Sorting Networks

6CCS3PAL-7CCSMPDA
Sorting networks

- Comparator
- Comparator network
- Sorting network
- 0-1 Principle
- Various sorts
- Bitonic sort
A sorting network consists of two types of items: comparators and wires. The wires are thought of as running from left to right, carrying values (one per wire) that traverse the network all at the same time. Each comparator connects two wires.
When a pair of values, traveling through a pair of wires, encounter a comparator, the comparator swaps the values if and only if the top wire’s value is greater than the bottom wire’s value.

If the top wire carries \( x \) and the bottom wire carries \( y \), then after hitting the comparator
the top wire carries \( x' = \min(x, y) \)
and the bottom one \( y' = \max(x, y) \)
Typically these diagrams assume an ascending comparator (sorts min, max)

But sometimes we will use a descending comparator (sorts max, min)
A network of wires and comparators that correctly sorts all possible inputs into ascending order is called a sorting network. A simple sorting network is shown below. The first four comparators "sink" the largest value to the bottom and "float" the smallest value to the top. The final comparator sorts out the middle two wires.
Sort 3 numbers?

- The basic comparator sorts 2 numbers
Sort 3 numbers?

- The basic comparator sorts 2 numbers.
- Can you think of a network to sort 3 numbers correctly?
Sort 3 numbers?

- The basic comparator sorts 2 numbers
- Can you think of a network to sort 3 numbers correctly?
- And how do you know it is correct?
Sort 3 numbers correctly

123, 132, 213, 231, 312, 321
Sort 3 numbers correctly

Check all?

123, 132, 213, 231, 312, 321
Check 231
Parallel sorting: comparator stages

We label the $n$ wires 0, 1, ..., $n - 1$
A comparator $[i : j]$ sorts the $i$–th and the $j$–th element of a data sequence into nondecreasing order.
A comparator stage is a sequential collection $S$ of comparators $S = [i_1 : j_1]...[i_k : j_k]$, such that all $i_r$ and $j_s$ are distinct (all start and end wires are distinct, and no other comparators in between).
Comparators within a comparator stage can be executed in parallel.
How many stages does the following figure have?

The figure has 2 stages.
How many stages does the following figure have?

The figure has 2 stages.
How many stages?

3 stages
A comparator network is a composition of comparator stages.

A sorting network is a comparator network that sorts all input sequences. The comparator network of the example above is not a sorting network, since it does not (e.g.) sort the sequences 3 1 4 2 or 3 2 4 1. It is the same as the introductory network, but the last comparator is missing.
Sorting network: Bubblesort

An example of a sorting network

The Bubblesort network has a first diagonal of \( n - 1 \) comparators to move the greatest element to the last position. The remaining \( n - 1 \) elements are sorted recursively by applying the same procedure. The second diagonal moves second largest to second from last position etc.
Bubblesort has $n(n-1)/2$ comparators, and $2n-3$ comparator stages.
The figure shows making $BS(6)$ from $BS(5)$ by adding the top line of comparators.
One new one at each end (9 stages from 7)
3 stage Bubblesort
This is Bubblesort from an Interconnection Networks lecture

**Algorithm Bubble-Sort** \((A[1..n])\)

for \(i = 1...n - 1\) do
  for \(j = 1...n - 1\) do
    if \(A[j] > A[j + 1]\) then
      switch entries of \(A[j]\) and \(A[j + 1]\)
  end Bubble-Sort

To keep the code simple we included more comparisons then we needed
For $n = 3$ the code says

define
for i=1 compare 12 23
for i=2 compare 12 23

We used 4 comparators and stages. We only needed 3

--- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
The network odd-even transposition sort for n input data consists of n comparator stages. At each stage, either all inputs at odd index positions or all inputs at even index positions are compared with their neighbours. Odd and even stages alternate.

We used this network for sorting on 1-D meshes.
Even-Odd Transposition Sort on $1D$ Mesh

**Algorithm Even-Odd-Sort-$1D$-Mesh** \(L[0..n-1]\)
For Step \(s = 0..n-1\) do

If Step \(s\) Even then
    For all Even processor labels \(i\) in parallel do
        Compare-Exchange \((L[i], L[i+1])\)
else
    If Step \(s\) Odd then
        For all Odd processor labels \(i\) in parallel do
            Compare-Exchange \((L[i], L[i+1])\)
end algorithm
Q: How do we know these sorting networks really work properly?
Q: How do we know these sorting networks really work properly?

A: It would need to be proved.
Q: How do we know these sorting networks really work properly?
A: It would need to be proved
Q: How could we do that?
Q: How do we know these sorting networks really work properly?

A: It would need to be proved.

Q: How could we do that?

A: The 0–1 principle can be used to check the correctness of sorting networks.
Q: How do we *know* these sorting networks really work properly?
A: It would need to be proved
Q: How could we do that?
A: The 0–1 principle can be used to check the correctness of sorting networks
But it is not an easy question to answer
The 0-1-principle

Whether an arbitrary comparator network is a sorting network or not is independent of the input set. It only depends on the structure of the network. The 0-1-principle essentially states this fact.

**Theorem: (The 0-1-principle)**

A comparator network with $n$ inputs that sorts all $2^n$ sequences of zeroes and ones is a sorting network (i.e. it sorts all sequences of arbitrary values, too).
Theorem: (The 0-1-principle)

A comparator network with \( n \) inputs that sorts all \( 2^n \) sequences of zeroes and ones is a sorting network (i.e. it sorts all sequences of arbitrary values, too).

Basic idea: if \( a < b \) but we get mistake \( \ldots b, a \ldots \) in sorted sequence, then put all numbers \( a \) less than \( b \) to zero and all numbers \( \geq b \) to 1. The 0 for \( a \) and 1 for \( b \) are still out of place when we sort the 0–1 sequence.
Theorem: (The 0-1-principle)

Why useful?
Theorem: (The 0-1-principle)

Why useful?
A comparator network with $n$ inputs that sorts all $2^n$ sequences of zeroes and ones is a sorting network.
Theorem: (The 0-1-principle)

Why useful?
A comparator network with $n$ inputs that sorts all $2^n$ sequences of zeroes and ones is a sorting network.
We can check $2^n$ inputs, or prove the network is ok for 0–1 inputs but we can’t check all sequences of $n$ numbers (Why not?)
Exercise: Check for all 0 – 1 sequences

We only need to check 100, 010, 101, 110. Why not 000, 001, 011, 111?
A sorting network is a special kind of sorting algorithm, where the sequence of comparisons is not data-dependent. This makes sorting networks suitable for implementation in hardware or in parallel processor arrays. Bitonic sort [Batcher 1968] is one of the fastest sorting networks. The sorting network bitonic sort consists of $O(n \log^2(n))$ comparators and sorts in $O(\log^2(n))$ parallel time (span). The term bitonic is a technical term meaning that the sequence is twofold monotonic.
A 0-1-sequence is called bitonic, if it contains at most two changes between 0 and 1
i.e. if there exist subsequence lengths $k, m$ such that

$a_0, ..., a_{k-1} = 0, a_k, ..., a_{m-1} = 1, a_m, ..., a_{n-1} = 0$ or
$a_0, ..., a_{k-1} = 1, a_k, ..., a_{m-1} = 0, a_m, ..., a_{n-1} = 1$
A special network $B_n$

$B_2$ is just the basic comparator.

For $n$ even the comparator network $B_n$ is defined as follows:

$$B_n = [0 : n/2] \ [1 : n/2 + 1] \ \ldots \ \ [n/2 - 1 : n - 1]$$

Example: $n = 8$ so $n/2 = 4$ and edges 04, 15, 26, 37
For $n$ even, let $a = a_0, \ldots, a_{n-1}$ be a bitonic 0-1-sequence.

- Application of comparator network $B_n$ to $a$ yields $B_n(a) = b_0, \ldots, b_{n/2-1}, c_0, \ldots, c_{n/2-1}$ where:
  - All $b_i$ are less than or equal to all $c_j$, i.e. $b_i \leq c_j$ for all $i, j \in \{0, \ldots, n/2-1\}$
  - Both halfs are bitonic $b_0, \ldots, b_{n/2-1}$ is bitonic and $c_0, \ldots, c_{n/2-1}$ is bitonic
- The theorem says: Afterwards at least one half is completely sorted (all 0 or all 1). Why?
  - Either the initial sequence $a$ was all 0 or all 1, or
  - Either the first half of the final sequence is all 0, or the second half is all 1
- It also says we can reapply $B_{n/2}$ to the final half sequences.
Theorem

For $n$ even, let $a = a_0, \ldots, a_{n-1}$ be a bitonic 0-1-sequence.

- Application of comparator network $B_n$ to $a$ yields $B_n(a) = b_0, \ldots, b_{n/2-1}, c_0, \ldots, c_{n/2-1}$ where:
  
  ▶ All $b_i$ are less than or equal to all $c_j$, i.e. $b_i \leq c_j$ for all $i, j \in \{0, \ldots, n/2-1\}$

  ▶ Both halfs are bitonic $b_0, \ldots, b_{n/2-1}$ is bitonic and $c_0, \ldots, c_{n/2-1}$ is bitonic

  ▶ The theorem says: Afterwards at least one half is completely sorted (all 0 or all 1)

  ▶ Either the initial sequence $a$ was all 0 or all 1, or

  ▶ Either the first half of the final sequence is all 0, or the second half is all 1

  ▶ It also says we can reapply $B_{n/2}$ to the final half sequences
Theorem

For $n$ even, let $a = a_0, \ldots, a_{n-1}$ be a bitonic 0-1-sequence.

- Application of comparator network $B_n$ to $a$ yields $B_n(a) = b_0, \ldots, b_{n/2-1}, c_0, \ldots, c_{n/2-1}$ where:
  - All $b_i$ are less than or equal to all $c_j$, i.e. $b_i \leq c_j$ for all $i, j \in \{0, \ldots, n/2 - 1\}$
  - Both halfs are bitonic $b_0, \ldots, b_{n/2-1}$ is bitonic and $c_0, \ldots, c_{n/2-1}$ is bitonic

The theorem says: Afterwards at least one half is completely sorted (all 0 or all 1) Why?

- Either the initial sequence $a$ was all 0 or all 1, or
- Either the first half of the final sequence is all 0, or the second half is all 1

It also says we can reapply $B_n$ to the final half sequences
Theorem

For $n$ even, let $a = a_0, \ldots, a_{n-1}$ be a bitonic 0-1-sequence.

- Application of comparator network $B_n$ to $a$ yields $B_n(a) = b_0, \ldots, b_{n/2-1}, c_0, \ldots, c_{n/2-1}$ where:
  - All $b_i$ are less than or equal to all $c_j$, i.e. $b_i \leq c_j$ for all $i, j \in \{0, \ldots, n/2 - 1\}$
  - Both halves are bitonic
    $b_0, \ldots, b_{n/2-1}$ is bitonic and $c_0, \ldots, c_{n/2-1}$ is bitonic

- Why?
  - Either the initial sequence $a$ was all 0 or all 1, or
  - Either the first half of the final sequence is all 0, or the second half is all 1

- It also says we can reapply $B_n$ to the final half sequences
Theorem

For $n$ even, let $a = a_0, \ldots, a_{n-1}$ be a bitonic 0-1-sequence.

- Application of comparator network $B_n$ to $a$ yields $B_n(a) = b_0, \ldots, b_{n/2-1}, c_0, \ldots, c_{n/2-1}$ where:
  - All $b_i$ are less than or equal to all $c_j$, i.e. $b_i \leq c_j$ for all $i, j \in \{0, \ldots, n/2 - 1\}$
  - Both halves are bitonic $b_0, \ldots, b_{n/2-1}$ is bitonic and $c_0, \ldots, c_{n/2-1}$ is bitonic
  - The theorem says: Afterwards at least one half is completely sorted (all 0 or all 1) Why?

- Either the initial sequence $a$ was all 0 or all 1, or
  - Either the first half of the final sequence is all 0, or the second half is all 1

- It also says we can reapply $B_n/2$ to the final half sequences
Theorem

For $n$ even, let $a = a_0, \ldots, a_{n-1}$ be a bitonic 0-1-sequence.

- Application of comparator network $B_n$ to $a$ yields $B_n(a) = b_0, \ldots, b_{n/2-1}, c_0, \ldots, c_{n/2-1}$ where:
  - All $b_i$ are less than or equal to all $c_j$, i.e. $b_i \leq c_j$ for all $i, j \in \{0, \ldots, n/2 - 1\}$
  - Both halves are bitonic $b_0, \ldots, b_{n/2-1}$ is bitonic and $c_0, \ldots, c_{n/2-1}$ is bitonic
- The theorem says: Afterwards at least one half is completely sorted (all 0 or all 1) Why?
- Either the initial sequence $a$ was all 0 or all 1, or
Theorem

For $n$ even, let $a = a_0, \ldots, a_{n-1}$ be a bitonic 0-1-sequence.

- Application of comparator network $B_n$ to $a$ yields $B_n(a) = b_0, \ldots, b_{n/2-1}, c_0, \ldots, c_{n/2-1}$ where:
  - All $b_i$ are less than or equal to all $c_j$, i.e. $b_i \leq c_j$ for all $i, j \in \{0, \ldots, n/2 - 1\}$
  - Both halves are bitonic
    - $b_0, \ldots, b_{n/2-1}$ is bitonic and $c_0, \ldots, c_{n/2-1}$ is bitonic
  - The theorem says: Afterwards at least one half is completely sorted (all 0 or all 1) Why?
    - Either the initial sequence $a$ was all 0 or all 1, or
    - Either the first half of the final sequence is all 0, or the second half is all 1
Theorem

For $n$ even, let $a = a_0, \ldots, a_{n-1}$ be a bitonic 0-1-sequence.

- Application of comparator network $B_n$ to $a$ yields $B_n(a) = b_0, \ldots, b_{n/2-1}, c_0, \ldots, c_{n/2-1}$ where:
  - All $b_i$ are less than or equal to all $c_j$, i.e. $b_i \leq c_j$ for all $i, j \in \{0, \ldots, n/2 - 1\}$
  - Both halves are bitonic
    - $b_0, \ldots, b_{n/2-1}$ is bitonic and $c_0, \ldots, c_{n/2-1}$ is bitonic
  - The theorem says: Afterwards at least one half is completely sorted (all 0 or all 1) Why?
    - Either the initial sequence $a$ was all 0 or all 1, or
    - Either the first half of the final sequence is all 0, or the second half is all 1
  - It also says we can reapply $B_{n/2}$ to the final half sequences
$B_{16}$ acting on bitonic 0–1-sequences

0’s are drawn white and 1’s gray.
$B_n$ is sometimes called a half cleaner because afterwards half the sequence is clean.
How to use this?

- $B_n$ is sometimes called a half cleaner because afterwards half the sequence is clean
- First we need to build a bitonic sequence of length $n$ (Bitonic Build)
How to use this?

- $B_n$ is sometimes called a half cleaner because afterwards half the sequence is clean.
- First we need to build a bitonic sequence of length $n$ (Bitonic Build).
- Then we need to merge the bitonic sequence correctly by $B_n$ followed by two parallel $B_{n/2}$ etc (Bitonic Merge).
How to use this?

- $B_n$ is sometimes called a half cleaner because afterwards half the sequence is clean.
- First we need to build a bitonic sequence of length $n$ (Bitonic Build).
- Then we need to merge the bitonic sequence correctly by $B_n$ followed by two parallel $B_{n/2}$ etc (Bitonic Merge).
- In summary. Build a bitonic sequence and sort it using a bitonic merging network.
How to use this?

- $B_n$ is sometimes called a half cleaner because afterwards half the sequence is clean.
- First we need to build a bitonic sequence of length $n$ (Bitonic Build).
- Then we need to merge the bitonic sequence correctly by $B_n$ followed by two parallel $B_{n/2}$ etc (Bitonic Merge).
- In summary. Build a bitonic sequence and sort it using a bitonic merging network.
- In both cases we use $B_n$. There are several ways to put them together.
Example: Action of $B_n$

Figure 27.9 The comparison network $\text{Bitonic-Sorter}[n]$, shown here for $n = 8$. (a) The recursive construction: $\text{Half-Cleaner}[n]$ followed by two copies of $\text{Bitonic-Sorter}[n/2]$ that operate in parallel. (b) The network after unrolling the recursion. Each half-cleaner is shaded. Sample zero-one values are shown on the wires.

In this course Bitonic-Sorter [CLRS] is called Bitonic Merge. Figure from another version of CLRS Chapter 27 given at mitpress.mit.edu/sites/default/files/Chapter%2027.pdf
Bitonic sort $n = 8$

- $x$, $x'$ sorted (ascending)
- $y$, $y'$ sorted (descending)
- $a$, $a'$ bitonic
- $b \leq c$
- $b' \geq c'$
- $b, c, b', c'$ bitonic
- $d$ sorted (ascending)
- $d'$ sorted (descending)
- $f \leq f'$
- $g \leq h \leq g' \leq h'$
- $f, f'$ bitonic
- $g, h, g', h'$ i sorted
- $e$ bitonic
- $i$ bitonic
Bitonic sort $n = 2$

This $B_2$ is $\oplus B_2$,

This one is $\ominus B_2$
Bitonic sort $n = 4$

The logic

Sequence $a$ bitonic

$b \leq c$ bitonic

$d$ sorted within $b, c$

<table>
<thead>
<tr>
<th>$\oplus B_2$</th>
<th>$B_4$</th>
<th>$B_2$</th>
<th>$B_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ominus B_2$</td>
<td>Build</td>
<td>Merge</td>
<td></td>
</tr>
</tbody>
</table>
Bitonic sort $n = 4$

Write $BM[2] = B_2$, and for $n = 4$, $BM[4]$ is

$$BM[4] = \begin{array}{|c|c|}
B_4 & B_2 \\
\hline
B_2 & B_2 \\
\end{array}$$

$$\begin{array}{|c|c|}
\ominus BM[2] & Build \\
\end{array}$$

Merge
Bitonic sort $n = 8$

$\oplus B_2 \oplus B_2 \oplus B_4 \oplus B_2 \oplus B_2 \oplus B_2 \oplus B_2 \oplus B_2$

$B_8 \quad B_4 \quad B_2$

<table>
<thead>
<tr>
<th>Build</th>
<th>Merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>B_8</td>
<td>B_4</td>
</tr>
<tr>
<td>B_2</td>
<td>B_2</td>
</tr>
</tbody>
</table>

- $x, x'$ sorted (ascending)
- $y, y'$ sorted (descending)
- $b \leq c$
- $b' \geq c'$
- $b, c, b', c'$ bitonic
- $d$ sorted (ascending)
- $d'$ sorted (descending)
- $e$ bitonic
- $f \leq f'$
- $g \leq h \leq g' \leq h'$
- $f, f'$
- $g, h, g', h'$
- $i$ sorted bitonic
Bitonic sort $n = 8$
Bitonic Sort, $n = 16$
BitonicSort[16] is sequentially formed from $\oplus, \ominus$ variants \{BM[2], BM[4], BM[8], BM[16]\} with parallel sections (see top of previous figure).

BitonicSort[n] makes a sorted sequence of length $n$ from two sorted sequences of length $n/2$, (one ascending, one descending) using BM[n].

$BM[n]$ has $\log_2 n$ comparator stages. e.g. the 3 = log(8) comparator stages to form sequence $i$ from $d$ and $d'$ using BM[8], (see previous figures).
The number of comparator stages
\[ T(BS[n]) = T(BM[n]) + T(BS[n/2]) \]
where \( BS[n] = \text{BitonicSort}[n] \)

The number of comparator stages \( T(n) \) of the entire sorting network is given by:
\[ T(n) = \log(n) + T(n/2) \]

The solution of this recurrence equation is
\[
T(n) = \log(n) + \log(n) - 1 + \log(n) - 2 + ... + 1
= \log(n)(\log(n) + 1)/2
\]

Each stage of the sorting network consists of \( n/2 \) comparators. In total, there are \( \Theta(n\log^2(n)) \) comparators.
A final word

- The bitonic sort consists of the use bitonic merge $BM[n]$ in various ways, both for the build and merge phases.
- Bitonic merge $BM[n]$ is a divide and conquer algorithm based on $B_n$. Indeed $BM[n] = (B_n, \{BM[n/2], BM[n/2]\})$.
- The steps are build a bitonic sequence and then merge it.
Acknowledgements

With the permission of the author (H. W. Lang) these slides are based on material from the website http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/algoen.htm. We recommend you read http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/networks/indexen.htm. Other figures from https://en.wikipedia.org/wiki/Sorting_network and [CLRS]