely determine the operator.
Note that an operator may have several input signatures while a network has a single input signature. This leads to the following terminology. For an operator \( \alpha \) and signature \( s \), the *restriction of \( \alpha \) to \( s \),* denoted \( \alpha \upharpoonright_s \), is the operator defined by:

\[
\alpha \upharpoonright_s (V) \triangleq \begin{cases} 
\alpha(V), & ||V|| = s \\
\text{Undefined,} & \text{otherwise.}
\end{cases}
\]

An operator \( \alpha \) is *computable* (by networks) if, for every input signature \( s \) of \( \alpha \), there is a network that computes \( \alpha \upharpoonright_s \). For a natural number \( d \), an operator \( \alpha \) is *of depth \( d \)* if, for every input signature \( s \) of \( \alpha \), there is a network of depth at most \( d \) that computes \( \alpha \upharpoonright_s \).

An operator \( \alpha \) is *total* if \( \alpha(V) \) is always defined. An operator \( \alpha \) is called a *\( k \)-to-\( j \) operator* if its domain is a subset of \( \{0, 1\}^* \) \( k \) and its range is a subset of \( \{0, 1\}^* \) \( j \). A network that computes a \( j \)-to-\( k \) operator is called a *\( j \)-to-\( k \) network*. We now present several operators.

- **The Concatenation operator**: This operator, denoted \( C \), is a total operator of depth 0 that concatenates all the words of its argument into a single word. Formally, \( C(V) \triangleq V_1 \cdot V_2 \cdots \cdot V_{|V|} \), where \( \cdot \) denotes concatenation of sequences.

- **The Reverse operators**: We present three reverse operators, all are total and of depth zero. The first reverses the order of words within a vector; the second reverses the order of keys within the words of a vector and the third do both. These operators are denoted by \( \leftarrow \cdot \), \( \leftarrow \circ \) and \( \leftarrow \cdot \).
The first operator, ‘\(\leftarrow\)’, is defined by:
\[
\leftarrow (\langle V_1, V_2, \ldots, V_{|V|} \rangle) \triangleq \langle V_{|V|}, \ldots, V_2, V_1 \rangle
\]
The mapping ‘\(\cdot\)’ is defined on words by:
\[
\cdot (\langle w_1, w_2, \ldots, w_{|w|} \rangle) \triangleq \langle w_{|w|}, \ldots, w_2, w_1 \rangle
\]
The operator ‘\(\odot\)’ is defined on vectors by:
\[
\odot (\langle V_1, V_2, \ldots, V_{|V|} \rangle) \triangleq \langle \odot (V_1), \odot (V_2), \ldots, \odot (V_{|V|}) \rangle
\]
The operator ‘\(\rightarrow\)’ is defined by:
\[
\rightarrow (V) \triangleq \odot (\leftarrow (V))
\]

- **The MinMax operator**: Let us start by defining the following two natural mappings, \(\text{Min}\) and \(\text{Max}\), which are not operators. These mappings are defined on 2-vectors, \(\langle s, r \rangle\) with \(|s| = |r|\) by

\[
\text{Min}(\langle s, r \rangle) \triangleq (\min(s_1, r_1), \min(s_2, r_2), \ldots, \min(s_{|s|}, r_{|r|}))
\]
\[
\text{Max}(\langle s, r \rangle) \triangleq (\max(s_1, r_1), \max(s_2, r_2), \ldots, \max(s_{|s|}, r_{|r|}))
\]
The MinMax operator is a 2-to-2 operator of depth 1 defined on 2-vectors \(\langle s, r \rangle\) with \(|s| = |r|\) by \(\text{MinMax}(\langle s, r \rangle) \triangleq (\text{Min}(\langle s, r \rangle), \text{Max}(\langle s, r \rangle))\).

- **The Batcher operator**: This operator, denoted \(\mathcal{B}\), is performed by the first stage of Batcher’s Bitonic sorter [1]. It is a 1-to-2 operator of depth one. This operator is defined only on words of even width. For such a word \(w = w' \cdot w''\) with \(|w'| = |w''|\), \(\mathcal{B}\) is defined by \(\mathcal{B}(\langle w', w'' \rangle) \triangleq \text{MinMax}(\langle w', w'' \rangle)\).

Two operators, \(\alpha\) and \(\beta\), can be combined into a single operator in two manners – a sequential one and a parallel one. In the sequential manner \(\alpha\) and \(\beta\) are preformed one after another; this results in the composition of \(\alpha\) and \(\beta\), denoted \(\beta \circ \alpha\); that is, \((\beta \circ \alpha)(V) \triangleq \beta(\alpha(V))\), for any \(V\). Note that, by the special meaning of ‘\(\triangleq\)’, \((\beta \circ \alpha)(V)\) is defined if and only if \(\alpha(V)\) is defined and \(\beta(\alpha(V))\) is defined.

In the parallel manner, the two operators \(\alpha\) and \(\beta\) are performed simultaneously; to this end, the input vector \(V\) is divided, in a certain manner, into two vectors; one vector is given to \(\alpha\) and the other to \(\beta\). The two resulting vectors are combined, by concatenation, into a single vector. A subtle issue is the division of the input vector into arguments for \(\alpha\) and \(\beta\). We avoid this issue by applying the ‘\(+\)’ operation only on operators of the following form; let \(\alpha\) be a \(j^\alpha\)-to-\(k^\alpha\) operator and \(\beta\) be a \(j^\beta\)-to-\(k^\beta\) operator, for some \(j^\alpha, k^\alpha, j^\beta, k^\beta\). In this case, \((\alpha + \beta)\) is defined to be the \((j^\alpha + j^\beta)\)-to-\((k^\alpha + k^\beta)\) operator satisfying \((\alpha + \beta)(V' \cdot V'') \triangleq \alpha(V') \cdot \beta(V'')\) for \(|V'| = j^\alpha\) and \(|V''| = j^\beta\). If \(\alpha\) or \(\beta\) are not of the above form, \((\alpha + \beta)\) is defined to be the empty operator – the operator that is never defined.
Note that each of the operations ‘◦’ and ‘+’ is associative. Hence, expressions like $\alpha + \beta + \gamma$ and $\alpha \circ \beta \circ \gamma$ are meaningful; that is, parentheses may be omitted in expressions that involve at most one of the operations ‘+’ and ‘◦’. However, an expression like $\alpha + \beta \circ \gamma$ is meaningless. By our terminology, Batcher’s Bitonic sorter [1] of width 8 is composed as follows:

$$\mathcal{C} \circ (\mathcal{B} + \mathcal{B} + \mathcal{B} + \mathcal{B}) \circ (\mathcal{B} + \mathcal{B}) \circ \mathcal{B}$$

When the operator $(\mathcal{B} + \mathcal{B} + \mathcal{B} + \mathcal{B}) \circ (\mathcal{B} + \mathcal{B}) \circ \mathcal{B}$ is applied on a Bitonic word of width 8, it produces a vector with signature $\langle 1, 1, 1, 1, 1, 1, 1, 1 \rangle$ that is actually sorted. Next, the operator $\mathcal{C}$ concatenates this vector into a single sorted word.

The duality of Boolean algebra is well-known. In our terminology, this duality swaps zeros with ones, $\text{Min}$ with $\text{Max}$, and ‘≤’ with ‘≥’. This duality is the base of weak symmetry for which the following terminology is used. The negation of a binary key $k$, denoted $\neg(k)$, is defined by $\neg(k) \triangleq 1 - k$. Negation is naturally generalized to words and vector as follows. For a word $w$: define $\neg(w) \triangleq \langle \neg(w_1), \neg(w_2), \ldots, \neg(w_{|w|}) \rangle$ and for a vector $V$, define $\neg(V) \triangleq \langle \neg(V_1), \neg(V_2), \ldots, \neg(V_{|V|}) \rangle$. Note that, by our terminology, the negation transformation is not an operator since it violates Requirement (a) of Definition (6.2.1).

**Definition 6.2.2.** The inverse of a vector $V$, denoted $\mathcal{I}(V)$, is the vector

$$\mathcal{I}(V) \triangleq \overleftarrow{\neg(V)}$$

Namely, $\mathcal{I}(V)$ is derived from $V$ by:

- Swapping zeroes and ones.
- Reversing the order of words of $V$.
- Reversing the order of keys within each word of $V$.

Again, the $\mathcal{I}$ transformation is not an operator. Clearly, $\mathcal{I}(\mathcal{I}(V)) = V$ for every vector $V$.

**Definition 6.2.3.** Let $\alpha$ be an operator.

- The dual operator of $\alpha$, denoted $\hat{\alpha}$, is defined by $\hat{\alpha}(V) \triangleq \mathcal{I}(\mathcal{I}(\alpha(V)))$.
- $\alpha$ is called symmetric if $\alpha = \hat{\alpha}$.

In other words, $\alpha$ is symmetric if and only if $\alpha$ commutes with $\mathcal{I}$. Natural operators are usually symmetric as demonstrated in the following lemma.
Lemma 6.2.1. All the operators ‘$\oplus_\circ$’, ‘$\oplus^*$’, ‘$\leftarrow$', ‘$\mathcal{C}$’, ‘$\mathcal{M}$’ and ‘$\mathcal{B}$’ are symmetric.

Proof. Symmetry of all these operators besides ‘$\mathcal{M}$’ and ‘$\mathcal{B}$’ is straightforward. We focus only on the $\mathcal{M}$ operator. It suffices to show that $\mathcal{M}$ commutes with $I$. To this end, let $x, y \in \{0, 1\}^*$ and $|x| = |y|$. Then

$$\mathcal{M}(I((x, y))) =$$

$$\mathcal{M}(I(y), I(x)) =$$

$$\langle \text{Min}(I(y), I(x)), \text{Max}(I(y), I(x)) \rangle =$$

$$\langle \text{Min}(\neg(\leftarrow(y)), \neg(\leftarrow(x))), \text{Max}(\neg(\leftarrow(y)), \neg(\leftarrow(x))) \rangle =$$

$$\langle \neg(\leftarrow(\text{Max}(y, x))), \neg(\leftarrow(\text{Min}(y, x))) \rangle =$$

$$\neg(\leftarrow(\text{Min}(y, x), \text{Max}(y, x))) =$$

$$\neg(\leftarrow(\text{Min}(x, y), \text{Max}(x, y))) =$$

$$I(\mathcal{M}(x, y))$$

As demonstrated in Section 6.3.2, it is beneficial to work with symmetric operators. However, constructing such operators is sometime subtle. Minor modifications of an operator’s definition, which seem insignificant, may turn a symmetric operator to an asymmetric one and vice versa. Consider, for example, the following two 1-to-2 operators, $\psi$ and $\vartheta$. Both operators are defined only over words of even width. For a word $w = w' \cdot w''$ with $|w'| = |w''|$, define:

- $\psi(w) \triangleq \langle \text{Min}(w', \leftarrow(w'')), \text{Max}(\leftarrow(w'), w'') \rangle$

- $\vartheta(w) \triangleq \langle \text{Min}(w', \leftarrow(w'')), \text{Max}(w', \leftarrow(w'')) \rangle$

It is not hard to verify that $\psi$ is symmetric while $\vartheta$ is not symmetric.

The following lemma describe how the properties of duality and symmetry are maintained by the two operations, ‘$+$’ and ‘$\circ$’.
Lemma 6.2.2. Let $\alpha$ and $\beta$ be two operators. Then

\begin{enumerate}
\item $\widehat{\beta \circ \alpha} = \widehat{\beta} \circ \widehat{\alpha}$.
\item $\widehat{\beta + \alpha} = \widehat{\alpha} + \widehat{\beta}$
\item If $\alpha$ and $\beta$ are symmetric, than $\alpha \circ \beta$ is symmetric.
\item If $\alpha = \widehat{\beta}$, than $\alpha + \beta$ is symmetric.
\item Let $\gamma$ be a symmetric operator and let $\alpha = \widehat{\beta}$; then $\alpha + \gamma + \beta$ is symmetric.
\end{enumerate}

We say that a network is weakly symmetric if it computes a symmetric operator. A stronger version of symmetry of networks is discussed shortly. The following theorems are the main results of this section and they concern three functionalities of networks: sorting, merging and Bitonic sorting. They follow from the following trivial facts:

- A word $w$ is sorted if and only if $\mathcal{I}(w)$ is sorted.
- A word $w$ is Bitonic if and only if $\mathcal{I}(w)$ is Bitonic.
- A 2-vector $V$ is a pair of sorted words if and only if $\mathcal{I}(V)$ is a pair of sorted words.
- Let $\alpha$ be a 1-to-1 operator and $w$ a word such that $\alpha(w)$ is sorted. Then $\widehat{\alpha}(\mathcal{I}(w))$ is sorted.

Theorem 23. Let $N$ be a weakly symmetric, 1-to-1 network. Then the following four statements are equivalent:

- $N$ sorts every sequence (of the appropriate width).
- $N$ sorts every word\(^2\).
- $N$ sorts every 0-heavy word.
- $N$ sorts every 1-heavy word.

Theorem 24. Let $N$ be a weakly symmetric, 2-to-1 network. Then the following four statements are equivalent:

- $N$ sorts every pair of sorted sequences (of the appropriate width).
- $N$ sorts every pair of sorted words.
- $N$ sorts every 0-heavy pair of sorted words.
- $N$ sorts every 1-heavy pair of sorted words.

Note that if $\langle i, j \rangle$ is the input signature of a weakly symmetric network, as per Theorem 24, then $i = j$.

\(^2\)Recall that a word is a binary sequence.
Theorem 25. Let $N$ be a weakly symmetric, 1-to-1 network. Then the following four statements are equivalent:

- $N$ sorts every Bitonic sequence (of the appropriate width).
- $N$ sorts every Bitonic word.
- $N$ sorts every 0-heavy Bitonic word.
- $N$ sorts every 1-heavy Bitonic word.

All the above three theorems are interesting; however, this paper builds only on Theorem 25.

So far, we focused on the weaker version of symmetry that considers the network as a Black Box. However, the concepts of symmetry and duality are also relevant w.r.t. the structure of a network, as studied in [29]. Let us define the dual network, $\hat{N}$, of a network $N$ as follows. First, $N$ and $\hat{N}$ have the same graph: the same vertices and the same edges. Next, the types of the edges are flipped; that is, a $\text{Min}$-edge in $N$ becomes a $\text{Max}$-edge in $\hat{N}$ and a $\text{Max}$-edge in $N$ becomes a $\text{Min}$-edge in $\hat{N}$. Still, this is not enough. We also need to reverse the way input is fed into and output is collected from the network. In fact, these changes are equivalent to applying the reverse operator on the input vector and on the output vector. The input signature and output signature of $\hat{N}$ are derived by reversing the appropriate signatures of $N$. Moreover, if an input port $p$ of $N$ receives the $i$'th key of the $j$'th word of the input vector then, in $\hat{N}$, $p$ receives the $i$'th last key of the $j$'th last word of the input vector. Figure 6.3 depicts Batcher’s Bitonic sorter of width 8 and its dual network. Figure 6.4 depicts Batcher’s Odd-Even merging network of width 8 and its dual network.

As said, a network specifies, in a certain form, how the input vector is fed into the input ports and how the output vector is collected from the output ports. However, the networks of Figure 6.4 do not specify which of the two words, $a$ and $b$, is the first and which is the second word of the input vector. Fortunately, when the input vector has exactly two words, the dual network can be constructed without this specification.
A network is strongly symmetric if it is isomorphic to its dual network. This isomorphism, lets call it \( \pi \), needs to preserve (besides the graph topology) the types of edges, the way input is fed into the network and the way output is collected from the network. Namely, assume that the \( i \)’th key of the \( j \)’th word enter an input port \( p \) in a network \( N \). Then in \( \tilde{N} \), the \( i \)’th key of the \( j \)’th word enters \( \pi(p) \). The same holds for the output ports.

It is not hard to verify that the two networks of Figure 6.3 are isomorphic and that the two networks of Figure 6.4 are isomorphic. Hence, the Batcher’s Bitonic sorter of width 8, as well as Batcher’s odd-even merging network of width 8, are strongly symmetric. Moreover, it is not hard to verify that, for every \( n \) that is a power of two, the Batcher’s Bitonic sorter of width \( n \), as well as Batcher’s odd-even merging network of width \( n \), are strongly symmetric.

All the four drawings of Figures 6.3 and 6.4 are in a style that highlights the fact that the corresponding networks are strongly symmetric. In fact, this style is applicable for every strongly symmetric network. In this style, the graph of the network is the mirror image of itself w.r.t. a horizontal mirror. This mirror modifies the other details associated with the network. Namely, it swaps the Min/Max type of the edges and it reverses the association of the input vector to the input ports and the association of the output vector to the output ports.

Figure 6.4: Batcher’s odd-even merging network of width 8 and its dual network.
As shown in [29], a strongly symmetric network is also weakly symmetric. However, the opposite does not necessarily hold. Consider, for example, a 1-to-1 network of width \( n \) which is a sorting network. Such a network clearly computes a symmetric operator. Hence, by definition, such a network is weakly symmetric. However, such a network is not necessarily strongly symmetric. Moreover, when \( n \) is odd and greater then one, such a network is certainly not strongly symmetric due to the following arguments. Consider the edge \( e \) that carries the output key in the middle of the output sequence of \( N \). Clearly, the edge \( e \) emerges from a comparator (and not from an input port). Hence, it has a \textbf{Min/Max} type. By definition, \( e \), has different \textbf{Min/Max} types in \( N \) and in \( \hat{N} \). Since any isomorphism from \( N \) onto \( \hat{N} \) must map \( e \) to itself and must preserve the \textbf{Min/Max} type of \( e \), there is no such isomorphism. Hence, \( N \) is not strongly symmetric.

### 6.3 Certain Bitonic Sorters

As said, this paper constructs, for every \( n \), a Bitonic sorter of \( n \) keys. The depth of each of these Bitonic sorters is \([\log(n)] + 1\). This construction is done in three steps, producing three algorithms\(^3\), called \( A \), \( B \) and \( C \), all of depth \([\log(n)] + 1\), as follows:

1. The algorithm \( A \) sorts every Bitonic 1-heavy word.
2. The algorithm \( B \) is a variant of \( A \) which is defined only when the number of keys is even. This algorithm sorts every 1-heavy Bitonic word of even width; moreover, it is symmetric. Hence, by Theorem 25, \( B \) sorts every Bitonic sequence of even width.
3. The algorithm \( C \) is a variant of \( B \) which is defined only when the number of keys is odd. The great similarity of \( B \) and \( C \) is used to establish that \( C \) sorts every Bitonic sequence of odd width.

Algorithms \( A \) and \( B \) are presented in this section while algorithm \( C \) is presented in Section 6.5.

#### 6.3.1 Sorting 1-heavy words

We define two 1-to-2 operators of depth one, called \( LS \) and \( SL \), that partition a word \( w \) into two words \( \langle x, y \rangle \) such that \(||x| - |y|| \leq 1\). When \( |w| \) is odd, one of the two words \( x \) or \( y \) is longer (by one) from the other word. The mnemonics \( LS \) and \( SL \) indicate which of these words is the shorter one and which is the longer one. That is, ‘\( L \)’ stands for ‘Long’ and ‘\( S \)’ stands for ‘Short’; so, if \( \langle x, y \rangle = LS(w) \) then \( |x| \geq |y| \) and if \( \langle x, y \rangle = SL(w) \) then \( |x| \leq |y| \). To complete the definition of these two operators let \( u \) and \( v \) be two words with \( |u| = |v| \) and let \( b \) be a word with \( |b| \leq 1 \). (Note that \( b \) may be empty). Then

\[
LS(u \cdot b \cdot v) \triangleq \langle Min(u \cdot b, \leftarrow (b \cdot v)), Max(\leftarrow (u), v) \rangle \quad (6.10)
\]

\[
SL(u \cdot b \cdot v) \triangleq \langle Min(u, \leftarrow (v)), Max(\leftarrow (u \cdot b), b \cdot v) \rangle \quad (6.11)
\]

It is not hard to see that each of the operators \( LS \) and \( SL \) is the dual of the other. Moreover, when restricted to words of even width, both operators are identical; therefore, under this restriction, each of them is symmetric. We say that \( \langle x, y \rangle \) is a \textit{split} of a word \( w \), denoted \( \langle x, y \rangle \approx w \), if \( \langle x, y \rangle = LS(w) \) or \( \langle x, y \rangle = SL(w) \).

\(^3\)This is an abuse of terminology. By definition [19], an oblivious algorithm operates on a certain fixed number of keys. Actually, \( A \), \( B \) and \( C \) are families of algorithms – one for each number of keys for which the algorithm is defined.
In this section we use the symbol ‘\(\preceq\)’ in a special manner, as follows. The symbol ‘\(\preceq\)’ has the same meaning as ‘\(\leq\)’ except of the following twist. When one (or both) of the two arithmetic expressions, \(e_1\) and \(e_2\), is undefined then the phrase ‘\(e_1 \preceq e_2\)’ is not meaningless; rather, this phrase is considered to be true. The need for this strange notation arise from the following problem. Splitting a word may generate empty words. For an empty word \(w\), the notations \(w_1\) and \(w|w|\) are undefined and the special meaning of ‘\(\preceq\)’ is useful in this context. To illustrate the convenience of these notations consider the following lemma whose proof is straightforward. Note that we consider the expressions \(\min(e', e'')\) and \(\max(e', e'')\) to be undefined whenever \(e'\) or \(e''\) (or both) are undefined.

**Lemma 6.3.1.** Let each of \(q, r, s\) and \(t\) be either binary or undefined and let \(q + r \preceq s + t\). Then \(\min(q, r) \preceq \min(s, t)\) and \(\max(q, r) \preceq \max(s, t)\).

The following two lemmas concern the splitting of Bitonic words. Although the proofs of these lemmas are tedious, they are straightforward and therefore omitted.

**Lemma 6.3.2.** Let \(v\) be a Bitonic word and let \(v \approx \langle x, y \rangle\). Then:

a. \(x \cdot y = v\) or \(x \cdot y = \leftarrow(v)\).

b. \(x\) is descending-ascending.

c. \(y\) is ascending-descending.

d. If \(v\) is 1-heavy then \(y\) is 1-heavy and \(1 \preceq y_1 + y_{|y|}\).

e. If \(\langle x, y \rangle = \text{LS}(v)\) then \(n^0(x) \preceq n^0(y)\); if, in addition, \(v\) is 1-heavy then \(x_1 + x_{|x|} \preceq y_1 + y_{|y|}\).

**Lemma 6.3.3.** Let \(v\) be descending-ascending word and let \(v \approx \langle x, y \rangle\). Then:

a. \(y\) is ascending.

b. \(x_1 + x_{|x|} \preceq v_1 + v_{|v|}\).

c. If \(\langle x, y \rangle = \text{LS}(v)\) then \(x_{|x|} \preceq y_1\).

Define \(0^k\) and \(1^k\) as the words of width \(k\) that contain only 0’s and only 1’s, respectively. Define \(0^* \triangleq \{0^k | k \in \mathbb{N}\}\) and \(1^* \triangleq \{1^k | k \in \mathbb{N}\}\). The algorithm \(A\) starts by producing a ‘chunk’ of all 1’s containing approximately a quarter of the input keys, as per the following lemma.
Lemma 6.3.4. Let \( w \) be 1-heavy and Bitonic, let \( w \approx (a, b) \) and let \( b \approx (c, r) \). Then \( r \in 1^* \).

Proof. By Lemma 6.3.2(c,d), the word \( b \) is ascending-descending and \( 1 \preceq b_1 + b_{|b|} \). Therefore, \( b \) is either ascending or descending. Again, by Lemma 6.3.2(d), \( b \) is 1-heavy. This clearly implies that \( r \in 1^* \).

The core of the algorithm \( A \) is a certain iterative process, whose invariant is the following \( \diamond \)-property.

Definition 6.3.1. The \( \diamond \)-property is the following property of a 3-vector \( \langle d, m', m'' \rangle \):

a. \( d \) is descending-ascending.

b. \( m' \) and \( m'' \) are sorted and \( |m'| = |m''| \).

c. \( |n^0(m') - n^0(m'')| \leq n^0(d) \).

d. \( d_1 + d_{|d|} \preceq m'_1 + m''_1 \).

The algorithm \( A \) is composed of three parts. The first part, called the prologue, contains two stages, each of depth one. The first stage splits the input word into two words \( \langle a, b \rangle \). The second stage splits the word \( a \) into two words \( \langle d, m' \rangle \) and splits the word \( b \) into two words \( \langle m'', r \rangle \). It comes out that there is a way to perform these splits such that the vector \( \langle d, m', m'' \rangle \) will have the \( \diamond \)-property, as described in the following lemma.

Lemma 6.3.5. Let \( w \) be a 1-heavy Bitonic word. Let \( \langle a, b \rangle = \mathcal{L}S(w) \), let \( \langle d, m' \rangle = \mathcal{L}S(a) \) and let \( \langle m'', r \rangle \approx b \) such that \( |m'| = |m''| \). Then \( \langle d, m', m'' \rangle \) have the \( \diamond \)-property.

Proof. We verify the four conditions of Definition 6.3.1, as follows.

Condition (a) follows from Lemma 6.3.2(b).

Consider Condition (b). By the premiss of the lemma, \( |m'| = |m''| \). By Lemma 6.3.2, \( a \) is descending-ascending and \( b \) is ascending-descending. By Lemma 6.3.3(a) (with \( a \) and \( m' \) in place of \( v \) and \( y \) \( m' \) is sorted. By duality, the same holds for \( m'' \) and \( b \); that is, \( m'' \) is also sorted.

Consider Condition (c). By Lemma 6.3.2(e):

\[
n^0(d) \geq n^0(m') \tag{6.12}
\]

By our construction and previous lemmas,

\[
n^0(d) + n^0(m') = n^0(a) \geq n^0(b) = n^0(m'') \tag{6.13}
\]

The last two inequalities with the fact that \( n^0 \) is always non-negative imply:

\[
n^0(d) \geq n^0(m') - n^0(m'') \quad \text{and} \quad n^0(d) \geq n^0(m'') - n^0(m') \tag{6.14}
\]

The last inequality clearly implies Condition (c).
Let us consider Condition (d). We prove a stronger condition – that \( d_{|d|} \preceq m'_1 \) and \( d_1 \preceq m''_1 \). The first inequality follows from Lemma 6.3.3(c). Next, consider the second inequality. By Lemma 6.3.2(e):

\[
a_1 + a_{|a|} \preceq b_1 + b_{|b|}
\]

By our construction,

\[
d_1 = \min(a_1, a_{|a|}) \text{ and } m''_1 = \min(b_1, b_{|b|})
\]

By Lemma 6.3.1, this establishes the second inequality, proving Condition (d).

The algorithm \( A \) has three parts: a prologue, an iterative process and an epilogue; each part is composed of stages whose depth is at most one. The prologue has two stages and it computes a 1-to-4 operator as per Lemma 6.3.5. Namely, it transforms the input word \( w \) into a 4-vector \( \langle d, m', m'', r \rangle \). By Lemma 6.3.5, \( \langle d, m', m'' \rangle \) have the \( \diamond \)-property and by Lemma 6.3.4, \( r \in 1^* \). We consider the word \( r \) as a ‘reservoir’ of ones that supplies words of \( 1^* \) on demand.

After the prologue \( A \) performs an iterative process whose invariant is the \( \diamond \)-property; each iteration has a single stage, of depth one, that works as follows. It receives a 3-vector \( \langle d, m', m'' \rangle \), having the \( \diamond \)-property, and a reservoir \( r \) of all 1’s. It extracts, with no comparisons, a word \( \bar{r} \) (of a certain width) out of \( r \) and combines it with \( \langle d, m', m'' \rangle \) into a new 3-vector \( \langle \bar{d}, \bar{m}', \bar{m}'' \rangle \) having the \( \diamond \)-property. Moreover, \( |\bar{d}| = |d|/2 \). The 3-vector \( \langle \bar{d}, \bar{m}', \bar{m}'' \rangle \) serves as \( \langle d, m', m'' \rangle \) for the next iteration. Since \( r \in 1^* \), it does not matter which of the keys of \( r \) are extracted in each iteration but, for definiteness, let us decide that these are the first elements. These iterations use the operator \( \text{MaxMin} \triangleq (\ominus \circ \text{MinMax}) \).

The above combination and the fact that the resulting 3-vector has the \( \diamond \)-property is stated in the following lemma.

**Lemma 6.3.6.** Let the 3-vector \( \langle d, m', m'' \rangle \) have the \( \diamond \)-property and let \( \bar{r} = 1^{|d|/2} \). Let \( \langle \bar{d}, \bar{d} \rangle = \text{LS}(d), \langle \bar{m}', \bar{m}'' \rangle = \text{MaxMin}(\langle m', m'' \rangle), \bar{m}' = \bar{d} \cdot \bar{m}' \) and \( \bar{m}'' = \bar{m}'' \cdot \bar{r} \). Then \( \langle \bar{d}, \bar{m}', \bar{m}'' \rangle \) have the \( \diamond \)-property.

**Proof:** We verify the four conditions of Definition 6.3.1, as follows.

Condition (a) holds by Lemma 6.3.2(b).

Consider Condition (b). By our construction, the size of \( \bar{r} \) was selected so that \( |\bar{m}'| = |\bar{m}''| \). It remains to show that \( \bar{m}' \) and \( \bar{m}'' \) are sorted. By Lemma 6.3.3(a), \( \bar{d} \) is sorted. Clearly, \( \bar{m}' \) and \( \bar{m}'' \) are sorted. Recall that in a sorted word the smallest keys are on the left side and the higher keys are on the right side. The fact that \( \bar{m}'' \) is sorted follows from the fact that \( \bar{m}'' \) is sorted and that \( \bar{r} \in 1^* \). It remains to show that \( \bar{m}' \) is sorted. Namely, that \( d_{|d|} \preceq \bar{m}'_1 \).
By Condition (d) w.r.t. \( (d, m', m'') \):
\[
d_1 + d_{|d|} \preceq m_1' + m_1''
\]
By our construction,
\[
\tilde{d}_{|d|} = \max(d_1, d_{|d|}) \text{ and } \tilde{m}_1' = \max(m_1', m_1'')
\]
By Lemma 6.3.1, \( \tilde{d}_{|d|} \preceq \tilde{m}_1' \) implying Condition (b).
Consider Condition (c). The \( \Diamond \) -property of \( (d, m', m'') \) imply that \( |n^0(m'') - n^0(m')| \leq n^0(d) \).
By our construction, \( n^0(\tilde{m}') = \min(n^0(m'), n^0(m'')) \) and \( n^0(\tilde{m}'') = \max(n^0(m'), n^0(m'')) \).
Hence,
\[
0 \leq n^0(\tilde{m}'') - n^0(\tilde{m}') \leq n^0(d) \tag{6.15}
\]
Lemma 6.3.2(e), w.r.t. the splitting of \( d \) and the fact that \( d \) and \( (\tilde{d}, \tilde{d}) \) are isomorphic imply that:
\[
-n^0(\tilde{d}) \leq -n^0(\tilde{d}) = -n^0(d) + n^0(\tilde{d}) \tag{6.16}
\]
Adding Inequalities (6.15) and (6.16) yields:
\[
-n^0(\tilde{d}) \leq n^0(\tilde{m}'') - (n^0(\tilde{m}') + n^0(\tilde{d})) \leq n^0(d)
\]
Again by construction, \( n^0(\tilde{m}') = n^0(\tilde{m}'') \) and \( n^0(\tilde{m}') + n^0(\tilde{d}) = n^0(\tilde{m}') \); therefore:
\[
-n^0(\tilde{d}) \leq n^0(\tilde{m}'') - n^0(\tilde{m}') \leq n^0(\tilde{d}),
\]
and Condition (c) holds.
Let us consider Condition (d). We prove a stronger condition – that \( \tilde{d}_{|d|} \preceq \tilde{m}_1' \) and \( \tilde{d}_1 \preceq \tilde{m}_1'' \). The first inequality, \( \tilde{d}_{|d|} \preceq \tilde{m}_1' \), is established by considering two cases as follows.
Assume first that \( |\tilde{d}| > 0 \). In this case, \( \tilde{m}_1' = \tilde{d}_1 \). By Lemma 6.3.3(c), \( \tilde{d}_{|d|} \preceq \tilde{d}_1 = \tilde{m}_1' \).
Assume next that \( |\tilde{d}| = 0 \). In this case, \( \tilde{m}_1' = \max(m_1', m_1'') \) and \( \tilde{d}_{|d|} = \max(d_1, d_{|d|}) \).
Condition (d) w.r.t. \( (d, m', m'') \) states that \( d_1 + d_{|d|} \preceq m_1' + m_1'' \). By Lemma 6.3.1:
\[
\tilde{d}_{|d|} = \max(d_1, d_{|d|}) \preceq \max(m_1', m_1'') = \tilde{m}_1'
\]
Next, consider the second inequality; namely, that \( \tilde{d}_1 \preceq \tilde{m}_1'' \). As said, since Condition (d) holds w.r.t. \( (d, m', m'') \):
\[
d_1 + d_{|d|} \preceq m_1' + m_1''
\]
By our construction and Lemma 6.3.1:
\[
\tilde{d}_1 = \min(d_1, d_{|d|}) \preceq \min(m_1', m_1'') = \tilde{m}_1''
\]
Throughout the iterative process, the size difference between \( d \) and \( r \) remains constant.
In fact, \( |d| - |r| = 0 \) when the total number of keys is even and \( |d| - |r| = 1 \) when this number is odd. This iterative process terminates when the width of the new \( d \) is one.
Recall that we assume that the input of $A$ is 1-heavy and Bitonic. At the end of the iterative process, the keys are not completely sorted. Clearly, the single key of $\bar{d}$ is minimal; furthermore, the remaining key in the reservoir (if there is such a key) is 1. However, there are pairs of keys, $k' \in \bar{m}'$ and $k'' \in \bar{m}''$ such that the relative order between $k'$ and $k''$ is not determined. By Condition (c) of Definition 6.3.1, $|n^0(m') - n^0(m'')| \leq 1$. This enables the last part of $A$ – the epilogue – to merge the two sorted words $\bar{m}'$ and $\bar{m}''$ in a single stage by performing the $\text{MaxMin}$ operator on $\langle \bar{m}', \bar{m}'' \rangle$. After the epilogue, the resulting vector is actually sorted. Although it is a 4-vector rather than a single word, it can be transformed into a single sorted word by a predefined operator of depth 0.

The algorithm $A$ naturally leads to a network, denoted $A(n)$, where $n$ is the network’s width. It remains to consider the depth of $A(n)$. Let us count the number of stages of this network. The prologue has two stages. At the end of the prologue $|d| = \lceil n/4 \rceil$. The algorithm $A$ then performs an iterative process. At the end of the prologue $|d| = \lceil n/4 \rceil$. The iteration terminates when $|d| = 1$. By straightforward arithmetics, the number of iterations is $\lceil \log(n) \rceil - 2$. This is followed by the single stage of the epilogue. All together there are $\lceil \log(n) \rceil + 1$ stages, each of them is of depth at most one. Moreover, when $n > 2$, the depth of each stage is exactly one. This does not imply that the depth of $A(n)$ equals to the number of its stages even when $n > 2$. It only implies that the depth of $A(n)$ is at most the number of its stages, as stated in the following lemma.
Lemma 6.3.7. The network $A(n)$ sorts every 1-heavy Bitonic word of width $n$ and its depth is at most $\lceil \log(n) \rceil + 1$.

6.3.2 Sorting words of even width

We next present the Algorithm $B$ which is a variant of $A$ that sorts every Bitonic word of even width. The algorithm $B$ is very similar to the Algorithm $A$. It has the same three parts and the same number of stages in each part like $A$. The only difference between $A$ and $B$ lies in the iterative process. Moreover, this difference relates only to the handling of the reservoir (the word $r$). Recall that we assume that input is 1-heavy. Under this assumption, by Lemma 6.3.4, the reservoir contains only 1’s. In each iteration, the algorithm $A$ extracts, without any computation, a word $\bar{r}$ from the reservoir. In contrast to $A$, the algorithm $B$ extracts a word $\bar{r}$ (of the same width) from the reservoir via a certain processing. On the face of it, this processing of the reservoir is useless, since all the keys of the reservoir are 1. However, due to this processing, the algorithm $B$ is weakly symmetric and allows us to use Theorem 25. This symmetry is achieved as follows. In each iteration, $B$ splits the word $r$ by $\langle \bar{r}, \bar{r} \rangle = SL(r); \bar{r}$ serves as the new $r$ for the next iteration and $\bar{r}$ is used in the same manner as in $A$. Hence, $B$ processes the words $d$ and $r$ in a dual manner. By Lemmas 6.2.1 and 6.2.2(e), in each iteration, $B$ calculates a symmetric operator. By Lemma 6.2.2(a), the entire algorithm $B$ computes a symmetric operator. The algorithm $B$ naturally leads to a network. We denote this network of width $2k$ by $B(2k)$. The above discussion is summarized in the following lemma.

Lemma 6.3.8. The network $B(2k)$ sorts every 1-heavy Bitonic word of width $2k$, its depth is at most $\lceil \log(2k) \rceil + 1$ and it is weakly symmetric.

Note that the construction of $B$ is possible only when the total number of keys is even. Otherwise, $d$ and $r$ of an iteration are not of the same width; hence, they can not be processed in a dual manner.

The main result of this section is the following lemma.

Lemma 6.3.9. The network $B(2k)$ sorts every Bitonic word of width $2k$; its depth is $\lceil \log(2k) \rceil + 1$ when $k > 1$ and 1 when $k = 1$.

Proof. By Theorem 25 and Lemma 6.3.8, $B$ is a Bitonic sorter. It remains to establish the appropriate lower bound on the depth of $B(2k)$. When $k = 1$, all the stages but the first are of depth zero; hence, the depth of $B(2)$ is one.

When $k > 1$, we consider two cases according to whether $k$ is or is not a power of two. First assume that $k$ is not a power of two. By [18], the depth of any Bitonic sorter of width $n$, when $n$ is not a power of two, is at least $\lceil \log(n) \rceil + 1$. 

Technion - Computer Science Department - Ph.D. Thesis PHD-2011-10 - 2011
Next assume that \( k \) is a power of two. By [16] (Chapter 14), there is a unique Bitonic sorter of width \( 2k \) and of minimal depth. This Bitonic sorter is the Batcher’s Bitonic sorter and, if \( k > 1 \), it is not isomorphic to \( B(2k) \) due to the following reason. Let \( w \) be an input vector to these two networks. In the Batcher’s Bitonic sorter, of width \( 2k \), the input keys \( w_1 \) and \( w_{k+1} \) enter the same comparator. In contrast, the keys \( w_1 \) and \( w_{2k} \) enter the same comparator in \( B(2k) \).

6.4 Graphical representation of Networks.

This section provides another presentation of \( B(2k) \) which is based on an additional contribution of our work – a compact graphical representation of networks, as demonstrated in Figure 6.5. The same technique is later used to present the algorithm \( C \) which is a variant of \( B \) for odd number of keys.

In the previous section, the network \( B(2k) \) was divided into three parts: the prologue, the iterative process, and the epilogue. The prologue contains two stages, the iterative process contains \( \lceil \log(2k) \rceil - 2 \) stages and the epilogue contains a single stage. However, in this section, we prefer to present the same algorithm and the same network in a more unified manner. It comes out that the second stage of the prologue and the single stage of the epilogue can be added to the iteration part. Namely, in this section, we consider \( B \) to be of the form \( \beta \circ \beta \circ \cdots \circ \beta \circ \alpha \) where \( \alpha \) is a 1-to-4 operator of depth one and \( \beta \) is a 4-to-4 operator of depth one. The \( \beta \) operator is performed \( \lceil \log(n) \rceil \) times, where \( n \) is the width of the input word \( w \).

Figure 6.5 describes the Algorithm \( B \) according to the new scheme. In this figure ellipses represent operators and edges represent words. These operators receive and produce vectors – sequences of words. In our figures, the order of the words within such a vector is from left to right. The width of a word associated with an edge is usually written next to the edge in question.

The first stage, \( \alpha \), uses two new operators, \( \Phi \) and \( \mathcal{I} \mathcal{D} \), both of depth zero. The operator \( \Phi \) is the only 0-to-1 operator. This operator receives a vector with no words and produces a word with no keys. The operator \( \beta \) is the total operator defined by \( \mathcal{I} \mathcal{D}(V) \equiv V \).

Recall that the ‘+’ transformation is useful only for \( j \)-to-\( k \) operators. To this end, for an operator \( \pi \) and for \( k \in \mathbb{N} \), let \( \pi \rceil_k \) denote the restriction of \( \pi \) to \( \{0,1\}^k \); namely, \( \pi \rceil_k \equiv \pi \rceil_{\{0,1\}^k} \).

Using our terminology, \( \alpha \) is the 1-to-4 operator defined by \( \alpha \equiv (\mathcal{I} \mathcal{D} \lceil_1 + \Phi + \Phi + \mathcal{I} \mathcal{D} \lceil_1 ) \circ \mathcal{L} \mathcal{S} \). Note that \( \mathcal{L} \mathcal{S} \) is of depth one and \( (\mathcal{I} \mathcal{D} \lceil_1 + \Phi + \Phi + \mathcal{I} \mathcal{D} \lceil_1 ) \) is of depth zero; hence, \( \alpha \) is of depth one.

The operator \( \beta \) is the 4-to-4 operator defined by \( \beta \equiv (\mathcal{I} \mathcal{D} \lceil_1 + \mathcal{C}_2 + \mathcal{C}_2 + \mathcal{I} \mathcal{D} \lceil_1 ) \circ (\mathcal{L} \mathcal{S} + \mathcal{M}ax.\mathcal{M}in + \mathcal{S} \mathcal{L} \). Again, \( \beta \) is of depth one. As said, \( \beta \) is iterated \( \lceil \log(n) \rceil \) times.
Figure 6.5: Graphical representation of the networks $B(2k)$ and $C(2k - 1)$
The first and last iterations of $\beta$ are degenerated in the following senses. As evident in Figure 6.5, the MaxMin operator of the first iteration of $\beta$ processes empty words; therefore, it requires no comparators. In the last iteration of $\beta$, each of the operators LS and SL receives a single key and, again, requires no comparators. Due to this, these stages were not considered as part of the iteration in the previous section. Another reason in this regard is the fact that the reservoir (of 1’s) does not exist at the beginning of the first iteration.

It is worthwhile to consider when the iteration ends. In the last iteration of $\beta$ the words transferred from the LS operator and from the SL operator to the $C \upto 2$ operators are empty. Moreover, at this point, the resulting 4-vector $\langle d, m', m'', r \rangle$ is always a fixpoint of $\beta$; hence, adding more iterations of $\beta$ will not change the output of the algorithm. When $n > 2$, all these iterations are necessary, as addressed in the proof of Lemma 6.3.9. However, when $n \leq 2$, the operator $\beta$, in this context, is of depth zero and is clearly not necessary.

As discussed in Section 6.2, two distinct (non-isomorphic) networks may compute the same operator. This holds even when the two networks are restricted to be of minimal depth. However, an operator of depth one (or zero) has a unique network of minimal depth that computes it. Since the depth of each of the operators depicted in Figure 6.5 (by ellipses) is at most one, this figure actually specifies, for any integer $k$, a unique network—the network that implements each operator by a minimal depth sub-network. As said, we denote this unique network by $B(2k)$.

By Lemma 6.2.1, the operators MaxMin is symmetric. Clearly, $C \upto 2$ and $\mathcal{I}D \upto 1$ are symmetric. As said, LS and SL are the dual of each other. By Lemma 6.2.2, each of the operators $\alpha$ and $\beta$ are symmetric. Again, by Lemma 6.2.2, the network $B(2k)$ is weakly symmetric. In fact, the network $B(2k)$ is strongly symmetric; however, we do not rely on the latter property.

We next describe the algorithms $C$ which is a variant of $B$ that sorts Bitonic words of odd width. The algorithms $B$ and $C$ are depicted, side by side, in Figure 6.5. There are only minor differences between $B$ and $C$, as follows:

- The two operators, LS and SL, sometimes replace each other.
- The two functions, ‘$\lceil \cdot \rceil$’ and ‘$\lfloor \cdot \rfloor$’, sometimes replace each other.
- The width of two corresponding words may differ by at most one.

Again, $C$ is of the form $\beta \circ \beta \circ \cdots \circ \beta \circ \alpha$ where $\alpha$ is a 1-to-4 operator and $\beta$ is a 4-to-4. Again, $\beta$ is performed $\lceil \log(n) \rceil$ times.

In the case of $C$:

$$\alpha = (\mathcal{I}D \upto 1 + \Phi + \Phi + \mathcal{I}D \upto 1) \circ SL$$
$$\beta = (\mathcal{I}D \upto 1 + C \upto 2 + C \upto 2 + \mathcal{I}D \upto 1) \circ (SL + MaxMin + SL)$$

Again, these operators are of depth one. As said, the right hand part of Figure 6.5 can be interpreted as a specific network. We refer to this network, of width $2k - 1$, by $C(2k - 1)$. 
Up to now, we partition our networks in a ‘vertical’ manner to stages. To further study these networks, we also use a ‘horizontal’ partitioning. As depicted in Figure 6.5, the iterative part of each network is divided into three sub-networks: $D$ (Down), $M$ (Middle) and $U$ (Up). This naming relates to the fact that the algorithm tries to keep the smaller keys in $D$ and the larger keys in $U$.

We denote by $D_B(2^k)$ the sub-network $D$ of the network $B(2^k)$. Similarly, we denote the sub-networks $M$ and $U$ of the networks $B$ and $C$. Using these notations, it is not hard to see that:

$$M_B(2^k) = M_C(2^k-1) \quad (6.17)$$

$$U_B(2^k) = U_C(2^k-1) \quad (6.18)$$

Within each stage, the subnetwork $U_B(2^k)$ and $D_C(2^k-1)$ compute dual operators \(^{(6.19)}\)

The next section proves that $C$ sorts every Bitonic word of odd width. Surprisingly, in our proves, the behavior of the sub-network $U$ in $B(2k)$ and $C(2k-1)$ is never studied. Instead, our proves rely on the above equalities 6.17, 6.18 and 6.19.

### 6.5 Bitonic sorters of odd width

This section proves that, for every $k$, $C(2k-1)$ is a Bitonic sorter. The proof is indirect and builds on the similarity of $C(2k-1)$ to $B(2k)$ and on the fact that $B(2k)$ is a Bitonic sorter. In both networks, $B(2k)$ and $C(2k-1)$, it is easy to verify that the key (if there is any) coming out of $D$ or $U$ is a minimal key or a maximal key, respectively. Hence, the core of our proof is the claim that the output of $M$ is sorted.

To this end, we show that, for every Bitonic (not necessarily 1-heavy) word $w$ of width $2k-1$, there is a Bitonic word $w'$ of width $2k$ with the following property. When $C(2k-1)$ processes $w$ and $B(2k)$ processes $w'$, the sub-network $M_{C(2k-1)}$ and $M_{B(2k)}$ receive the same vector. By Equality 6.17, $M_{B(2k)} = M_{C(2k-1)}$; hence, these two sub-networks produce the same vector. Since $B(2k)$ is a Bitonic sorter, the output of $M_{B(2k)}$ is sorted. Hence, the same holds for $M_{C(2k-1)}$.

To construct the above $w'$, let $t : \{0, 1\}^* \to \{0, 1\}^*$ be defined as follows. The word $t(w)$ is derived from $w$ by inserting a single 0 bit. This bit is inserted in the longest interval of 0’s in $w$. If $w$ has several intervals of 0’s of maximal length, the bit is inserted in the last interval. For example: $t(00010010) = 000100100$, $t(0011100) = 00111000$, $t(11) = 110$.

For a word $w$, let $H^0(w)$ denote the number of 0’s at the head of $w$. Formally, $H^0(w)$ is the largest $i$ such that $0^i$ is a prefix of $w$. Similarly, let $H^1(w)$ be the largest $i$ such that $1^i$ is a prefix of $w$. 
Lemma 6.5.1. Let \( w \) be a Bitonic word\(^4\) and let \( \langle x, y \rangle = SC(w) \). Then either \( LS(t(w)) = \langle t(x), y \rangle \) or \( LS(t(w)) = \langle \neg(t(x)), y \rangle \).

Proof. Let \( w' = t(w) \) and let \( \langle x', y' \rangle = LS(w') \). We need to show that \( y' = y \) and that \( x' = t(x) \) or \( x' = \neg(t(x)) \). Clearly, \( |w'| = |w| + 1 \) and \( n^0(w') = n^0(w) + 1 \). By straightforward arithmetics \( |x'| = |x| + 1 \) and \( |y'| = |y| \). We consider the following cases which are conclusive but not disjoint:

Case 1: \( w \) is empty. This case is trivial.

Case 2: \( w = 1^i0^{j+1}k \) for some \( i, j, k \geq 0 \).

By Lemma 6.3.3, \( y \) and \( y' \) are ascending. Clearly:

\[
n^1(y') = \min(\max(i, k), |y|) = n^1(y)
\]

Therefore, \( y = y' \).

The facts that \( LS \) and \( SC \) are isomeric and that \( y = y' \) imply that \( n^1(x') = n^1(x) \) and \( n^0(x') = n^0(x) + 1 \). By Lemma 6.3.3, both \( x \) and \( x' \) are descending-ascending. Hence, it remains to show that \( H^1(x) = H^1(x') \). In fact, both these numbers are \( \min(i, k) \).

Case 3: \( w = 0^i1^{j+1}k \) for some \( i, j, k \geq 0 \).

We first consider the words \( x \) and \( x' \). By duality and Lemma 6.3.3, these words are ascending-descending. Clearly:

\[
H^0(x) + 1 = \min(\max(i, k), |x|) + 1 = \min(\max(i, k) + 1, |x'|) = H^0(x')
\]

Therefore, \( x' = t(x) \) or \( x' = \neg(t(x)) \). (Note that the latter happens when \( x \in 1^* \); in the current case it implies that \( w \in 1^* \).)

Next, consider the words \( y \) and \( y' \). By previous arguments, \( y \) and \( y' \) are isomeric. By Lemma 6.3.2(c), \( y \) and \( y' \) are ascending-descending. It remains to show that \( H^0(\neg(y)) = H^0(\neg(y')) \). In fact, both these numbers are \( \min(i, k) \).

Lemma 6.5.2. Let \( w \) be a Bitonic word of width \( 2k - 1 \). Assume that \( w \) is fed into \( C(2k - 1) \) and that \( t(w) \) is fed into \( B(2k) \). Then the two sub-networks \( M_{C(2k - 1)} \) and \( M_{B(2k)} \) (which are equal) receive the same vector.

Note that the word \( w \) of Lemma 6.5.2 is not necessarily 1-heavy.

Proof. Refer to Figure 6.5. The sub-network \( \mathcal{M} \) receives one vector from \( \mathcal{D} \) and one vector from \( \mathcal{U} \). Consider the latter. By Lemma 6.5.1, \( \mathcal{U}_{B(2k)} \) and \( \mathcal{U}_{C(2k - 1)} \) receive the same word. By Equality 6.18, \( \mathcal{U}_{C(2k - 1)} = \mathcal{U}_{B(2k)} \); hence, they produce the same vector.

\(^4\)Note that \( w \) is not necessarily 1-heavy.
\(^5\)\( LS(t(w)) \neq \langle t(x), y \rangle \) only when \( w \in 1^* \).
Next, consider the words that are transferred from $\mathcal{D}$ to $\mathcal{M}$. As shown in Figure 6.5, in each $\beta$ iteration $C(2k-1)$ performs the operator $SL$ on its $d$ while $B(2k)$ performs the operator $LS$ on its $d$. Let $x = \langle x_1, x_2 \ldots x_{\lceil \log(2k-1) \rceil} \rangle$ where $x_i$ is the word that enters the operator $SL$ in the $i'$th iteration of $\beta$ in the network $C(2k-1)$. Let $x' = \langle x'_1, x'_2 \ldots x'_{\lceil \log(2k) \rceil} \rangle$ where $x'_i$ is the word that enters the operator $LS$ in the $i'$th iteration of $\beta$ in the network $B(2k)$. Clearly, $LS(q) = \langle t(q) \rangle$, for every $q \in \{0, 1\}^*$. Hence, by straightforward induction and Lemma 6.5.1, $x'_i = t(x_i)$ or $x'_i = \langle t(x_i) \rangle$, for every $i$. Again by Lemma 6.5.1, the words that are transferred, under $w$, from $\mathcal{D}_{C(2k-1)}$ to $\mathcal{M}_{C(2k-1)}$ are identical to the words transferred, under $w'$, from $\mathcal{D}_{B(2k)}$ to $\mathcal{M}_{B(2k)}$.

The above discussion is summarized in the following lemma.

**Lemma 6.5.3.** For every $k$, the network $C(2k - 1)$ is a Bitonic sorter.

Let us summarize the main result of this paper. By Lemmas 6.3.9 and 6.5.3, for every $n$, there is a Bitonic sorters of width $n$ and of depth $\lceil \log(n) \rceil + 1$. By Levy and Litman [18], the depth of such a Bitonic sorter, when $n$ is not a power of two, is at least $\lceil \log(n) \rceil + 1$. Due to Batcher’s construction [1], $\log(n)$ is the minimal depth of a Bitonic sorter of $n$ keys, when $n$ is a power of two. This implies the main result of this paper:

**Theorem 1.** The minimal depth of a Bitonic sorter of $n$ keys is $2 \lceil \log(n) \rceil - \lfloor \log(n) \rfloor$.

**Thanks**

We would like the thank the anonymous reviewer for his useful comments.
Chapter 7

Conclusion

This chapter summarizes the work done in this research. It comes out that this study is not focused on a specific problem. Instead, it is more of a walk through the field of comparator networks. Hence, the papers in this work are presented chronologically by the progress of the research. This conclusion, on the other hand, is provided in a subject based manner. It starts with the more general - computability and complexity of functions under the comparator networks model and the min-max model. It then looks into more specific functionalities studied here.

7.1 The min-max model

The min-max model is a model of computation, similar - but not equivalent to the acceptable comparator networks model. The main difference between them is a certain fanout restriction which exists only in the comparator model. This difference makes the min-max model to be somewhat ‘stronger’ then the comparator model. It is stronger both in aspects of computability (some functions can be computed only in the min-max model) and complexity (some functions can be computed faster in the min-max model).

The min-max model is known and used for a long time. Knuth [14] (1973), demonstrated that insertion of a single key into a sorted sequence can be performed faster in the min-max model then the comparator model.
In this work, we study the min-max model by considering several 0-1 -like principles. It comes out that the min-max model is the ‘strongest’ model of computation which obeys our principles. That is, if a function is computable in a model of computation in which any of these principles holds, this function can be computed by a min-max network. Therefore, we find the min-max model to be more natural then the comparator model for solving key arrangement problems. This subject is addressed in Chapter 4. This work shows that in some cases sorting Bitonic sequences can also be performed faster by a min-max model then by a comparator network. However, the depth difference is at most one.

Some functions are computable only in the min-max model and not in the comparator networks model. This fact was also demonstrated by Knuth [14]. We strengthen this result by providing an isomeric mapping that is computable by a min-max network and not by a comparator network.

7.2 Conclusive sets

This work presents several tools for analysis of comparator networks and min-max networks. The 0-1 principle, introduced by Knuth [14], can be viewed as such. It states that a comparator network is a sorting network if and only if it sorts all binary inputs. In our terminology, it states that the set of binary vectors are a conclusive set for sorting. Clearly, sorting is not the only functionality that comparator networks are useful for nor the 0-1 Principle. Additional functionalities may include merging, separation, sorting restricted sets of vectors, etc. For each of these functionalities, some variant or another of the 0-1 principle [14] was used for proving the correctness of the networks in question and always using binary vectors.

This work investigates non-binary conclusive sets and proposes for the above functionalities conclusive sets of minimal size. That is, we state the minimal size of a conclusive set (which is dependent of the functionality in question and of the number of keys to be processed) and propose a conclusive set of that size. Our lower bounds on sizes of conclusive sets rely on a certain construction, for every non-sorted binary sequence \( v \), a comparator network that sorts all binary vectors besides \( v \).

Conclusive sets are studied in Chapter 2 and this construction is depicted in Figure 2.2. Still, conclusive sets are used extensively throughout this work.
7.3 Symmetry of comparator networks

This work studies the concept of symmetry of comparator networks. We distinguish between two types of such a symmetry. The first type, called strong symmetry, was studied in [29] and it concerns the structure of a comparator network. We propose a second type of symmetry, called weak symmetry and is more of a “black-box” approach; that is, this symmetry concerns only the input-to-output mapping of the network in question. In this work we use only the weak symmetry. Formality and details of these symmetries is discussed in Section 6.2.

The weak symmetry allows us to extend the well-known 0-1 Principle [14] as in Theorem 22. Replacing the term ‘weakly’ with ‘strongly’ dramatically reduces the power of Theorem 22. For example, comparator networks of odd width can not be strongly symmetric but can be weakly symmetric as are all sorting networks.

7.4 Merging networks

This work presents merging networks (comparator networks that merge two equal length sorted sequences) in which several of the outputs are accelerated. That is, these outputs are generated much faster than the other outputs, and this without hindering the other outputs. Namely, for every \(0 < k \leq n\), we present a merging network of minimal depth that merges two sorted sequences of length \(n\) into a single sorted sequence. This merging network produces either the lowest \(k\) keys or the highest \(k\) keys\(^1\) after a delay of \(\lceil \log(k) \rceil + 1\) comparators.

This acceleration is based on a new merging technique, the Tri-section technique, that separates, by a depth one network, two sorted sequences into three sets, such that every key in one set is smaller or equal to any key in the following set. After this separation, each of these sets can be sorted separately and this leads to the desired acceleration. The idea of separating the input into two sets is known and is used, for example, in the Bitonic sorter of Batcher [1]; however, to the best of our knowledge, separation into three sets as above is novel.

Building on that, we construct, for every \(0 < k < n\), an \(n\)-key sorting network that accelerates its \(k\) lowest or its \(k\) highest outputs. Namely, its depth is \(\frac{\log(n) \cdot \lceil \log(2n) \rceil}{2}\), the same depth as the Batcher sorting networks [1]. However, in contrast to the Batcher merge-sort networks which may accelerate only the first and last outputs, our merge-sort networks accelerates either the \(k\) lowest keys or \(k\) highest keys\(^1\) to a delay of less than \(\lceil \log(n) \rceil \cdot \lceil \log 2k \rceil\) comparators.

\(^1\)When \(n\) is a power of two, both the lowest \(k\) keys and the highest \(k\) keys can be accelerated.
Another contribution of this work concerns Batcher’s merging techniques. The family of Batcher merging networks is defined in Chapter 3. It was shown in [16] that all published merging networks, whose width is a power of two, are members of this family. All these merging networks are of minimal depth and have no degenerate comparators. (A degenerate comparator has a fixed incoming edge whose value is always greater or equal to the value on the other incoming edge, for every valid input of the network.) The above fact arise the following question : Are the Batcher merging networks the only merging networks with the following properties:

1. Their width, $2n$, is a power of two.
2. Their depth is minimal – $\log(2n)$.
3. They have no degenerate comparators.

The Tri-section technique provides a negative answer to this question.

### 7.5 Bitonic sorters

Chapter 6 consider Bitonic sorters - comparator networks that sort Bitonic sequences of a certain width. Building on previous works, it establishes that:

The depth of an $n$-keys Bitonic sorter is at least $2 \lceil \log(n) \rceil - \lfloor \log(n) \rfloor$.

It then layout, for every $n$, an $n$-keys Bitonic sorter of such a depth. This construction is summarized in Figure 6.5.

When $n$ is a power of two, $2 \lceil \log(n) \rceil - \lfloor \log(n) \rfloor = \log(n)$. The fact that, in this case, $\log(n)$ is the minimal depth is due to the seminal work of Batcher [1]. However, the minimal depth of Bitonic sorters, in the general case, was unknown.

As said, sorting Bitonic sequences (whose width is not a power of two) is an example of a 'natural' functionality for which the min-max model is faster then the comparator model. Let $T'(n)$ denote the minimal depth of a min-max network that sorts all Bitonic sequences of $n$ keys. The above construction clearly implies an upper bound on $T'(n)$. This and a simple reachability argument imply that

$$\lfloor \log(n) \rfloor \leq T'(n) \leq \lceil \log(n) \rceil + 1.$$
7.6 Further research

This work offers several questions that remained open for future research.

In the context of the Tri-section technique, one question, which remains open, concerns accelerating all the outputs of a merging network, each to a delay that is close to the trivial reachability bound of this output. This bound is due to the fact that the $j$ lowest (or highest) output may come from each of certain $2^j$ input edges. Namely, our question is:

\textit{Question 26}. For any $n$ (or arbitrary large $n$), is there a merging network of width $2n$ that, for every $j < n$, accelerates the $j$ lowest output and the $j$ highest output to a delay of $\log(j) + o(\log(j))$?

In the context of conclusive sets, other conclusive sets for these functionalities and for other functionalities may be of interest.

In the context of the min-max model the minimal depth of a min-max network that sorts Bitonic sequences remains open. Furthermore, we are interested in other ‘natural’ functionalities in which the min-max model is faster than the comparator networks model.
Summary

In this work we studied several aspects of comparator networks. The main focus of this work is on the fields of Bitonic sorters and merging networks; however, some of the tools we developed during this work are applicable to other comparator networks.

The main results presented in this work are:

1. Bitonic Sorters of minimal depth – When the number of keys to be sorted is not a power of two, minimal depth Bitonic sorters were already known for many years. However, in the general case, the minimal depth of Bitonic sorters was not known. This work establishes a lower bound on this depth and presents Bitonic sorters of this depth.

2. The MinMax model of computation – The comparator networks model of computation is widely used and studied. In this model keys propagate through the network (i.e., the comparators of the network) without being duplicated or dropped. This work studies a similar model model of computation, called the MinMax model, where this restriction is lifted. It was already known that the MinMax model is stronger then the comparator model both in the way of computability (functions that are computable only in the MinMax model and not in the comparator model) and in the way of speed (functions that can be computed faster in te MinMax model). This work shows that the MinMax model is the strongest model of computation in which several principles (0-1 like principles) are maintained.

3. Accelerated comparator networks – These are comparator networks in which several of the outputs are accelerated. That is, some outputs are generated much faster than the other outputs, and this without hindering the other outputs. Namely, we present merging networks that produce either the lowest $k$ keys or the highest $k$ keys after a delay of $\lceil \log(k) \rceil + 1$ comparators. Building on that, we construct, for every $0 < k < n$, an $n$-key sorting network that accelerates its $k$ lowest or its $k$ highest outputs. This sorting network is a merge-sort network (it follows the wellknown merge-sort algorithm) and has a minimal depth among these networks. Namely, its depth equals the depth of Batcher’s merge-sort networks [1]. However, in contrast to the Batcher merge-sort networks which may accelerate only the first and last outputs, our merge-sort networks accelerates either the $k$ lowest keys or $k$ highest keys to a delay of less than $\lceil \log(n) \rceil \cdot \lceil \log 2k \rceil$ comparators.
This acceleration is based on a new merging technique, the *Tri-section technique*, that separates, by a depth one network, two sorted sequences into three sets, such that every key in one set is smaller or equal to any key in the following set. After this separation, each of these sets can be sorted separately and this leads to the desired acceleration. The idea of separating the input into two sets is known and is used, for example, in the Bitonic sorter of Batcher [1]; however, to the best of our knowledge, separation into three sets as above is novel.

4. Conclusive sets for comparator networks – handy tools for analysis of comparator networks. These are sets of (not necessarily binary) vectors that verify a specific functionality. The 0-1 principle introduced by Knuth [14] states that a comparator network is a sorting network if and only if it sorts all binary inputs. Hence, it points out a certain binary conclusive set. This work investigates non-binary conclusive sets and proposes for the functionalities of sorting, merging, halving and Bitonic sorting, conclusive sets of minimal size. These tools are used through this work to easy up the analysis and proves of comparator networks.
Bibliography


Acknowledgements

First of all, I would like to thank my advisor, who was the engine and steering wheel of this research. Thank you Ami.

I would also like to thank my old family and my new one and finally to Tigran who stood by me when times were hard.
**Summary**

The work presented here is part of a larger study on the organization of vectors and the analysis of sorting algorithms. The focus is on the Trisection Merging Technique, which is a method for dividing and merging vectors to achieve optimal sorting.

Shmilovitz and Shtokumin proposed an algorithm for dividing vectors into subsets of length $k$, where $k = 70$. The algorithm aims to achieve an optimal sorting efficiency, taking into account the nature of the data and the specific requirements of the operation.

The Trisection Merging Technique is the basis for the proposed algorithm, which is designed to efficiently sort vectors of length $k$. The technique involves dividing the input into three parts, merging them, and repeating the process until the entire vector is sorted.

The work also considers the parallel processing aspects of the algorithm, highlighting its potential for efficient computation on modern parallel architectures. The implementation of the algorithm on parallel processors is also discussed, showing its adaptability to various computing environments.

The theoretical analysis and practical implementation of the algorithm provide insights into the effectiveness of the Trisection Merging Technique for sorting applications. The results suggest that the proposed method can achieve significant performance improvements over existing sorting algorithms, particularly in scenarios with large data volumes and high parallel processing capabilities.
The definition of the function MinMax is as follows:

1. If the input is a single element, return that element.
2. If the input is a set of elements, return a set containing the maximum.
3. If the input is a sequence of elements, return the maximum element.

As mentioned by Knuth [14], the MinMax function can be used to solve the following problem:

The problem of finding the maximum element in a sequence of integers can be solved using the MinMax function.

The MinMax function is defined as follows:

1. If the input is a single element, return that element.
2. If the input is a set of elements, return a set containing the maximum.
3. If the input is a sequence of elements, return the maximum element.

The function can be used to solve the following problem:

The problem of finding the maximum element in a sequence of integers can be solved using the MinMax function.

The MinMax function can be used to solve the following problem:

The problem of finding the maximum element in a sequence of integers can be solved using the MinMax function.

The MinMax function can be used to solve the following problem:

The problem of finding the maximum element in a sequence of integers can be solved using the MinMax function.

The MinMax function can be used to solve the following problem:

The problem of finding the maximum element in a sequence of integers can be solved using the MinMax function.

The MinMax function can be used to solve the following problem:

The problem of finding the maximum element in a sequence of integers can be solved using the MinMax function.
לброוח את מפעלי או את השורות של שלושBushortרוי, כusterity. בפשטות, על דוגמה קומול או בקשו שמות

המינימלי של מיון סדרות יטוניות של הפקחות הוואו. אם וג מציינו עקר כל z מימי.

בטרויאלי ותחתון, אנו קובעים על בונים עטש בש. 

טענה ששתהוקה של 2' נבתה בש. 2', העבדה ומקמריא אלו הזעויות המינימליים של

מימי ביניים, או העבדה הפריזת לקרר של לשעתה, בקסיים בובה ח. או לא ודוקa של 2. Batcher מהנוקט getters מפלט.

לזרב הנוקטertos ממינימלי כנלי לא 정도. נק núm ב-מג' ואת הנוקט getters מפלט משמשים שיפוריים ב

둘ר בוניום, בתו 5 פקחות. אם מצטרפים עשת את משיה ידוע דע נל על הפונקציה T. תוף התוקט טרויאלי

קנוע כיו

\[ \log(n) \leq T(n) \] (1)

העובדה של [1] Batcher קובעת כי

\[ T(2n) \leq T(n) + 1 \] (2)

אידוקציה. כי

\[ T(2^n) = j \] (3)


 Comcast יחי. או מיון ביניים יטוניות והרחבית (ודאימו) ממקל מיניים יטוניות. הדגמים הם ביניים

משתנים את מיון ביניים יטוניות והרחבית (ודאימו) ממקל מיניים יטוניות. הлибо

\[ T(i * j) \leq T(i) + T(j) \] (4)

טוכניקה הנוזחת (לפני בוזה) משל מיון ביניים יטוניות והרחבית (ודאימו) ממקל מיניים יטוניות. אנדרטת עם [22] Batcher & Liszka

\[ T(n) \leq \max \{ T(\lfloor n/2 \rfloor), T(\lfloor n/2 \rfloor) \} + 2 \] (5)

באמנצטים אנדרטב קביעה קבולה כי

\[ T(n) \leq 2 \log(n) - 1 \] (6)

כאמור, בוזה בצוואת את הסוס התוקט. בヵו שוח לא ודוקa של 2 פורסמה, "ז".

\[ T(n) \geq \log(n) \] (7)

ידכ מע-א-שויים של [3] z המ cocci T מחי לא פסונטי. כאמור בוזה הנקט יוכדת את העבר המדריך של T ימכיר

. \[ T(n) = 2 \log(n) - \log(n) \] (8)
עמל רשתות מיזוג

היתור על מחקר

לשם מחזור חלקי של הדרישות לקבלת תואר דוקטור לפילוסופיה

תמיר לֵּי

הוגש להנהלה הטכניון - מכון טכנולוגי לישראל

2011
علم رشته ماجد

팀ير لري