Bitonic sorters of minimal depth

Tamir Levi* Ami Litman†

May 3, 2010

Abstract

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

Keywords: Bitonic Sorting, Comparator Networks, Oblivious Algorithms, Symmetric Comparator Networks

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 1** 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 [7], these Bitonic sorters are of minimal depth, when $n$ is not a power of two.

An additional contribution of this paper is a certain concept of symmetry of comparator networks (and of oblivious algorithms). In a nutshell, a network is symmetric if it processes the smaller keys and the larger keys in a “similar manner”. This concept allows us to extend the well-known 0-1 Principle [3] 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). This work presents (in Section 3.1) a precise definition of ‘symmetric comparator network’ and establishes the following extension of the 0-1 Principle:

---

*Faculty of Computer Science, Technion, Haifa 32000, Israel. levyt@cs.technion.ac.il
†Faculty of Computer Science, Technion, Haifa 32000, Israel. litman@cs.technion.ac.il
Theorem 2 Let $S$ be a 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.

The 0-1 Principle has many variants related to other functionalities, for example, merging and Bitonic sorting. The same holds for Theorem 2 and several of its variants are presented in Section 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 5.

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 $n$,

$$\lceil \log(n) \rceil \leq T(n). \tag{1}$$

Due to the constructions of Batcher [1]:

$$T(2n) \leq T(n) + 1. \tag{2}$$

This and Inequality (1) imply that:

$$T(2^j) = j. \tag{3}$$

Nakatani et al. [9] established that:

$$T(i \cdot j) \leq T(i) + T(j). \tag{4}$$

The only prior technique that constructs Bitonic sorters of any width is due to Batcher and Liszka [8]. They show that:

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

This and a straightforward induction imply:

$$T(n) \leq 2 \lceil \log(n) \rceil - 1. \tag{5}$$
The first, non-trivial, lower bound on $T(n)$ is due to Levy and Litman [7]. They showed that for every $n$ that is not a power of two:

$$\lfloor \log(n) \rfloor + 1 \leq T(n).$$  \hfill (6)

This result, combined with Equality 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 $n$:

$$T(n) = 2 \lfloor \log(n) \rfloor - \lfloor \log(n) \rfloor .$$  \hfill (7)

In other words,

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

Another model of oblivious computation, called min-max networks, was studied by Levy and Litman [5]. (These networks are discussed in Section 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 1, the exact value of $T'(n)$ is almost known, as implied by the following arguments.

The same reachability argument imply Inequality (1) also for min-max networks; therefore:

$$\lfloor \log(n) \rfloor \leq T'(n)$$  \hfill (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 (9)

Inequalities (8,9) and Theorem 1 imply that for every $n$:

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

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 (3,8) and (9).
- $T'(n) = \lfloor \log(n) \rfloor + 1$, for every odd $n$. Levy and Litman [4] established that $\lfloor \log(n) \rfloor + 1 \leq T'(n)$, for every odd $n$. Inequality (9) and Theorem 1 provide the matching upper bound.
- $T'(n) = \lfloor \log(n) \rfloor$ for $n \in (10 \cdot 2^i)$. This was established in [4].
- $T'(n) = \lfloor \log(n) \rfloor$ for $n \in (6 \cdot 2^i)$, as shown in the next paragraph.

Figure 2 depicts a min-max network, presented in [5], 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 [9] and Batcher [1] are applicable also to min-max networks; hence, Inequalities (2) and 4 hold also for $T'$. Together with Inequality (8), we get that $T'(6 \cdot 2^i) = 3 + i = \lfloor \log(6 \cdot 2^i) \rfloor$.

As discussed in [4] and [7], 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 $n = 10 \cdot 2^i$. 

3
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 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 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].

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\(^1\) 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 1(a).

Another model of oblivious computation, the min-max model, was studied by Levy and Litman [5]. Figure 2 depicts a min-max network, presented in [5], 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. Levy and Litman [5] show that the min-max model is the strongest model that obeys certain variants of the

---

\(^1\)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.
Figure 2: A min-max network of width 6 and depth 3.

0-1 Principle. Due to this maximality, they suggest to define an oblivious algorithm as an algorithm that can be executed by a min-max network.

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 [3] 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, \(n^0(V) + n^1(V) = \sum_{i=1}^{\lfloor |V| / 2 \rfloor} |V_i|\).

2 Symmetric Oblivious Algorithms

This chapter studies symmetric oblivious algorithms. In a nutshell, an algorithm is called symmetric if it processes the smaller keys and the larger keys in a similar manner. This chapter provides a formal definition of the symmetry concept. The main results of the section are Theorems 8, 9 and 10. They state that a 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. This chapter consider both forms; it comes out 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 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 \( \| V \| \triangleq (|V_1|, |V_2|, \ldots, |V_{|V|}) \). 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 3** An operator is a partial mapping \( \alpha : \{0, 1\}^* \to \{0, 1\}^* \) 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 3, 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 \( \langle 7, 7 \rangle \) and its output signature is \( \langle 14 \rangle \). That is, every vector with signature \( \langle 7, 7 \rangle \) 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 \( \alpha|_s \), is the operator defined by:

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

An operator \( \alpha \) is computable (by networks) if, for every input signature \( s \) of \( \alpha \), there is a network that computes \( \alpha|_s \). 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 \( \alpha|_s \).
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, $C(V) \triangleq V_1 \cdot V_2 \cdot \ldots \cdot V_{|V|}$.

- **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 `$\circ$ ', `$\bullet$ ' and `$\leftarrow$ '.

  The first operator, `$\circ$ ', is defined by:
  \[
  \leftarrow ((V_1, V_2, \ldots, V_{|V|})) \triangleq (V_{|V|}, \ldots, V_2, V_1)
  \]

  The mapping `$\bullet$ ' is defined on words by:
  \[
  \leftarrow (w_1, w_2, \ldots, w_{|w|}) \triangleq (w_{|w|}, \ldots, w_2, w_1)
  \]

  The operator `$\bullet$ ' is defined on vectors by:
  \[
  \leftarrow ((V_1, V_2, \ldots, V_{|V|})) \triangleq (\leftarrow (V_1), \leftarrow (V_2), \ldots, \leftarrow (V_{|V|})
  \]

  The operator `$\leftarrow$ ' is defined by:
  \[
  \leftarrow (V) \triangleq \leftarrow (\leftarrow (V))
  \]

- **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
  \[
  \text{Min}(\langle s, r \rangle) \triangleq \langle \min(s_1, r_1), \min(s_2, r_2), \ldots, \min(s_{|s|}, r_{|r|}) \rangle
  \]
  \[
  \text{Max}(\langle s, r \rangle) \triangleq \langle \max(s_1, r_1), \max(s_2, r_2), \ldots, \max(s_{|s|}, r_{|r|}) \rangle
  \]

  The $\text{MinMax}$ operator is a 2-to-2 operator of depth 1 defined on 2-vectors $\langle s, r \rangle$ with $|s| = |r|$ by $\text{MinMax}(\langle s, r \rangle) \triangleq \langle \text{Min}(\langle s, r \rangle), \text{Max}(\langle s, r \rangle) \rangle$.

- **The Batcher operator**: This operator, denoted $B$, 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 $|w'| = |w''|$, $B$ is defined by $B(\langle w' \cdot w'' \rangle) \triangleq \text{MinMax}(\langle w', w'' \rangle)$. 


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 ‘$\circ$’ 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 ‘$\circ$’. 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:

$$C \circ (B + B + B + B) \circ (B + B) \circ B$$

When the operator $(B + B + B + B) \circ (B + B) \circ B$ 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 $C$ concatenates this vector into a single sorted word.

The duality of Boolean algebra is well-known. In our terminology, this duality swaps zeroes with ones, $\min$ with $\max$, and ‘$\leq$’ with ‘$\geq$’. This duality is the base of our 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 (3).

**Definition 4** The inverse of a vector $V$, denoted $I(V)$, is the vector

$$I(V) \triangleq \neg(\neg(V))$$

Namely, $I(V)$ 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 $I$ transformation is not an operator. Clearly, $I(I(V)) = V$ for every vector $V$. 

8
Definition 5 Let $\alpha$ be an operator.

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

In other words, $\alpha$ is symmetric if and only if $\alpha$ commutes with $I$. Natural operators are usually symmetric as demonstrated in the following lemma.

Lemma 6 All the operators ‘$\sqcap$', ‘$\checkmark$', ‘$\leftarrow$', ‘$\land$', ‘MinMax' and ‘$B$’ are symmetric.

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

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

As demonstrated in Section 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:

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

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 7 Let $\alpha$ and $\beta$ be two operators. Then

a. $\tilde{\beta \circ \alpha} = \tilde{\beta} \circ \tilde{\alpha}$.

b. $\tilde{\beta + \alpha} = \tilde{\alpha} + \tilde{\beta}$

c. If $\alpha$ and $\beta$ are symmetric, than $\alpha \circ \beta$ is symmetric.
d. If $\alpha = \widehat{\beta}$, than $\alpha + \beta$ is symmetric.

e. Let $\gamma$ be a symmetric operator and let $\alpha = \widehat{\beta}$; then $\alpha + \gamma + \beta$ is symmetric.

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 $I(w)$ is sorted.
- A word $w$ is Bitonic if and only if $I(w)$ is Bitonic.
- A 2-vector $V$ is a pair of sorted words if and only if $I(V)$ 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 $\widehat{\alpha}(I(w))$ is sorted.

**Theorem 8** 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 9** 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 9, then $i = j$.

**Theorem 10** 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).

$^2$Recall that a word is a binary sequence.
• *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 10.

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. We define the dual network, $\tilde{N}$, of a network $N$ as follows. First, $N$ and $\tilde{N}$ have the same graph: the same vertices and the same edges. Next, the types of the edges are flipped; that is, a Min-edge in $N$ becomes a Max-edge in $\tilde{N}$ and a Max-edge in $N$ becomes a Min-edge in $\tilde{N}$. 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, this changes are equivalent to applying the reverse operator on the input vector and on the output vector. The input signature and output signature of $\tilde{N}$ 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 $\tilde{N}$, $p$ receives the $i$’th last key of the $j$’th last word of the input vector. Figure 3 depicts Batcher’s Bitonic sorter of width 8 and its dual network. Figure 4 depicts Batcher’s Odd-Even merging network of width 8 and its dual network.

![Figure 3: Batcher’s Bitonic sorter of width 8 and its dual network.](image)

![Figure 4: Batcher’s odd-even merging network of width 8 and its dual network.](image)

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 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 3 are isomorphic and that the two networks of Figure 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, these networks of Batcher are usually strongly symmetric. 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 3 and 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.

It is not hard to see that a strongly symmetric network is also weakly symmetric; that is, it computes a symmetric operator; however, the opposite does not 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. This does not imply that the network is strongly symmetric. In fact, when \(n\) is odd and greater then one, such a network is certainly not strongly symmetric due to the following arguments. Let \(w\) be the input vector of a sorting network \(N\) of width \(2k + 1\). The middle key, \(w_{k+1}\), clearly enters a comparator; let \(e\) be the other incoming edge of this comparator. In \(\tilde{N}\), the same middle key enters the same comparator; let \(e'\) be the other incoming edge of this comparator. We consider two cases according to the source of \(e\). If \(e\) arrives from a comparator then the same holds for \(e'\). However, the Min/Max type of \(e\) and \(e'\) are different. If \(e\) arrives from an input port then so does \(e'\). However, the two ports from which \(e\) and \(e'\) arrive are associated with different members of \(w\). In both cases \(N\) and \(\tilde{N}\) are not isomorphic.

### 3 Bitonic Sorters of even width

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

1. The algorithm \(A\) sorts every Bitonic 1-heavy word.

\(^3\)This is an abuse of terminology. By definition [5], 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.
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 10, $B$ 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.

3.1 Sorting 1-heavy words

We define two 1-to-2 operators of depth one, called $\mathcal{LS}$ and $\mathcal{SL}$, that partition a word $w$ into two words $x; y$ 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 $\mathcal{LS}$ and $\mathcal{SL}$ 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 $h_{x,y}(w) = \mathcal{LS}(w)$ then $|x| > |y|$ and if $h_{x,y}(w) = \mathcal{SL}(w)$ 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 $|b| = 1$. (Note that $b$ may be empty). Then

\[
\mathcal{LS}(u \cdot b \cdot v) \triangleq \langle \text{Min}(u \cdot b, \leftarrow (b \cdot v)), \text{Max}(\leftarrow (u), v) \rangle
\]

\[
\mathcal{SL}(u \cdot b \cdot v) \triangleq \langle \text{Min}(u, \leftarrow (v)), \text{Max}(\leftarrow (u \cdot b), b \cdot v) \rangle
\]

(10)

(11)

It is not hard to see that each of the operators $\mathcal{LS}$ and $\mathcal{SL}$ 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 $h_{x,y}(w)$ is a split of a word $w$, denoted $h_{x,y}(w)$, if $h_{x,y}(w) = \mathcal{LS}(w)$ or $h_{x,y}(w) = \mathcal{SL}(w)$.

In this section we use the symbol ‘$\leq$’ in a special manner, as follows. The symbol ‘$\leq$’ 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 ‘$e_1 \leq e_2$’ is not meaningless; rather, this phrase is considered to be true. We use this symbol in the following restricted context. For an empty word $w$, the notations $w_1$ and $w_{|w|}$ are undefined; this leads to phrases of the above form. Splitting a word may generate empty words and the special meaning of ‘$\leq$’ is useful in this context. To illustrate the convenience of these notations consider the following lemma whose proof is straightforward.

**Lemma 11** Let each of $q; r; s$ and $t$ be either binary or undefined and let $q + r \leq s + t$. Then $\min(q, r) \leq \min(s, t)$ and $\max(q, r) \leq \max(s, t)$.

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 12** Let $v$ be a Bitonic word and let $v \approx \langle x, y \rangle$. Then:

a. $x \cdot y = v$ or $x \cdot y = \langle v \rangle$.

b. $x$ is descending-ascending.
c. \( y \) is ascending-descending.

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

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

Lemma 13 Let \( v \) be descending-ascending word and let \( v \approx \langle x, y \rangle \). Then:

a. \( y \) is ascending.

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

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

Define \( 0^k \) and \( 1^k \) as the words of width \( k \) that contain only 0’s and only 1’s, respectively. Define \( 0^* \triangleq \{0^k \mid k \in \mathbb{N}\} \) and \( 1^* \triangleq \{1^k \mid k \in \mathbb{N}\} \). 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 14 Let \( w \) be 1-heavy and Bitonic, let \( w \approx \langle a, b \rangle \) and let \( b \approx \langle c, r \rangle \). Then \( r \in 1^* \).

Proof: By Lemma 12(c,d), the word \( b \) is ascending-descending and \( 1 \leq b_1 + b_{|b|} \). Therefore, \( b \) is either ascending or descending. Again, by Lemma 12(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 15 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. \( d_1 + d_{|d|} \leq m'_1 + m''_1 \).

The algorithm \( A \) is composed of three parts. The first part, called the prologue, contains two stages, each of depth one. The first stage performs the 1-to-2 operator \( \mathcal{LS} \) and the second stage performs the 2-to-4 operator \( \mathcal{LS} + \mathcal{SL} \). The following lemma establishes that the result of this prologue have the \( \diamond \)-property.

Lemma 16 Let \( w \) be a 1-heavy Bitonic word. Let \( \langle a, b \rangle = \mathcal{LS}(w) \) and let \( \langle d, m', m'', r \rangle = (\mathcal{LS} + \mathcal{SL})(\langle a, b \rangle) \). Then \( \langle d, m', m'' \rangle \) have the \( \diamond \)-property.
Proof: We verify the four conditions of Definition 15, as follows.

Condition (a) follows from Lemma 12(b).
Consider Condition (b). By straightforward arithmetics, $|m'| = |m''|$. By Lemma 12, $a$ is descending-ascending and $b$ is ascending-descending. By Lemma 13(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 12(e):
\[ n^0(d) \geq n^0(m') \]  \hspace{1cm} (12)

By our construction and previous lemmas,
\[ n^0(d) + n^0(m') = n^0(a) \geq n^0(b) = n^0(m'') \]  \hspace{1cm} (13)

The last two inequalities with the fact that $n^0$ is always non-negative imply:
\[ n^0(d) \geq n^0(m') - n^0(m'') \text{ and } n^0(d) \geq n^0(m'') - n^0(m'). \]  \hspace{1cm} (14)

which 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 13(c). Next, consider the second inequality. By Lemma 12(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 11, this establishes the second inequality, proving Condition (d). \[ \blacksquare \]

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 16. Namely, it transforms the input word $w$ into a 4-vector $\langle d, m', m'', r \rangle$. By Lemma 16, $\langle d, m', m'' \rangle$ have the $\diamondsuit$-property and by Lemma 14, $r \in 1^*$. 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 $\diamondsuit$-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 $\diamondsuit$-property, and a reservoir $r$ of all 1’s. It extracts, with no comparisons, a word $\tilde{r}$ (of a certain width) out of $r$ and combines it with $\langle d, m', m'' \rangle$ into a new 3-vector $\langle d, \tilde{m}', \tilde{m}'' \rangle$ having the $\diamondsuit$-property. Moreover, $|\tilde{d}| = \lceil |d| / 2 \rceil$. The 3-vector $\langle \tilde{d}, \tilde{m}', \tilde{m}'' \rangle$ 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 $\mathcal{M}ax\mathcal{M}in \triangleq (\triangleright \circ \mathcal{M}in, \mathcal{M}ax)$. The above combination and the fact that the resulting 3-vector has the $\diamondsuit$-property is stated in the following lemma.

Lemma 17 Let the 3-vector $\langle d, m', m'' \rangle$ have the $\diamondsuit$-property and let $\tilde{r} = 1^{\lfloor |d| / 2 \rfloor}$. Let $\langle \tilde{d}, \tilde{d} \rangle = \mathcal{L}S(d)$, $\langle \tilde{m}', \tilde{m}'' \rangle = \mathcal{M}ax\mathcal{M}in(\langle m', m'' \rangle)$, $\tilde{m}' = \tilde{d} \cdot \tilde{m}'$ and $\tilde{m}'' = \tilde{m}'' \cdot \tilde{r}$. Then $\langle \tilde{d}, \tilde{m}', \tilde{m}'' \rangle$ have the $\diamondsuit$-property.
Proof: We verify the four conditions of Definition 15, as follows.

Condition (a) holds by Lemma 12(b).

Consider Condition (b). By our construction, the size of \( \tilde{r} \) was selected so that \( |\tilde{m}'| = |\tilde{m}''| \). It remains to show that \( \tilde{m}' \) and \( \tilde{m}'' \) are sorted. By Lemma 13(a), \( \tilde{d} \) is sorted. Clearly, \( \tilde{m}' \) and \( \tilde{m}'' \) 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 \( \tilde{m}'' \) is sorted follows from the fact that \( \tilde{m}' \) is sorted and that \( \tilde{r} \in \mathbb{1}^* \). It remains to show that \( \tilde{m}' \) is sorted. Namely, that \( \tilde{d}_{|\tilde{d}|} \preceq \tilde{m}' \).

By Condition (d) w.r.t. \( \langle d, m', m'' \rangle \):

\[
d_1 + d_{|\tilde{d}|} \leq m'_1 + m''_1
\]

By our construction,

\[
\tilde{d}_{|\tilde{d}|} = \max(d_1, d_{|\tilde{d}|}) \text{ and } \tilde{m}'_1 = \max(m'_1, m''_1)
\]

By Lemma 11, \( \tilde{d}_{|\tilde{d}|} \preceq \tilde{m}'_1 \) implying Condition (b).

Consider Condition (c). The \( \heartsuit \)-property of \( \langle d, m', m'' \rangle \) 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,

\[
0 \leq n^0(\tilde{m}'') - n^0(\tilde{m}') \leq n^0(d)
\]

(15)

Lemma 12(e), w.r.t. the splitting of \( d \) and the fact that \( d \) and \( \langle \tilde{d}, \tilde{d} \rangle \) are isomeric imply that:

\[
-n^0(\tilde{d}) \leq -n^0(\tilde{d}) = -n^0(d) + n^0(\tilde{d})
\]

(16)

Adding Inequalities (15) and (16) yields:

\[
-n^0(\tilde{d}) \leq n^0(\tilde{m}'') - (n^0(\tilde{m}') + n^0(\tilde{d})) \leq n^0(\tilde{d})
\]

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 \( \tilde{d}_{|\tilde{d}|} \preceq \tilde{m}'_1 \) and \( \tilde{d}_1 \preceq \tilde{m}'_1 \). The first inequality, \( \tilde{d}_{|\tilde{d}|} \preceq \tilde{m}'_1 \), 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 13(c), \( \tilde{d}_{|\tilde{d}|} \preceq \tilde{d}_1 = \tilde{m}'_1 \). Assume next that \( |\tilde{d}| = 0 \). In this case, \( \tilde{m}'_1 = \max(m'_1, m''_1) \) and \( \tilde{d}_{|\tilde{d}|} = \max(d_1, d_{|\tilde{d}|}) \). Condition (d) w.r.t. \( \langle d, m', m'' \rangle \) states that \( d_1 + d_{|\tilde{d}|} \preceq m'_1 + m''_1 \). By Lemma 11:

\[
\tilde{d}_{|\tilde{d}|} = \max(d_1, d_{|\tilde{d}|}) \preceq \max(m'_1, m''_1) = \tilde{m}'_1
\]

Next, consider the second inequality; namely, that \( \tilde{d}_1 \preceq \tilde{m}'_1 \). As said, since Condition (d) holds w.r.t. \( \langle d, m', m'' \rangle \):

\[
d_1 + d_{|\tilde{d}|} \leq m'_1 + m''_1
\]
By our construction and Lemma 11:

\[ \tilde{d}_1 = \min(d_1, d_{|d|}) \preceq \min(m'_1, m''_1) = \tilde{m}'_1 \]

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 \( \tilde{d} \) is minimal; furthermore, the remaining key in the reservoir (if there is such a key) is 1. However, there are pairs of keys, \( k' \in \tilde{m}' \) and \( k'' \in \tilde{m}'' \) such that the relative order between \( k' \) and \( k'' \) is not determined. By Condition (c) of Definition 15, \(|n^0(m') - n^0(m'')| \leq 1\). This enables the last part of \( A \) – the epilogue – to merge the two sorted words \( \tilde{m}' \) and \( \tilde{m}'' \) in a single stage by performing the \( \text{MaxMin} \) operator on \( \langle \tilde{m}', \tilde{m}'' \rangle \). 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; each iteration has a single stage and it reduces \(|d| \) to \( \lceil |d|/2 \rceil \). 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 18 The network \( A(n) \) sorts every 1-heavy Bitonic word of width \( n \) and its depth is at most \( \lceil \log(n) \rceil + 1 \).

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 14, the reservoir contains only 1’s. In each iteration, the algorithm \( A \) extracts, without any computation, a word \( \tilde{r} \) from the reservoir. In contrast to \( A \), the algorithm \( B \) extracts a word \( \tilde{r} \) (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 symmetric and allows us to use Theorem 10. This symmetry is achieved as follows. In each iteration, \( B \) splits the word \( r \) by \( \langle \tilde{r}, \bar{r} \rangle = S\mathcal{L}(r) \); \( \tilde{r} \) 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 and 7(e), in each iteration, $B$ calculates a symmetric operator. By Lemma 7(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 19** 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 cannot be processed in a dual manner.

The main result of this section is the following lemma.

**Lemma 20** 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 10 and Lemma 19, $B$ 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 [7], the depth of any Bitonic sorter of width $n$, when $n$ is not a power of two, is at least $\lceil \log(n) \rceil + 1$.

Next assume that $k$ is a power of two. By [6] (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 $B(2k)$.

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 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 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 $\mathcal{ID}$, 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 $\mathcal{ID}$ is the total operator defined by $\mathcal{ID}(V) \triangleq V$.

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 $\pi|_k$ denote the restriction of $\pi$ to $\{0, 1\}^*^k$; namely, $\pi|_k \triangleq \pi|_{\{0, 1\}^k}$.

Using our terminology, $\alpha$ is the 1-to-4 operator defined by $\alpha \triangleq (\mathcal{ID}|_1 + \Phi + \Phi + \mathcal{ID}|_1) \circ \mathcal{LS}$. Note that $\mathcal{LS}$ is of depth one and $(\mathcal{ID}|_1 + \Phi + \Phi + \mathcal{ID}|_1)$ is of depth zero; hence, $\alpha$ is of depth one.

The operator $\beta$ is the 4-to-4 operator defined by $\beta \triangleq (\mathcal{ID}|_1 + \mathcal{C}|_2 + \mathcal{C}|_2 + \mathcal{ID}|_1) \circ (\mathcal{LS} + \text{MaxMin} + \mathcal{SL})$. Again, $\beta$ is of depth one. As said, $\beta$ is iterated $\lceil \log(n) \rceil$ times.

The first and last iterations of $\beta$ are degenerated in the following senses. As evident in Figure 5, the $\text{MaxMin}$ 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 $\mathcal{LS}$ and $\mathcal{SL}$ 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 $\mathcal{LS}$ operator and from the $\mathcal{SL}$ operator to the $\mathcal{C}|_2$ operators are empty. Moreover, at this point, the resulting 4-vector $\langle d, m', m'', r \rangle$ is always a fixed point 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 20. However, when $n \leq 2$, the operator $\beta$, in this context, is of depth zero and is clearly not necessary.

As discussed in Section 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 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, the operators $\text{MaxMin}$ is symmetric. Clearly, $\mathcal{C}|_2$ and $\mathcal{ID}|_1$ are symmetric. As said, $\mathcal{LS}$ and $\mathcal{SL}$ are the dual of each other. By Lemma 7, each of the operators $\alpha$ and $\beta$ are symmetric. Again, by Lemma 7, 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 5. There are only minor differences between $B$ and $C$, as follows:

- The two operators, $\mathcal{LS}$ and $\mathcal{SL}$, sometimes replace each other.
- The two functions, ‘$[\cdot]$’ and ‘$\langle \cdot \rangle$’, sometimes replace each other.
- The width of two corresponding words may differ by at most one.
Figure 5: Graphical representation of the networks $B(2k)$ and $C(2k - 1)$
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 $C$:

$$
\alpha = (\mathcal{ID}|_1 + \Phi + \Phi + \mathcal{ID}|_1) \circ \mathcal{SL}
\beta = (\mathcal{ID}|_1 + \mathcal{C}|_2 + \mathcal{C}|_2 + \mathcal{ID}|_1) \circ (\mathcal{SL} + \text{MaxMin} + \mathcal{SL})
$$

Again, these operators are of depth one. As said, the right hand part of Figure 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 5, the iterative part of each network is divided into three sub-networks: $\mathcal{D}$ (Down), $\mathcal{M}$ (Middle) and $\mathcal{U}$ (Up). This naming relates to the fact that the algorithm tries to keep the smaller keys in $\mathcal{D}$ and the larger keys in $\mathcal{U}$.

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

$$\mathcal{M}_{B(2k)} = \mathcal{M}_{C(2k-1)} \quad \text{(17)}$$

$$\mathcal{U}_{B(2k)} = \mathcal{U}_{C(2k-1)} \quad \text{(18)}$$

## 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 $\mathcal{D}$ or $\mathcal{U}$ is a minimal key or a maximal key, respectively. Hence, the core of our proof is the claim that the output of $\mathcal{M}$ 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 $\mathcal{M}_{C(2k-1)}$ and $\mathcal{M}_{B(2k)}$ receive the same vector. By Equality 17, $\mathcal{M}_{B(2k)} = \mathcal{M}_{C(2k-1)}$; hence, these two sub-networks produce the same vector. Since $B(2k)$ is a Bitonic sorter, the output of $\mathcal{M}_{B(2k)}$ is sorted. Hence, the same holds for $\mathcal{M}_{C(2k-1)}$.

To construct the above $w'$, let $t : \{0, 1\}^* \rightarrow \{0, 1\}^*$ 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: $t(00100010) = 0000100010$, $t(1011100) = 10111100$, $t(11) = 110$.

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 21** Let $w$ be a Bitonic word\(^4\) and let $\langle x, y \rangle = \mathcal{SL}(w)$. Then either $\mathcal{LS}(t(w)) = \langle t(x), y \rangle$ or $\mathcal{LS}(t(w)) = \langle \langle t(x) \rangle, y \rangle$.

\(^4\)Note that $w$ is not necessarily 1-heavy.

\(^5\) $\mathcal{LS}(t(w)) \neq \langle t(x), y \rangle$ only when $w \in 1^*$.  

21
Proof:

Let \( w' = t(w) \) and let \( (x', y') = \text{LS}(w') \). We need to show that \( y' = y \) and that \( x' = t(x) \) or \( x' = \leftarrow (t(x)) \). 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: \( w = 1^01^{i+1}1^k \) for some \( i, j, k \geq 0 \).

By Lemma 13, \( y \) and \( y' \) are ascending. Clearly:

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

Therefore, \( y = y' \).

The facts that \( \text{LS} \) and \( \text{SL} \) are isomeric and that \( y = y' \) imply that \( n^1(x') = n^1(x) \) and \( n^0(x') = n^0(x) + 1 \). By Lemma 13, 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: \( w = 0^10^k \) for some \( i, j, k \geq 0 \).

We first consider the words \( x \) and \( x' \). By duality and Lemma 13, these words are ascending. Clearly:

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

Therefore, \( x' = t(x) \) or \( x' = \leftarrow (t(x)) \). (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 12(c), \( y \) and \( y' \) are ascending-descending. It remains to show that \( H^0(\leftarrow (y)) = H^0(\leftarrow (y')) \). In fact, both these numbers are \( \min(i, k) \).

\[\blacksquare\]

Lemma 22 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 22 is not necessarily 1-heavy.

Proof: Refer to Figure 5. The sub-network \( M \) receives one vector from \( \mathcal{D} \) and one vector from \( \mathcal{U} \). Consider the latter. By Lemma 21, \( \mathcal{U}_{B(2k)} \) and \( \mathcal{U}_{C(2k-1)} \) receive the same word. By Equality 18, \( \mathcal{U}_{C(2k-1)} = \mathcal{U}_{B(2k)} \); hence, they produce the same vector.

Next, consider the words that are transferred from \( \mathcal{D} \) to \( M \). As shown in Figure 5, in each \( \beta \) iteration \( C(2k - 1) \) performs the operator \( \text{SL} \) on its \( d \) while \( B(2k) \) performs the operator \( \text{LS} \) on its \( d \). Let \( x = (x_1, x_2 \ldots x_{\log(2k-1)}) \) where \( x_i \) is the word that enters the operator \( \text{SL} \) in the \( i \)'th iteration of \( \beta \) in the network \( C(2k - 1) \). Let \( x' = (x'_1, x'_2 \ldots x'_{\log(2k)}) \) where \( x'_i \) is the word that enters the operator \( \text{LS} \) in the \( i \)'th iteration of \( \beta \) in the network \( B(2k) \). Clearly, \( \text{LS}(q) = \text{LS}(\leftarrow (q)) \), for every \( q \in \{0, 1\}^* \). Hence, by straightforward induction and Lemma 21, \( x'_i = t(x_i) \) or \( x'_i = \leftarrow (t(x_i)) \), for every \( i \). Again by Lemma 21, the words that are transferred, under \( w \), from \( \mathcal{D}_{C(2k-1)} \) to \( M_{C(2k-1)} \) are identical to the words transferred, under \( w' \), from \( \mathcal{D}_{B(2k)} \) to \( M_{B(2k)} \). \[\blacksquare\]
The above discussion is summarized in the following lemma.

**Lemma 23** For every $k$, the network $C(2k−1)$ is a Bitonic sorter.

Let us summarize the main result of this paper. By Lemmas 20 and 23, for every $n$, there is a Bitonic sorters of width $n$ and of depth $\lceil \log(n) \rceil + 1$. By Levy and Litman [4], 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 $2 \lceil \log(n) \rceil − \lfloor \log(n) \rfloor$.

### References


