I. Sorting Networks

Thomas Sauerwald

Easter 2015

UNIVERSITY OF CAMBRIDGE

Outline

Outline of this Course

Introduction to Sorting Networks

Batcher's Sorting Network

Counting Networks

(Tentative) List of Topics

Algorithms (I, II) → Complexity Theory → Advanced Algorithms

(Tentative) List of Topics

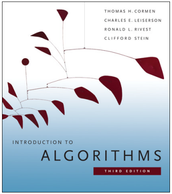
Algorithms (I, II) → Complexity Theory → Advanced Algorithms

- I. Sorting Networks (Sorting, Counting, Load Balancing) → 2
- II. Matrix Multiplication (Serial and Parallel)
- III. Linear Programming (Formulating, Applying and Solving) → 2
- IV. Approximation Algorithms: Covering Problems
- V. Approximation Algorithms via Exact Algorithms
- VI. Approximation Algorithms: Travelling Salesman Problem
- VII. Approximation Algorithms: Randomisation and Rounding
- VIII. Approximation Algorithms: MAX-CUT Problem

(Tentative) List of Topics

Algorithms (I, II) → Complexity Theory → Advanced Algorithms

- I. Sorting Networks (Sorting, Counting, Load Balancing)
- II. Matrix Multiplication (Serial and Parallel)
- III. Linear Programming (Formulating, Applying and Solving)
- IV. Approximation Algorithms: Covering Problems
- V. Approximation Algorithms via Exact Algorithms
- VI. Approximation Algorithms: Travelling Salesman Problem
- VII. Approximation Algorithms: Randomisation and Rounding
- VIII. Approximation Algorithms: MAX-CUT Problem

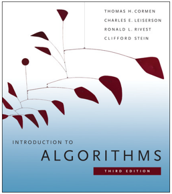


Closely follow the book and use the same numberring of theorems/lemmas etc.

(Tentative) List of Topics

Algorithms (I, II) → Complexity Theory → Advanced Algorithms

- I. Sorting Networks (Sorting, Counting, Load Balancing)
- II. Matrix Multiplication (Serial and Parallel)
- III. Linear Programming (Formulating, Applying and Solving)
- IV. Approximation Algorithms: Covering Problems
- V. Approximation Algorithms via Exact Algorithms
- VI. Approximation Algorithms: Travelling Salesman Problem
- VII. Approximation Algorithms: Randomisation and Rounding
- VIII. Approximation Algorithms: MAX-CUT Problem



Closely follow the book and use the same numberring of theorems/lemmas etc.

Outline

Outline of this Course

Introduction to Sorting Networks

Batcher's Sorting Network

Counting Networks

Overview: Sorting Networks

(Serial) Sorting Algorithms

- we already know several (comparison-based) sorting algorithms: Insertion sort, Bubble sort, Merge sort, Quick sort, Heap sort
- execute one operation at a time
- can handle arbitrarily large inputs
- sequence of comparisons is not set in advance

Overview: Sorting Networks

(Serial) Sorting Algorithms

- we already know several (comparison-based) sorting algorithms: Insertion sort, Bubble sort, Merge sort, Quick sort, Heap sort
- execute one operation at a time
- can handle arbitrarily large inputs
- sequence of comparisons is not set in advance

Sorting Networks

- only perform comparisons
- can only handle inputs of a fixed size
- sequence of comparisons is set in advance

Overview: Sorting Networks

(Serial) Sorting Algorithms

- we already know several (comparison-based) sorting algorithms: Insertion sort, Bubble sort, Merge sort, Quick sort, Heap sort
- execute one operation at a time
- can handle arbitrarily large inputs
- sequence of comparisons is not set in advance

Sorting Networks

- only perform comparisons
- can only handle inputs of a fixed size
- sequence of comparisons is set in advance
- Comparisons can be performed in parallel

Allows to sort n numbers in sublinear time!

Overview: Sorting Networks

(Serial) Sorting Algorithms

- we already know several (comparison-based) sorting algorithms: Insertion sort, Bubble sort, Merge sort, Quick sort, Heap sort
- execute one operation at a time
- can handle arbitrarily large inputs
- sequence of comparisons is not set in advance

Sorting Networks

- only perform comparisons
- can only handle inputs of a fixed size
- sequence of comparisons is set in advance
- Comparisons can be performed in parallel

Allows to sort n numbers in sublinear time!

Simple concept, but surprisingly deep and complex theory!

Comparison Networks

Comparison Network

- A comparison network consists solely of wires and comparators:

Comparison Network

- A comparison network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs x' and y'

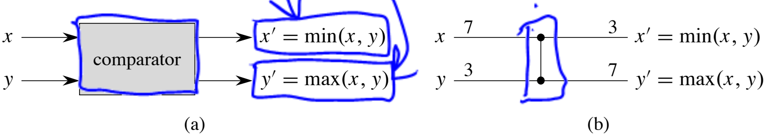


Figure 27.1 **(a)** A comparator with inputs x and y and outputs x' and y'. **(b)** The same comparator, drawn as a single vertical line. Inputs $x = 7$, $y = 3$ and outputs $x' = 3$, $y' = 7$ are shown.

Comparison Networks

Comparison Network

- A comparison network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs x' and y'

operates in $O(1)$

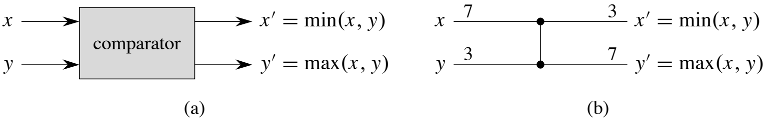


Figure 27.1 **(a)** A comparator with inputs x and y and outputs x' and y'. **(b)** The same comparator, drawn as a single vertical line. Inputs $x = 7$, $y = 3$ and outputs $x' = 3$, $y' = 7$ are shown.

Comparison Networks

Comparison Network

- A comparison network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs x' and y'
  - wire connect output of one comparator to the input of another

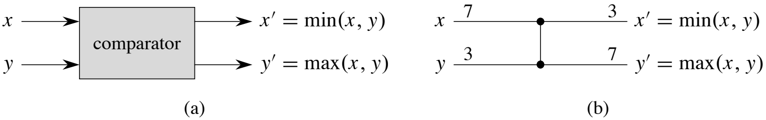


Figure 27.1 **(a)** A comparator with inputs x and y and outputs x' and y'. **(b)** The same comparator, drawn as a single vertical line. Inputs $x = 7$, $y = 3$ and outputs $x' = 3$, $y' = 7$ are shown.

Comparison Networks

Comparison Network

- A comparison network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs x' and y'
  - wire connect output of one comparator to the input of another
  - special wires: n input wires $a_1, a_2, \ldots, a_n$ and n output wires $b_1, b_2, \ldots, b_n$

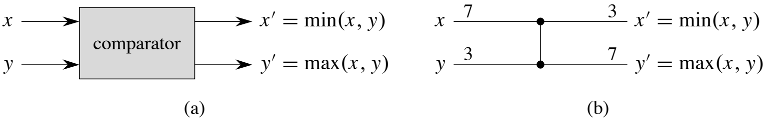


Figure 27.1 **(a)** A comparator with inputs x and y and outputs x' and y'. **(b)** The same comparator, drawn as a single vertical line. Inputs $x = 7$, $y = 3$ and outputs $x' = 3$, $y' = 7$ are shown.

Comparison Networks

Comparison Network

- A comparison network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs x' and y'
  - wire connect output of one comparator to the input of another
  - special wires: n input wires $a_1, a_2, \ldots, a_n$ and n output wires $b_1, b_2, \ldots, b_n$

Convention: use the same name for both a wire and its value.

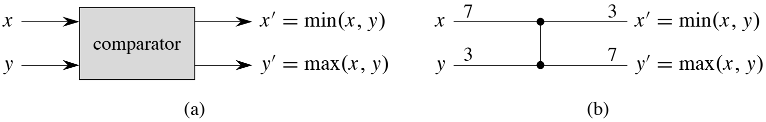


Figure 27.1 **(a)** A comparator with inputs x and y and outputs x' and y'. **(b)** The same comparator, drawn as a single vertical line. Inputs $x = 7$, $y = 3$ and outputs $x' = 3$, $y' = 7$ are shown.

Comparison Networks

A sorting network is a comparison network which works correctly (that is, it sorts every input)

Comparison Network

- A comparison network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs x' and y'
  - wire connect output of one comparator to the input of another
  - special wires: n input wires $a_1, a_2, \ldots, a_n$ and n output wires $b_1, b_2, \ldots, b_n$

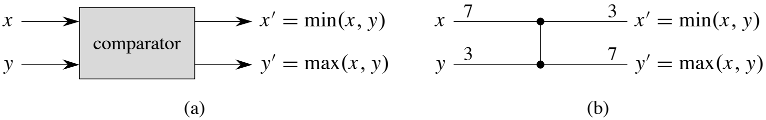
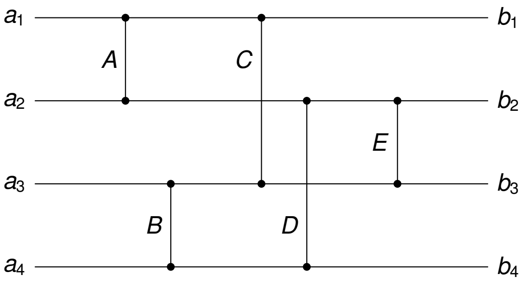


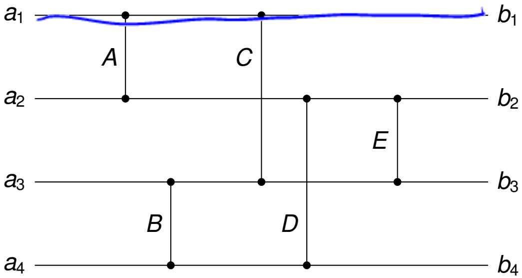
Figure 27.1 **(a)** A comparator with inputs x and y and outputs x' and y'. **(b)** The same comparator, drawn as a single vertical line. Inputs $x = 7$, $y = 3$ and outputs $x' = 3$, $y' = 7$ are shown.

Example of a Comparison Network (Figure 27.2)



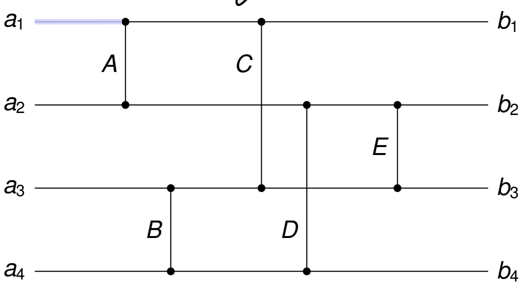
Example of a Comparison Network (Figure 27.2)

A horizontal line represents a sequence of distinct wires

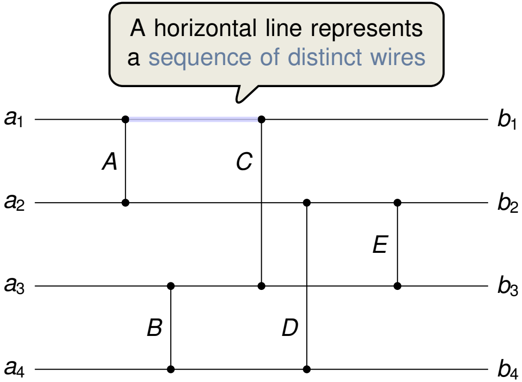


Example of a Comparison Network (Figure 27.2)

A horizontal line represents
a sequence of distinct wires

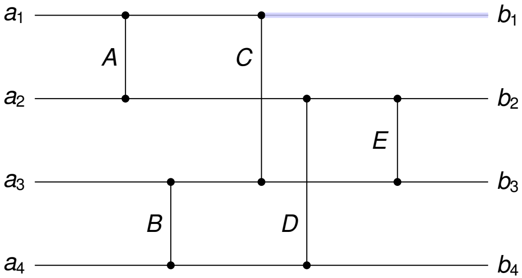


Example of a Comparison Network (Figure 27.2)

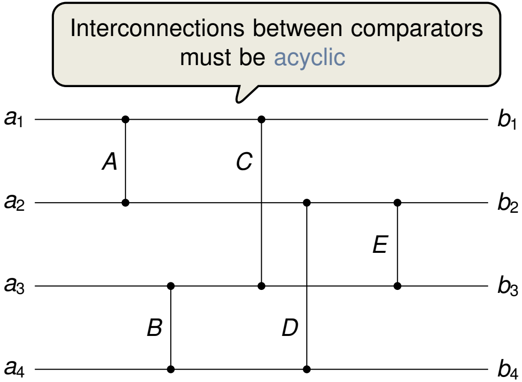


Example of a Comparison Network (Figure 27.2)

A horizontal line represents
a sequence of distinct wires

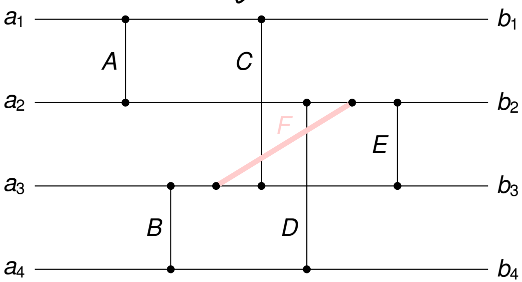


Example of a Comparison Network (Figure 27.2)



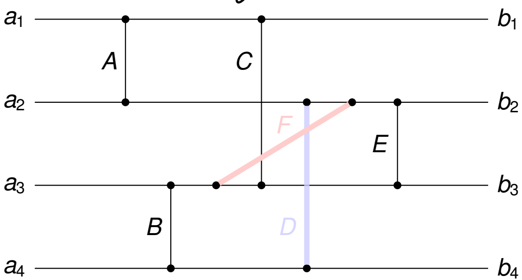
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic

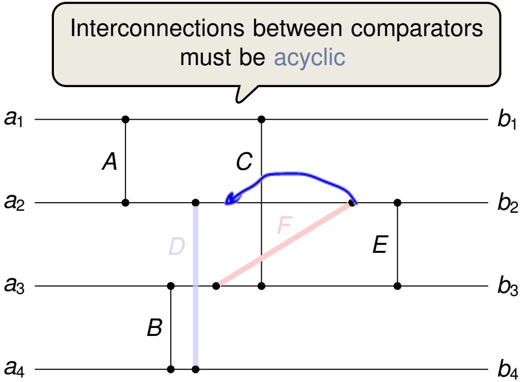


Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic

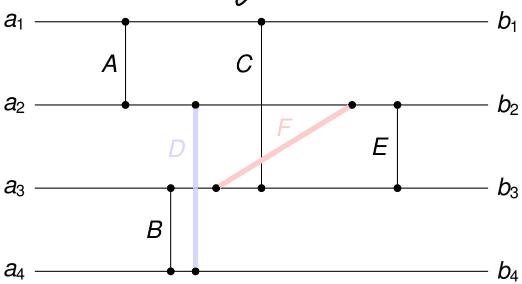


Example of a Comparison Network (Figure 27.2)



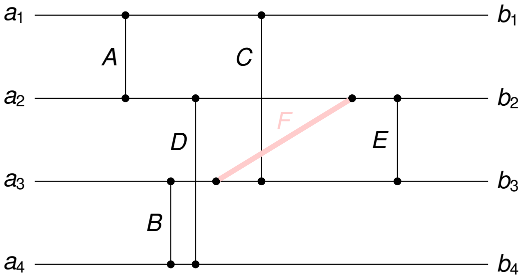
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic



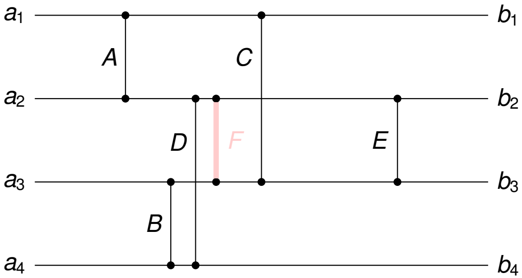
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic



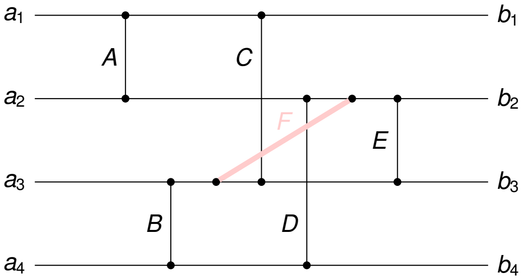
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic

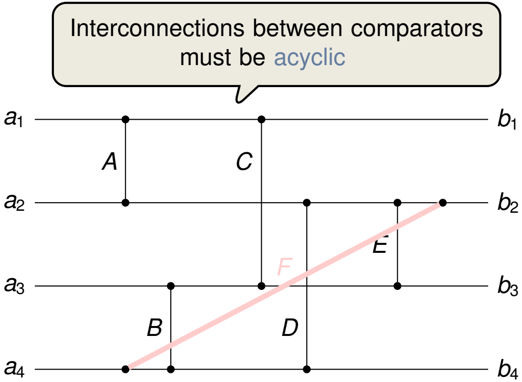


Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic ✓

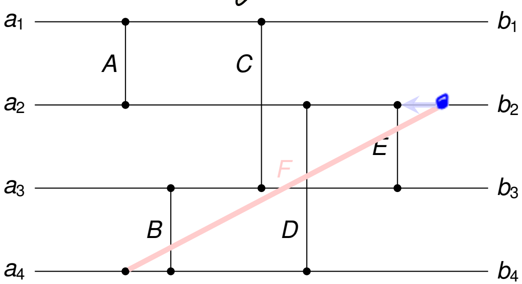


Example of a Comparison Network (Figure 27.2)



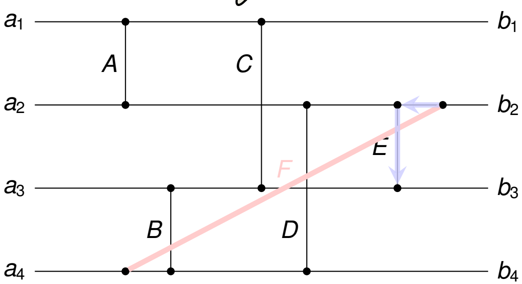
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic



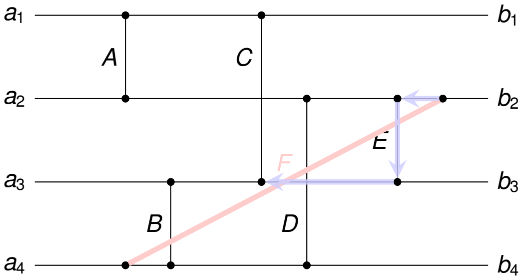
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic



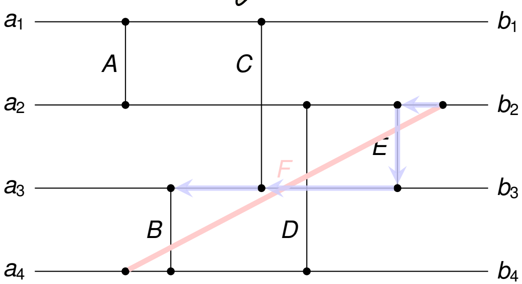
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic



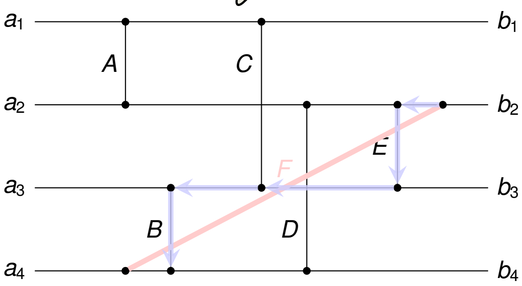
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic



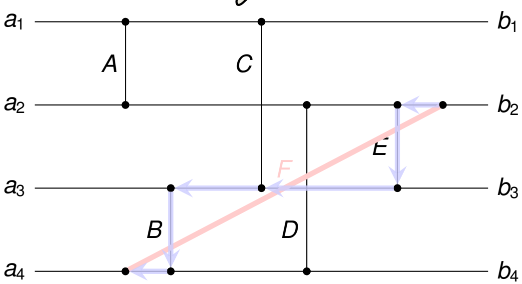
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic



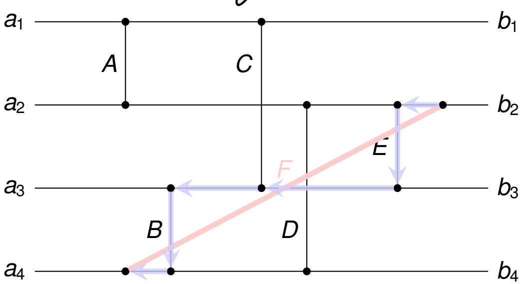
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic



Example of a Comparison Network (Figure 27.2)

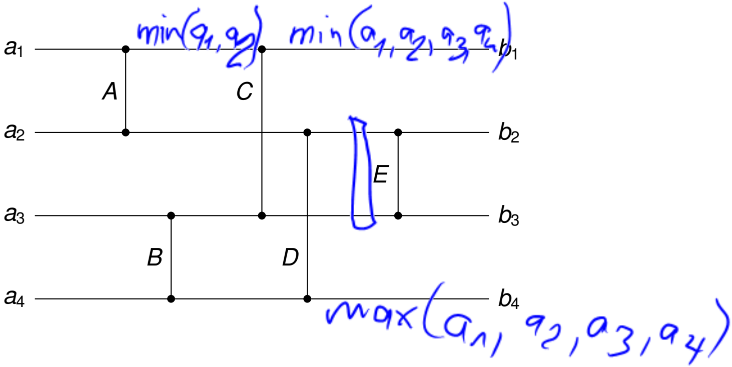
Interconnections between comparators must be acyclic



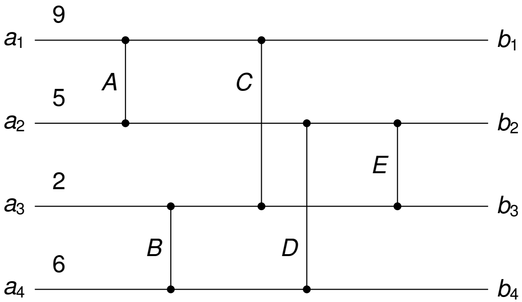
Tracing back a path must never cycle back on itself and go through the same comparator twice.

Example of a Comparison Network (Figure 27.2)

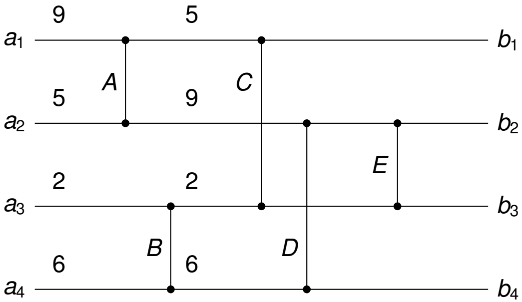
"Proof" that it is a Sorting Network:



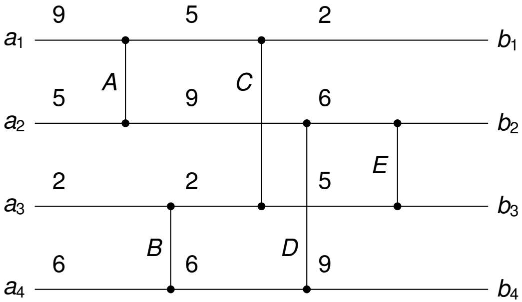
Example of a Comparison Network (Figure 27.2)



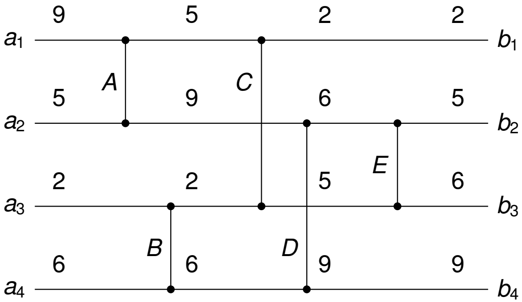
Example of a Comparison Network (Figure 27.2)



Example of a Comparison Network (Figure 27.2)

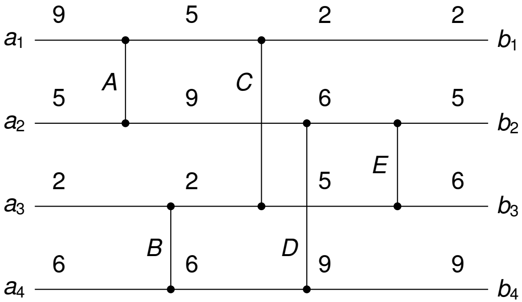


Example of a Comparison Network (Figure 27.2)



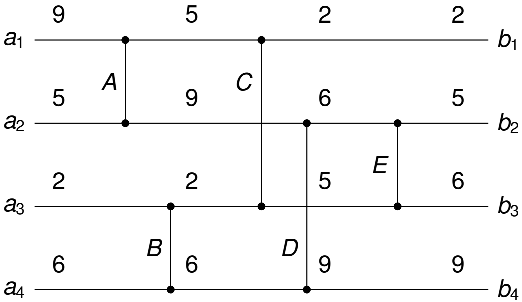
This network is in fact a sorting network!

Example of a Comparison Network (Figure 27.2)



Depth of a wire:

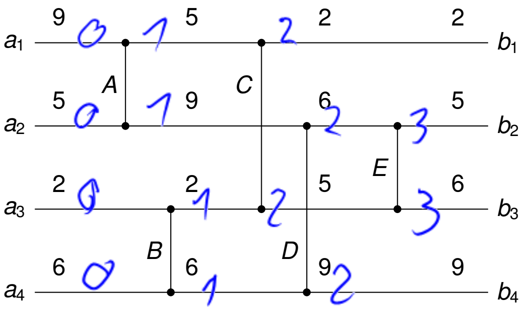
Example of a Comparison Network (Figure 27.2)



Depth of a wire:

- Input wire has depth 0

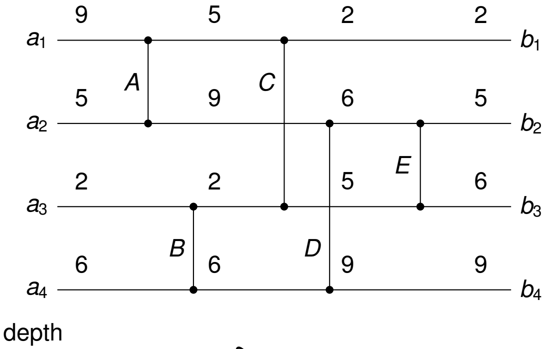
Example of a Comparison Network (Figure 27.2)



Depth of a wire:

- Input wire has depth 0
- If a comparator has two inputs of depths d_x and d_y, then outputs have depth $\max\{d_x, d_y\} + 1$

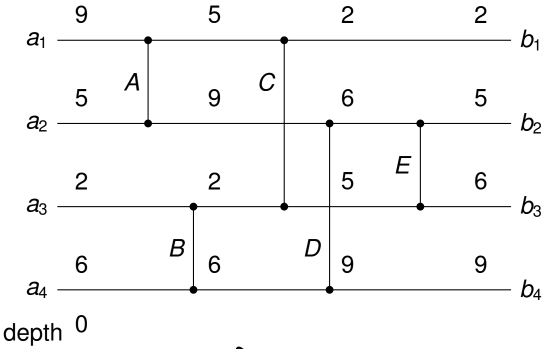
Example of a Comparison Network (Figure 27.2)



Depth of a wire:

- Input wire has depth 0
- If a comparator has two inputs of depths d_x and d_y, then outputs have depth $\max\{d_x, d_y\} + 1$

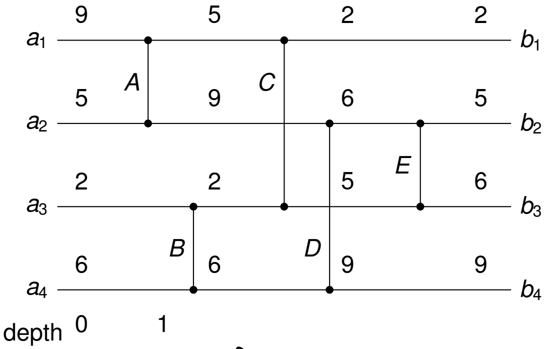
Example of a Comparison Network (Figure 27.2)



Depth of a wire:

- Input wire has depth 0
- If a comparator has two inputs of depths d_x and d_y, then outputs have depth $\max\{d_x, d_y\} + 1$

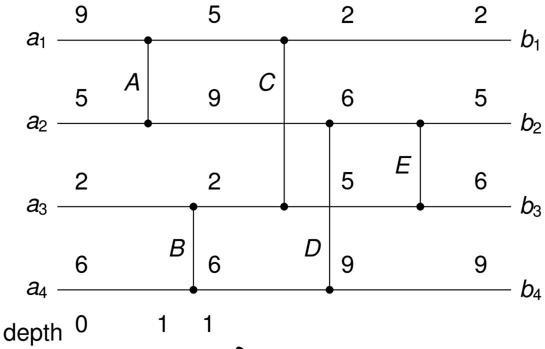
Example of a Comparison Network (Figure 27.2)



Depth of a wire:

- Input wire has depth 0
- If a comparator has two inputs of depths d_x and d_y, then outputs have depth $\max\{d_x, d_y\} + 1$

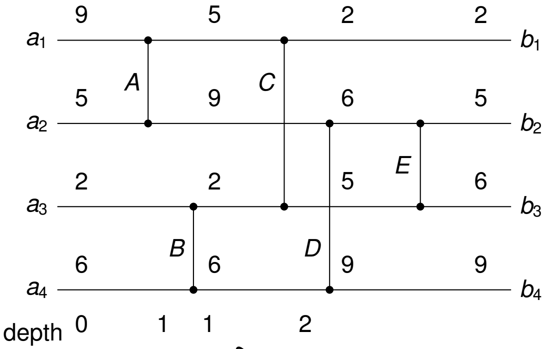
Example of a Comparison Network (Figure 27.2)



Depth of a wire:

- Input wire has depth 0
- If a comparator has two inputs of depths d_x and d_y, then outputs have depth $\max\{d_x, d_y\} + 1$

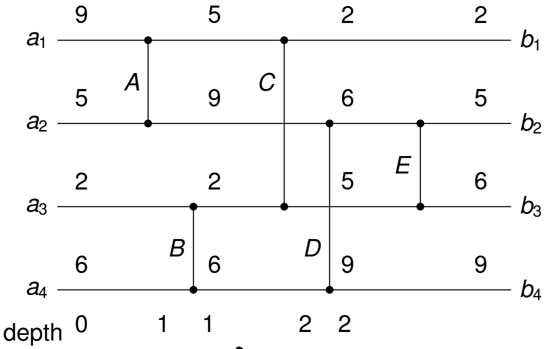
Example of a Comparison Network (Figure 27.2)



Depth of a wire:

- Input wire has depth 0
- If a comparator has two inputs of depths d_x and d_y, then outputs have depth $\max\{d_x, d_y\} + 1$

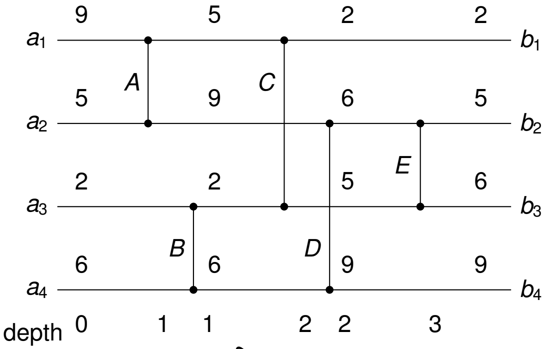
Example of a Comparison Network (Figure 27.2)



Depth of a wire:

- Input wire has depth 0
- If a comparator has two inputs of depths d_x and d_y, then outputs have depth $\max\{d_x, d_y\} + 1$

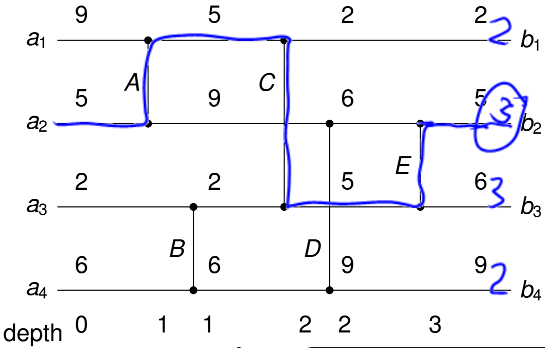
Example of a Comparison Network (Figure 27.2)



Depth of a wire:

- Input wire has depth 0
- If a comparator has two inputs of depths d_x and d_y, then outputs have depth $\max\{d_x, d_y\} + 1$

Example of a Comparison Network (Figure 27.2)



Depth of a wire:

- Input wire has depth 0
- If a comparator has two inputs of depths d_x and d_y, then outputs have depth $\max\{d_x, d_y\} + 1$

Maximum depth of an output wire equals total running time

Zero-One Principle

Zero-One Principle: A sorting networks works correctly on arbitrary inputs if it works correctly on binary inputs.

Zero-One Principle

Zero-One Principle: A sorting networks works correctly on arbitrary inputs if it works correctly on binary inputs.

Lemma 27.1

If a comparison network transforms the input $a = \langle a_1, a_2, \ldots, a_n \rangle$ into the output $b = \langle b_1, b_2, \ldots, b_n \rangle$, then for any monotonically increasing function f, the network transforms $f(a) = \langle f(a_1), f(a_2), \ldots, f(a_n) \rangle$ into $f(b) = \langle f(b_1), f(b_2), \ldots, f(b_n) \rangle$.

$$
\begin{tikzcd}
a \arrow[r, "C"] \arrow[d, "f"'] & b \arrow[d, "f"] \\
f(a) \arrow[r, "C"] & f(b)
\end{tikzcd}
$$

Zero-One Principle

Zero-One Principle: A sorting networks works correctly on arbitrary inputs if it works correctly on binary inputs.

Lemma 27.1

If a comparison network transforms the input $a = \langle a_1, a_2, \ldots, a_n \rangle$ into the output $b = \langle b_1, b_2, \ldots, b_n \rangle$, then for any monotonically increasing function f, the network transforms $f(a) = \langle f(a_1), f(a_2), \ldots, f(a_n) \rangle$ into $f(b) = \langle f(b_1), f(b_2), \ldots, f(b_n) \rangle$.

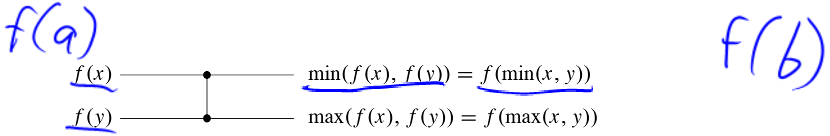


Figure 27.4 The operation of the comparator in the proof of Lemma 27.1. The function f is monotonically increasing.

Zero-One Principle

Zero-One Principle: A sorting networks works correctly on arbitrary inputs if it works correctly on binary inputs.

Lemma 27.1

If a comparison network transforms the input $a = \langle a_1, a_2, \ldots, a_n \rangle$ into the output $b = \langle b_1, b_2, \ldots, b_n \rangle$, then for any monotonically increasing function f, the network transforms $f(a) = \langle f(a_1), f(a_2), \ldots, f(a_n) \rangle$ into $f(b) = \langle f(b_1), f(b_2), \ldots, f(b_n) \rangle$.

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correctly.

Proof of the Zero-One Principle

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correctly.

Proof of the Zero-One Principle

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correctly.

Proof:

Proof of the Zero-One Principle

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correctly.

Proof:

- For the sake of contradiction, suppose the network does not correctly sort.

Proof of the Zero-One Principle

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correctly.

Proof:

$i < j, \; i > j$

- For the sake of contradiction, suppose the network does not correctly sort.
- Let $a = \langle a_1, a_2, \ldots, a_n \rangle$ be the input with $\underline{a_i} < \underline{a_j}$, but the network places a_j before a_i in the output

Proof of the Zero-One Principle

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correctly.

Proof:

- For the sake of contradiction, suppose the network does not correctly sort.
- Let $a = \langle a_1, a_2, \ldots, a_n \rangle$ be the input with $a_i < a_j$, but the network places a_j before a_i in the output
- Define a monotonically increasing function f as:

Proof of the Zero-One Principle

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correctly.

Proof:

- For the sake of contradiction, suppose the network does not correctly sort.
- Let $a = \langle a_1, a_2, \ldots, a_n \rangle$ be the input with $a_i < a_j$, but the network places a_j before a_i in the output
- Define a monotonically increasing function f as:

$$f(x) = \begin{cases} 0 & \text{if } x \leq a_i, \\ 1 & \text{if } x > a_i. \end{cases}$$

Proof of the Zero-One Principle

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correctly.

Proof:

- For the sake of contradiction, suppose the network does not correctly sort.
- Let $a = \langle a_1, a_2, \ldots, a_n \rangle$ be the input with $a_i < a_j$, but the network places a_j before a_i in the output
- Define a monotonically increasing function f as:

$$f(x) = \begin{cases} 0 & \text{if } x \leq a_i, \\ 1 & \text{if } x > a_i. \end{cases}$$

- Since the network places a_j before a_i, by the previous lemma

Proof of the Zero-One Principle

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correctly.

Proof:

- For the sake of contradiction, suppose the network does not correctly sort.
- Let $a = \langle a_1, a_2, \ldots, a_n \rangle$ be the input with $a_i < a_j$, but the network places a_j before a_i in the output
- Define a monotonically increasing function f as:

$$f(x) = \begin{cases} 0 & \text{if } x \leq a_i, \\ 1 & \text{if } x > a_i. \end{cases}$$

- Since the network places a_j before a_i, by the previous lemma $\Rightarrow$ $\underbrace{f(a_j)}_{=1}$ is placed before $\underbrace{f(a_i)}_{=0}$

Proof of the Zero-One Principle

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correctly.

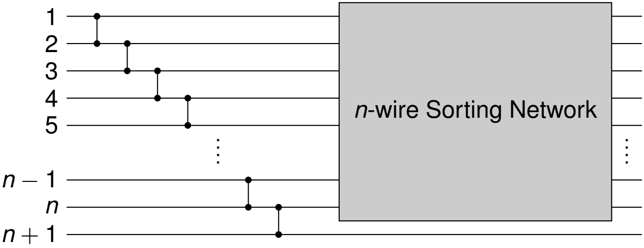
Proof:

- For the sake of contradiction, suppose the network does not correctly sort.
- Let $a = \langle a_1, a_2, \ldots, a_n \rangle$ be the input with $a_i < a_j$, but the network places a_j before a_i in the output
- Define a monotonically increasing function f as:

$$f(x) = \begin{cases} 0 & \text{if } x \leq a_i, \\ 1 & \text{if } x > a_i. \end{cases}$$

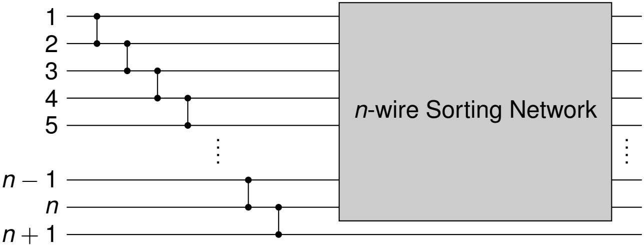
- Since the network places a_j before a_i, by the previous lemma $\Rightarrow$ $f(a_j)$ is placed before $f(a_i)$
- But $f(a_j) = 1$ and $f(a_i) = 0$, which contradicts the assumption that the network sorts all sequences of 0's and 1's correctly $\square$

Some Basic (Recursive) Sorting Networks



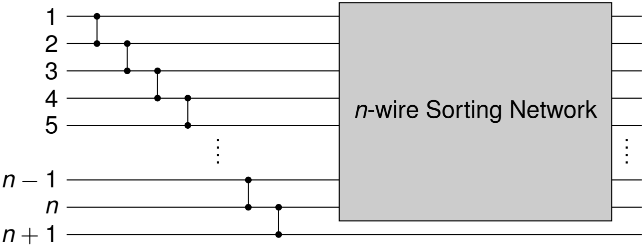
???

Some Basic (Recursive) Sorting Networks

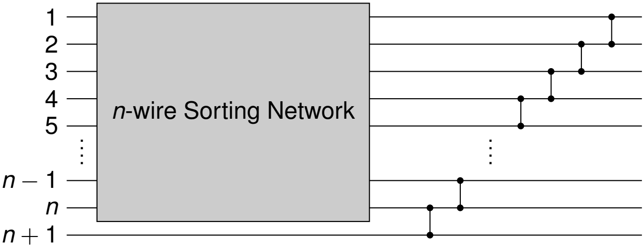


Bubble Sort

Some Basic (Recursive) Sorting Networks

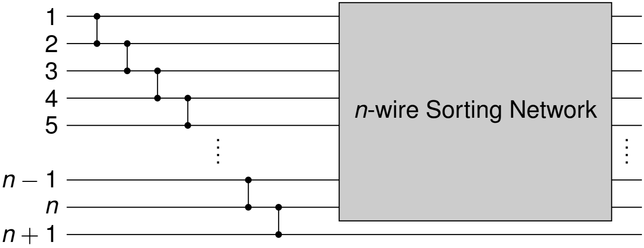


Bubble Sort

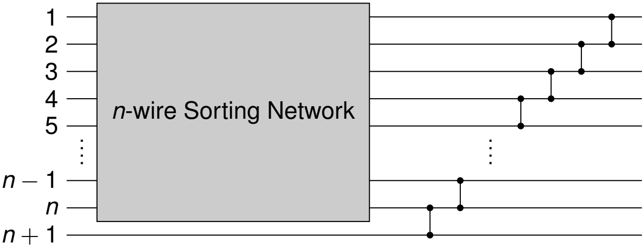


???

Some Basic (Recursive) Sorting Networks

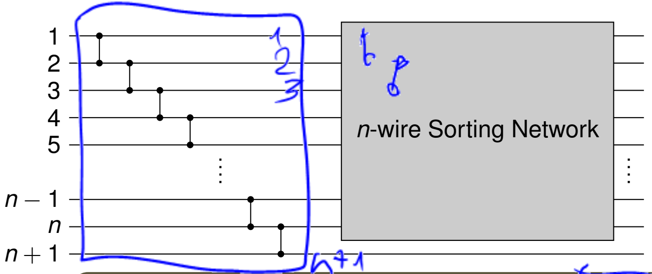


Bubble Sort



Insertion Sort

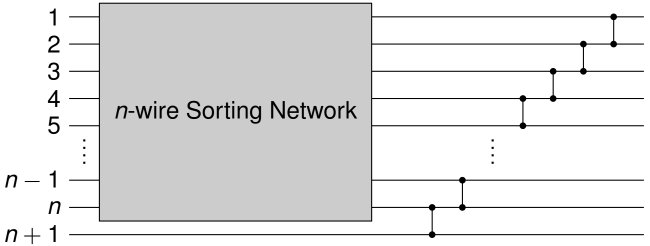
Some Basic (Recursive) Sorting Networks



Bubble Sort

$T(n) = n + T(n-1)$

These are Sorting Networks, but with depth $\boxed{\Theta(n)}$.



Insertion Sort

Outline

Outline of this Course

Introduction to Sorting Networks

Batcher's Sorting Network

Counting Networks

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Sequences of one or two numbers are defined to be bitonic.

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples:

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples:

- $\langle 1, 4, 6, 8, 3, 2 \rangle$?

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples:

- $\langle 1, 4, 6, 8, 3, 2 \rangle$ ✓

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples:

- $\langle 1, 4, 6, 8, 3, 2 \rangle$ ✓
- $\langle 6, 9, 4, 2, 3, 5 \rangle$?

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples:

- $\langle 1, 4, 6, 8, 3, 2 \rangle$ ✓
- $\langle 6, 9, 4, 2, 3, 5 \rangle$ ✓

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples:

- $\langle 1, 4, 6, 8, 3, 2 \rangle$ ✓
- $\langle 6, 9, 4, 2, 3, 5 \rangle$ ✓
- $\langle 9, 8, 3, 2, 4, 6 \rangle$?

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples:

- $\langle 1, 4, 6, 8, 3, 2 \rangle$ ✓
- $\langle 6, 9, 4, 2, 3, 5 \rangle$ ✓
- $\langle 9, 8, 3, 2, 4, 6 \rangle$ ✓

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples:

- $\langle 1,4,6,8,3,2\rangle$ ✓
- $\langle 6,9,4,2,3,5\rangle$ ✓
- $\langle 9,8,3,2,4,6\rangle$ ✓
- $\langle 4,5,7,1,2,6\rangle$?

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples:

- $\langle 1, 4, 6, 8, 3, 2 \rangle$ ✓
- $\langle 6, 9, 4, 2, 3, 5 \rangle$ ✓
- $\langle 9, 8, 3, 2, 4, 6 \rangle$ ✓
- $\cancel{\langle 4, 5, 7, 1, 2, 6 \rangle}$ 3

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples:

- $\langle 1, 4, 6, 8, 3, 2 \rangle$ ✓
- $\langle 6, 9, 4, 2, 3, 5 \rangle$ ✓
- $\langle 9, 8, 3, 2, 4, 6 \rangle$ ✓
- $\cancel{\langle 4, 5, 7, 1, 2, 6 \rangle}$
- binary sequences: ?

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples:

- $\langle 1, 4, 6, 8, 3, 2 \rangle$ ✓
- $\langle 6, 9, 4, 2, 3, 5 \rangle$ ✓
- $\langle 9, 8, 3, 2, 4, 6 \rangle$ ✓
- $\cancel{\langle 4, 5, 7, 1, 2, 6 \rangle}$
- binary sequences: $0^i 1^j 0^k$, or, $1^i 0^j 1^k$, for $i, j, k \geq 0$.

Towards Bitonic Sorting Networks

Half-Cleaner

A half-cleaner is a comparison network of depth 1 in which input wire i is compared with wire $i + n/2$ for $i = 1, 2, \ldots, n/2$.

Towards Bitonic Sorting Networks

Half-Cleaner

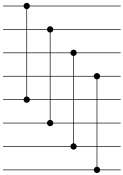
A half-cleaner is a comparison network of depth 1 in which input wire i is compared with wire $i + n/2$ for $i = 1, 2, \ldots, \boxed{n/2.}$

We always assume that n is even.

Towards Bitonic Sorting Networks

Half-Cleaner

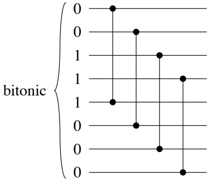
A half-cleaner is a comparison network of depth 1 in which input wire i is compared with wire $i + n/2$ for $i = 1, 2, \ldots, n/2$.



Towards Bitonic Sorting Networks

Half-Cleaner

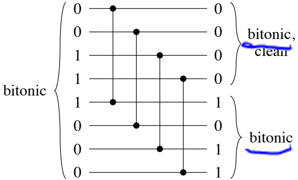
A half-cleaner is a comparison network of depth 1 in which input wire i is compared with wire $i + n/2$ for $i = 1, 2, \ldots, n/2$.



Towards Bitonic Sorting Networks

Half-Cleaner

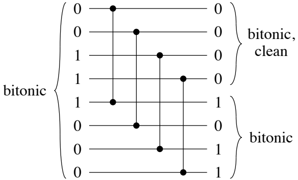
A half-cleaner is a comparison network of depth 1 in which input wire i is compared with wire $i + n/2$ for $i = 1, 2, \ldots, n/2$.

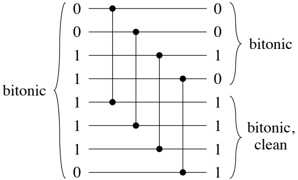


Towards Bitonic Sorting Networks

Half-Cleaner

A half-cleaner is a comparison network of depth 1 in which input wire i is compared with wire $i + n/2$ for $i = 1, 2, \ldots, n/2$.





Towards Bitonic Sorting Networks

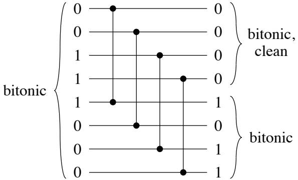
Half-Cleaner

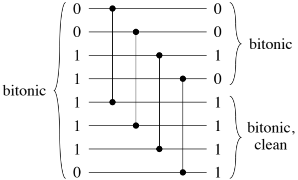
A half-cleaner is a comparison network of depth 1 in which input wire i is compared with wire $i + n/2$ for $i = 1, 2, \ldots, n/2$.

Lemma 27.3

If the input to a half-cleaner is a bitonic sequence of 0's and 1's, then the output satisfies the following properties:

- both the top half and the bottom half are bitonic,
- every element in the top is not larger than any element in the bottom,
- at least one half is clean.





Towards Bitonic Sorting Networks

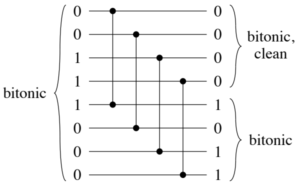
Half-Cleaner

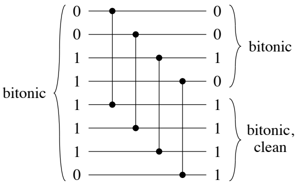
A half-cleaner is a comparison network of depth 1 in which input wire i is compared with wire $i + n/2$ for $i = 1, 2, \ldots, n/2$.

Lemma 27.3

If the input to a half-cleaner is a bitonic sequence of 0's and 1's, then the output satisfies the following properties:

- both the top half and the bottom half are bitonic,
- every element in the top is not larger than any element in the bottom,
- at least one half is clean.



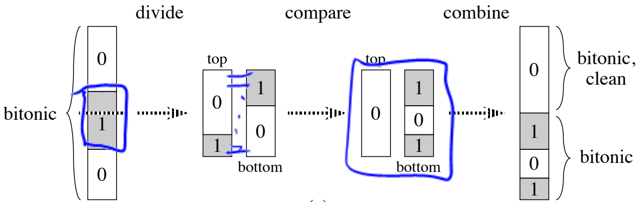


Proof of Lemma 27.3

W.l.o.g. assume that the input is of the form $0^i 1^j 0^k$, for some $i, j, k \geq 0$.

Proof of Lemma 27.3

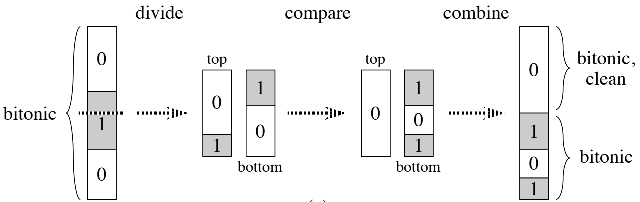
W.l.o.g. assume that the input is of the form $0^i 1^j 0^k$, for some $i, j, k \geq 0$.



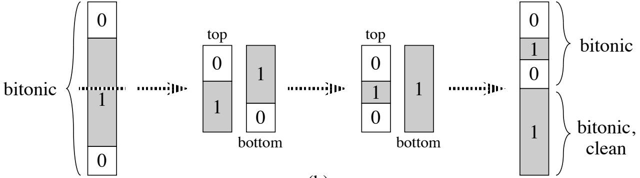
(a)

Proof of Lemma 27.3

W.l.o.g. assume that the input is of the form $0^i 1^j 0^k$, for some $i, j, k \geq 0$.



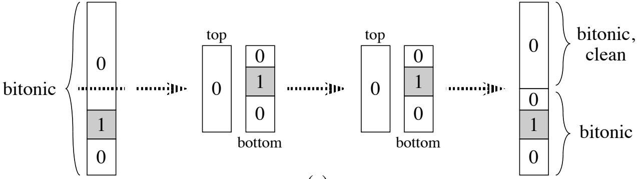
(a)



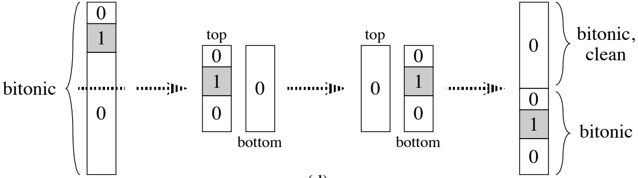
(b)

Proof of Lemma 27.3

W.l.o.g. assume that the input is of the form $0^i 1^j 0^k$, for some $i, j, k \geq 0$.



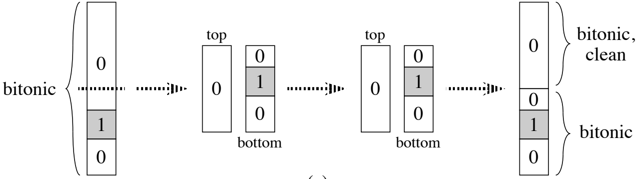
(c)



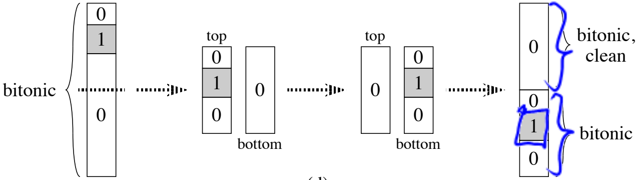
(d)

Proof of Lemma 27.3

W.l.o.g. assume that the input is of the form $0^i 1^j 0^k$, for some $i, j, k \geq 0$.



(c)



(d)

This suggests a recursive approach, since it now suffices to sort the top and bottom half separately.

The Bitonic Sorter

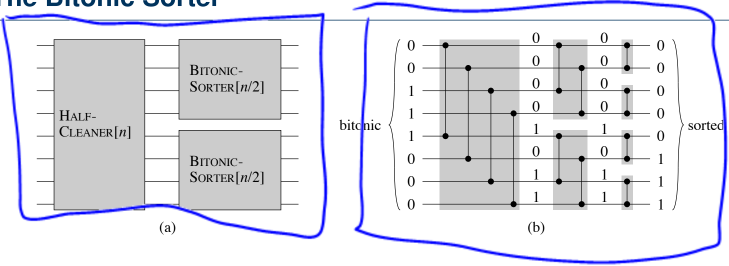


Figure 27.9 The comparison network BITONIC-SORTER[n], shown here for $n = 8$. **(a)** The recursive construction: HALF-CLEANER[n] followed by two copies of BITONIC-SORTER[$n/2$] that operate in parallel. **(b)** The network after unrolling the recursion. Each half-cleaner is shaded. Sample zero-one values are shown on the wires.

The Bitonic Sorter

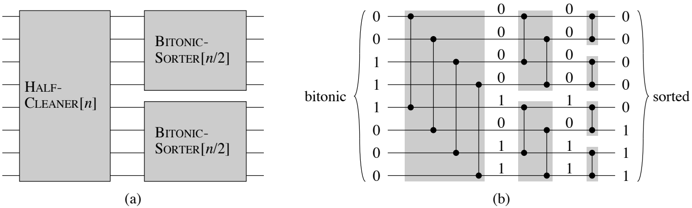


Figure 27.9 The comparison network BITONIC-SORTER[n], shown here for $n = 8$. **(a)** The recursive construction: HALF-CLEANER[n] followed by two copies of BITONIC-SORTER[$n/2$] that operate in parallel. **(b)** The network after unrolling the recursion. Each half-cleaner is shaded. Sample zero-one values are shown on the wires.

Recursive Formula for depth $D(n)$:

$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ D(n/2) + 1 & \text{if } n = 2^k. \end{cases}$$

The Bitonic Sorter

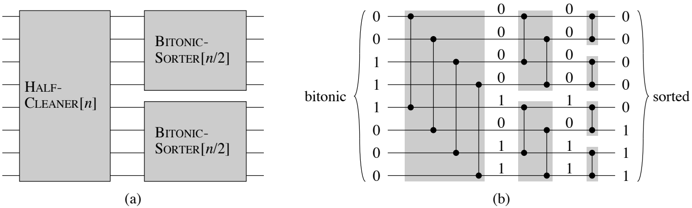


Figure 27.9 The comparison network BITONIC-SORTER[n], shown here for $n = 8$. **(a)** The recursive construction: HALF-CLEANER[n] followed by two copies of BITONIC-SORTER[$n/2$] that operate in parallel. **(b)** The network after unrolling the recursion. Each half-cleaner is shaded. Sample zero-one values are shown on the wires.

Henceforth we will always assume that n is a power of 2.

Recursive Formula for depth $D(n)$:

$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ D(n/2) + 1 & \text{if } n = 2^k. \end{cases}$$

The Bitonic Sorter

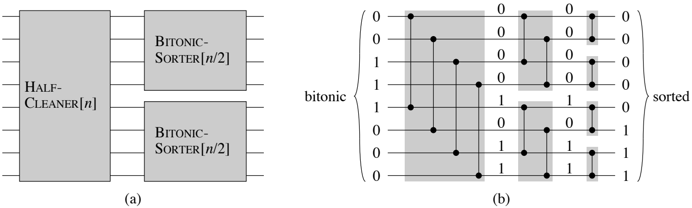


Figure 27.9 The comparison network BITONIC-SORTER[n], shown here for $n = 8$. **(a)** The recursive construction: HALF-CLEANER[n] followed by two copies of BITONIC-SORTER[$n/2$] that operate in parallel. **(b)** The network after unrolling the recursion. Each half-cleaner is shaded. Sample zero-one values are shown on the wires.

Henceforth we will always assume that n is a power of 2.

Recursive Formula for depth $D(n)$:

$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ D(n/2) + 1 & \text{if } n = 2^k. \end{cases}$$

BITONIC-SORTER[n] has depth $\log n$ and sorts any zero-one bitonic sequence.

Merging Networks

Merging Networks

- can merge two sorted input sequences into one sorted output sequences
- will be based on a modification of BITONIC-SORTER[n]

Merging Networks

Merging Networks

- can merge two sorted input sequences into one sorted output sequences
- will be based on a modification of BITONIC-SORTER[n]

Basic Idea:

Merging Networks

Merging Networks

- can merge two sorted input sequences into one sorted output sequences
- will be based on a modification of BITONIC-SORTER[n]

Basic Idea:

- consider two given sequences $X = 00000111$, $Y = 00001111$

Merging Networks

Merging Networks

- can merge two sorted input sequences into one sorted output sequences
- will be based on a modification of BITONIC-SORTER[n]

Basic Idea:

- consider two given sequences $X = 00000111$, $Y = 00001111$
- concatenating X with Y^R (the reversal of Y) $\Rightarrow 0000011111110000$

Merging Networks

Merging Networks

- can merge two sorted input sequences into one sorted output sequences
- will be based on a modification of BITONIC-SORTER[n]

Basic Idea:

- consider two given sequences $X = 00000111$, $Y = 00001111$
- concatenating X with Y^R (the reversal of Y) $\Rightarrow$ 0000011111110000

This sequence is bitonic!

Merging Networks

Merging Networks

- can merge two sorted input sequences into one sorted output sequences
- will be based on a modification of BITONIC-SORTER$[n]$

Basic Idea:

- consider two given sequences $X = 00000111$, $Y = 00001111$
- concatenating X with Y^R (the reversal of Y) $\Rightarrow$ 0000011111110000

This sequence is bitonic!

Hence in order to merge the sequences X and Y, it suffices to perform a bitonic sort on X concatenated with Y^R.

Construction of a Merging Network (1/2)

- Given two sorted sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$

Construction of a Merging Network (1/2)

- Given two sorted sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$
- We know it suffices to bitonically sort $\langle a_1, a_2, \ldots, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+1} \rangle$

Construction of a Merging Network (1/2)

- Given two sorted sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$
- We know it suffices to bitonically sort $\langle a_1, a_2, \ldots, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+1} \rangle$
- Recall: first half-cleaner of BITONIC-SORTER[n] compares i and $n/2 + i$

Construction of a Merging Network (1/2)

- Given two sorted sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$
- We know it suffices to bitonically sort $\langle a_1, a_2, \ldots, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+1} \rangle$
- Recall: first half-cleaner of BITONIC-SORTER[n] compares i and $n/2 + i$

$\Rightarrow$ First part of MERGER[n] compares inputs i and $n - i$ for $i = 1, 2, \ldots, n/2$

Construction of a Merging Network (1/2)

- Given two sorted sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$
- We know it suffices to bitonically sort $\langle a_1, a_2, \ldots, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+1} \rangle$
- Recall: first half-cleaner of BITONIC-SORTER[n] compares i and $n/2 + i$

$\Rightarrow$ First part of MERGER[n] compares inputs i and $n - i$ for $i = 1, 2, \ldots, n/2$

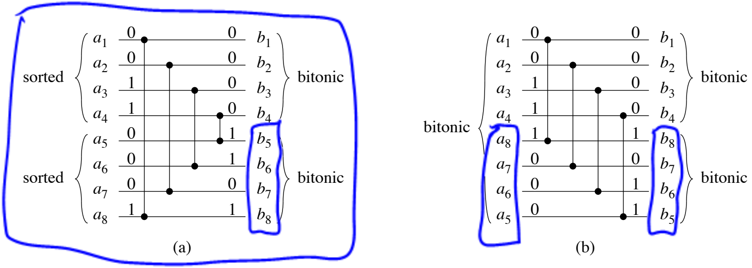
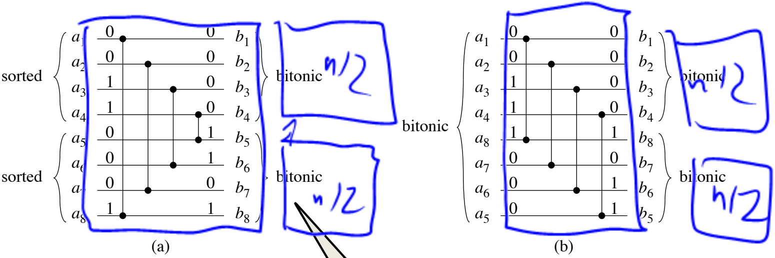


Figure 27.10 Comparing the first stage of MERGER[n] with HALF-CLEANER[n], for $n = 8$. **(a)** The first stage of MERGER[n] transforms the two monotonic input sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$ into two bitonic sequences $\langle b_1, b_2, \ldots, b_{n/2} \rangle$ and $\langle b_{n/2+1}, b_{n/2+2}, \ldots, b_n \rangle$. **(b)** The equivalent operation for HALF-CLEANER[n]. The bitonic input sequence $\langle a_1, a_2, \ldots, a_{n/2-1}, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+2}, a_{n/2+1} \rangle$ is transformed into the two bitonic sequences $\langle b_1, b_2, \ldots, b_{n/2} \rangle$ and $\langle b_n, b_{n-1}, \ldots, b_{n/2+1} \rangle$.

Construction of a Merging Network (1/2)

- Given two sorted sequences $\langle a_1, a_2, \ldots, a_{n/2}\rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n\rangle$
- We know it suffices to bitonically sort $\langle a_1, a_2, \ldots, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+1}\rangle$
- Recall: first half-cleaner of BITONIC-SORTER[n] compares i and $n/2+i$

$\Rightarrow$ First part of MERGER[n] compares inputs i and $n-i$ for $i = 1, 2, \ldots, n/2$



Lemma 27.3 still applies, since the reversal of a bitonic sequence is bitonic.

Figure 27.10 Comparing the first stage of MERGER[n] with HALF-CLEANER[n], for $n = 8$. **(a)** The first stage of MERGER[n] transforms the two monotonic input sequences $\langle a_1, a_2, \ldots, a_{n/2}\rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n\rangle$ into two bitonic sequences $\langle b_1, b_2, \ldots, b_{n/2}\rangle$ and $\langle b_{n/2+1}, b_{n/2+2}, \ldots, b_n\rangle$. **(b)** The equivalent operation for HALF-CLEANER[n]. The bitonic input sequence $\langle a_1, a_2, \ldots, a_{n/2-1}, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+2}, a_{n/2+1}\rangle$ is transformed into the two bitonic sequences $\langle b_1, b_2, \ldots, b_{n/2}\rangle$ and $\langle b_n, b_{n-1}, \ldots, b_{n/2+1}\rangle$.

Construction of a Merging Network (1/2)

- Given two sorted sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$
- We know it suffices to bitonically sort $\langle a_1, a_2, \ldots, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+1} \rangle$
- Recall: first half-cleaner of BITONIC-SORTER[n] compares i and $n/2 + i$

$\Rightarrow$ First part of MERGER[n] compares inputs i and $n - i$ for $i = 1, 2, \ldots, n/2$

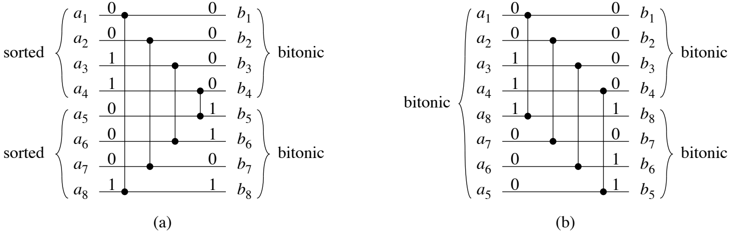


Figure 27.10 Comparing the first stage of MERGER[n] with HALF-CLEANER[n], for $n = 8$. **(a)** The first stage of MERGER[n] transforms the two monotonic input sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$ into two bitonic sequences $\langle b_1, b_2, \ldots, b_{n/2} \rangle$ and $\langle b_{n/2+1}, b_{n/2+2}, \ldots, b_n \rangle$. **(b)** The equivalent operation for HALF-CLEANER[n]. The bitonic input sequence $\langle a_1, a_2, \ldots, a_{n/2-1}, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+2}, a_{n/2+1} \rangle$ is transformed into the two bitonic sequences $\langle b_1, b_2, \ldots, b_{n/2} \rangle$ and $\langle b_n, b_{n-1}, \ldots, b_{n/2+1} \rangle$.

Construction of a Merging Network (1/2)

- Given two sorted sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$
- We know it suffices to bitonically sort $\langle a_1, a_2, \ldots, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+1} \rangle$
- Recall: first half-cleaner of BITONIC-SORTER[n] compares i and $n/2 + i$

$\Rightarrow$ First part of MERGER[n] compares inputs i and $n - i$ for $i = 1, 2, \ldots, n/2$

- Remaining part is identical to BITONIC-SORTER[n]

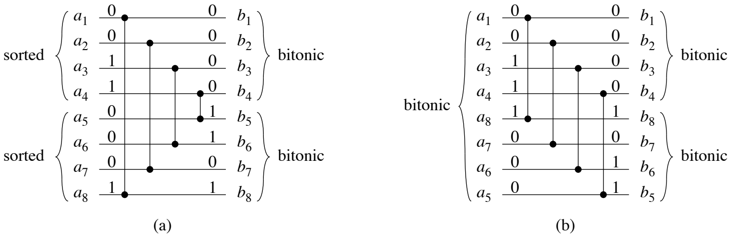


Figure 27.10 Comparing the first stage of MERGER[n] with HALF-CLEANER[n], for $n = 8$. **(a)** The first stage of MERGER[n] transforms the two monotonic input sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$ into two bitonic sequences $\langle b_1, b_2, \ldots, b_{n/2} \rangle$ and $\langle b_{n/2+1}, b_{n/2+2}, \ldots, b_n \rangle$. **(b)** The equivalent operation for HALF-CLEANER[n]. The bitonic input sequence $\langle a_1, a_2, \ldots, a_{n/2-1}, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+2}, a_{n/2+1} \rangle$ is transformed into the two bitonic sequences $\langle b_1, b_2, \ldots, b_{n/2} \rangle$ and $\langle b_n, b_{n-1}, \ldots, b_{n/2+1} \rangle$.

Construction of a Merging Network (2/2)

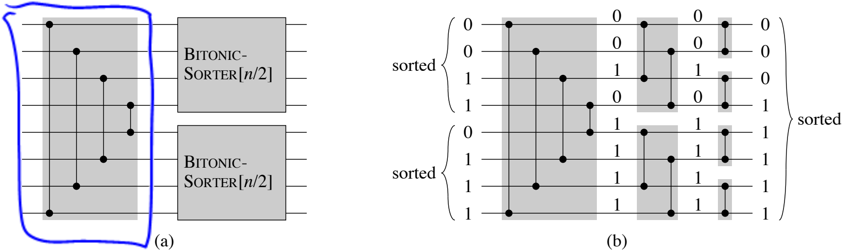


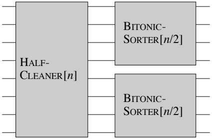
Figure 27.11 A network that merges two sorted input sequences into one sorted output sequence. The network MERGER[n] can be viewed as BITONIC-SORTER[n] with the first half-cleaner altered to compare inputs i and $n-i+1$ for $i = 1, 2, \ldots, n/2$. Here, $n = 8$. **(a)** The network decomposed into the first stage followed by two parallel copies of BITONIC-SORTER[$n/2$]. **(b)** The same network with the recursion unrolled. Sample zero-one values are shown on the wires, and the stages are shaded.

Construction of a Sorting Network

Main Components

1. BITONIC-SORTER[n]
   - sorts any bitonic sequence
   - depth $\log n$

1.

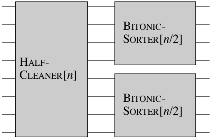


Construction of a Sorting Network

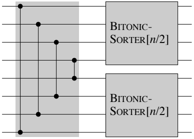
Main Components

1. BITONIC-SORTER[n]
   - sorts any bitonic sequence
   - depth $\log n$
2. MERGER[n]
   - merges two sorted input sequences
   - depth $\log n$

1.



2.

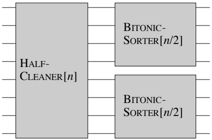


Construction of a Sorting Network

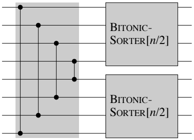
Main Components

1. BITONIC-SORTER[n]
   - sorts any bitonic sequence
   - depth $\log n$
2. MERGER[n]
   - merges two sorted input sequences
   - depth $\log n$

1.

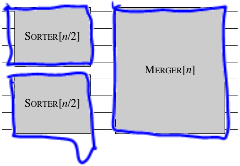


2.



Batcher's Sorting Network

- SORTER[n] is defined recursively:
  - If $n = 2^k$, use two copies of SORTER[$n/2$] to sort two subsequences of length $n/2$ each. Then merge them using MERGER[n].
  - If $n = 1$, network consists of a single wire.

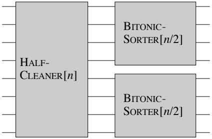


Construction of a Sorting Network

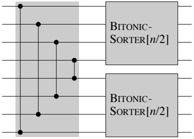
Main Components

1. BITONIC-SORTER$[n]$
   - sorts any bitonic sequence
   - depth $\log n$
2. MERGER$[n]$
   - merges two sorted input sequences
   - depth $\log n$

1.



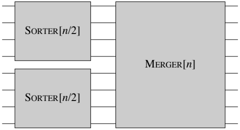
2.



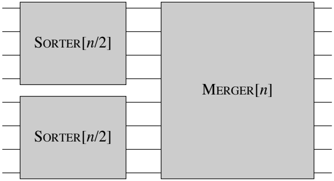
Batcher's Sorting Network

- SORTER$[n]$ is defined recursively:
  - If $n = 2^k$, use two copies of SORTER$[n/2]$ to sort two subsequences of length $n/2$ each. Then merge them using MERGER$[n]$.
  - If $n = 1$, network consists of a single wire.

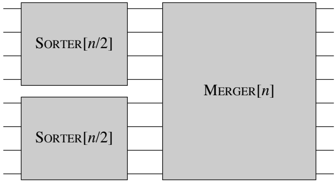
can be seen as a parallel version of merge sort

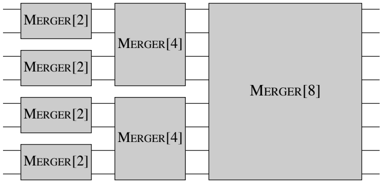


Unrolling the Recursion (Figure 27.12)

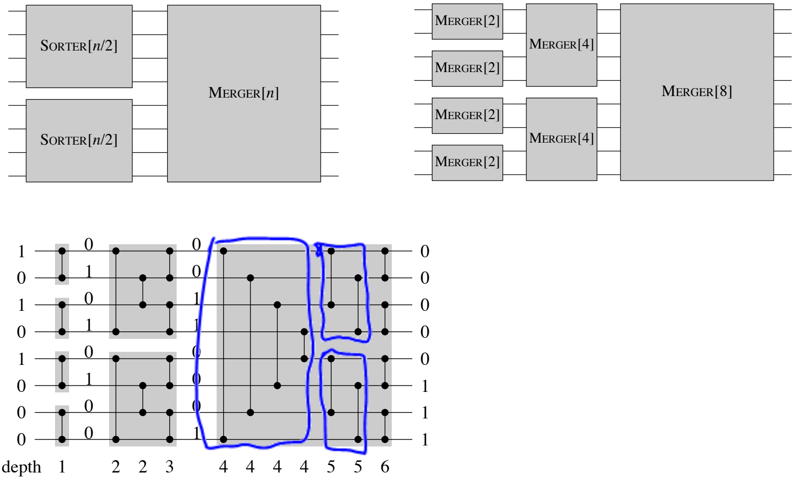


Unrolling the Recursion (Figure 27.12)

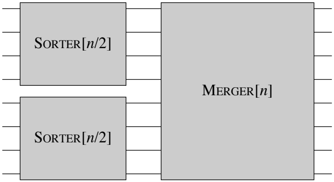


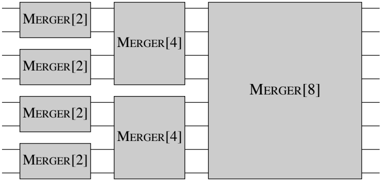


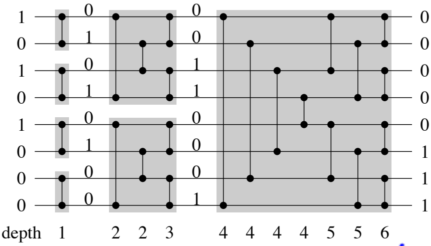
Unrolling the Recursion (Figure 27.12)



Unrolling the Recursion (Figure 27.12)







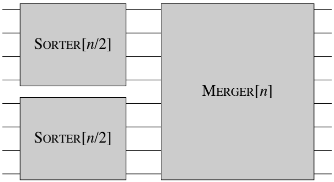
Recursion for $D(n)$:

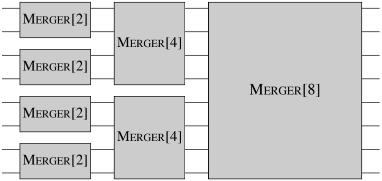
$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ \underline{D(n/2)} + \underline{\log n} & \text{if } n = 2^k. \end{cases}$$

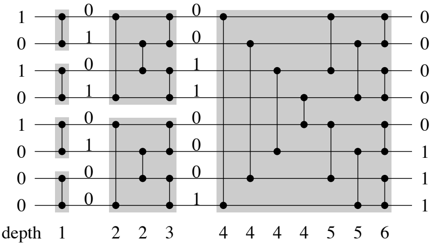
$$D(n) = \log n + \log\left(\frac{n}{2}\right) + \log\left(\frac{n}{4}\right) + \ldots + \log(2)$$

$$= \log n + (\log n - 1) + (\log n - 2) + \ldots + 1$$

Unrolling the Recursion (Figure 27.12)





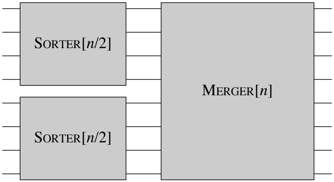


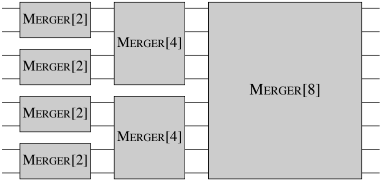
Recursion for $D(n)$:

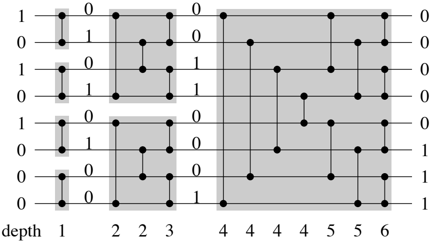
$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ D(n/2) + \log n & \text{if } n = 2^k. \end{cases}$$

Solution: $D(n) = \Theta(\log^2 n)$.

Unrolling the Recursion (Figure 27.12)







Recursion for $D(n)$:

$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ D(n/2) + \log n & \text{if } n = 2^k. \end{cases}$$

Solution: $D(n) = \Theta(\log^2 n)$.

SORTER$[n]$ has depth $\Theta(\log^2 n)$ and sorts any input.

A Glimpse at the AKS Network

Ajtai, Komlós, Szemerédi (1983)

There exists a sorting network with depth $\boxed{O(\log n)}$.

A Glimpse at the AKS Network

Ajtai, Komlós, Szemerédi (1983)

There exists a sorting network with depth $O(\log n)$.

Quite elaborate construction, and involves huges constants.

A Glimpse at the AKS Network

Ajtai, Komlós, Szemerédi (1983)

There exists a sorting network with depth $O(\log n)$.

Perfect Halver

A perfect halver is a comparator network that, given any input, places the $n/2$ smaller keys in $b_1, \ldots, b_{n/2}$ and the $n/2$ larger keys in $b_{n/2+1}, \ldots, b_n$.

A Glimpse at the AKS Network

Ajtai, Komlós, Szemerédi (1983)

There exists a sorting network with depth $O(\log n)$.

Perfect Halver

A perfect halver is a comparator network that, given any input, places the $n/2$ smaller keys in $b_1, \ldots, b_{n/2}$ and the $n/2$ larger keys in $b_{n/2+1}, \ldots, b_n$.

Perfect halver of depth $\log_2 n$ exist $\rightsquigarrow$ yields sorting networks of depth $\boxed{\Theta((\log n)^2)}$.

A Glimpse at the AKS Network

Ajtai, Komlós, Szemerédi (1983)

There exists a sorting network with depth $O(\log n)$.

Perfect Halver

A perfect halver is a comparator network that, given any input, places the $n/2$ smaller keys in $b_1, \ldots, b_{n/2}$ and the $n/2$ larger keys in $b_{n/2+1}, \ldots, b_n$.

Approximate Halver

An (n, ϵ)-approximate halver, $\epsilon < 1$, is a comparator network that for every $k = 1, 2, \ldots, n/2$ places at most ϵk of its k smallest keys in $b_{n/2+1}, \ldots, b_n$ and at most ϵk of its k largest keys in $b_1, \ldots, b_{n/2}$.

A Glimpse at the AKS Network

Ajtai, Komlós, Szemerédi (1983)

There exists a sorting network with depth $O(\log n)$.

Perfect Halver

A perfect halver is a comparator network that, given any input, places the $n/2$ smaller keys in $b_1, \ldots, b_{n/2}$ and the $n/2$ larger keys in $b_{n/2+1}, \ldots, b_n$.

Approximate Halver

An (n, ϵ)-approximate halver, $\epsilon < 1$, is a comparator network that for every $k = 1, 2, \ldots, n/2$ places at most ϵk of its k smallest keys in $b_{n/2+1}, \ldots, b_n$ and at most ϵk of its k largest keys in $b_1, \ldots, b_{n/2}$.

We will prove that such networks can be constructed in constant depth!