Advanced Algorithms

I. Course Intro and Sorting Networks

Thomas Sauerwald

Easter 2017
Outline of this Course

Some Highlights

Introduction to Sorting Networks

Batcher’s Sorting Network

Counting Networks
(Tentative) List of Topics

IA Algorithms
IB Complexity Theory
II Advanced Algorithms

I. Course Intro and Sorting Networks
II. Sorting Networks (Sorting, Counting)
III. Matrix Multiplication (and Parallel Algorithms)
IV. Linear Programming
V. Approximation Algorithms: Covering Problems
VI. Approximation Algorithms via Exact Algorithms
VII. Approximation Algorithms: Travelling Salesman Problem

closely follow CLRS3 and use the same numbering however, slides will be self-contained (mostly)
(Tentative) List of Topics

I. Course Intro and Sorting Networks
   Outline of this Course

IA Algorithms

IB Complexity Theory

II Advanced Algorithms

- I. Sorting Networks (Sorting, Counting)
- II. Matrix Multiplication (and Parallel Algorithms)
- III. Linear Programming
- IV. Approximation Algorithms: Covering Problems
- V. Approximation Algorithms via Exact Algorithms
- VI. Approximation Algorithms: Travelling Salesman Problem
- VII. Approximation Algorithms: Randomisation and Rounding
(Tentative) List of Topics

IA Algorithms

IB Complexity Theory

II Advanced Algorithms

- I. Sorting Networks (Sorting, Counting)
- II. Matrix Multiplication (and Parallel Algorithms)
- III. Linear Programming
- IV. Approximation Algorithms: Covering Problems
- V. Approximation Algorithms via Exact Algorithms
- VI. Approximation Algorithms: Travelling Salesman Problem
- VII. Approximation Algorithms: Randomisation and Rounding

- closely follow CLRS3 and use the same numbering
- however, slides will be self-contained (mostly)
(Tentative) List of Topics

I. Sorting Networks (Sorting, Counting)
II. Matrix Multiplication (and Parallel Algorithms)
III. Linear Programming
IV. Approximation Algorithms: Covering Problems
V. Approximation Algorithms via Exact Algorithms
VI. Approximation Algorithms: Travelling Salesman Problem
VII. Approximation Algorithms: Randomisation and Rounding

- closely follow CLRS3 and use the same nummbering
- however, slides will be self-contained (mostly)
Outline

Outline of this Course

Some Highlights

Introduction to Sorting Networks

Batcher’s Sorting Network

Counting Networks
maximize $3x_1 + x_2 + 2x_3$

subject to

$x_1 + x_2 + 3x_3 \leq 30$
$2x_1 + 2x_2 + 5x_3 \leq 24$
$4x_1 + x_2 + 2x_3 \leq 36$
$x_1, x_2, x_3 \geq 0$
Linear Programming and Simplex

maximize $3x_1 + x_2 + 2x_3$

subject to

- $x_1 + x_2 + 3x_3 \leq 30$
- $2x_1 + 2x_2 + 5x_3 \leq 24$
- $4x_1 + x_2 + 2x_3 \leq 36$
- $x_1, x_2, x_3 \geq 0$

I. Course Intro and Sorting Networks Some Highlights
maximize $3x_1 + x_2 + 2x_3$

subject to

$x_1 + x_2 + 3x_3 \leq 30$

$2x_1 + 2x_2 + 5x_3 \leq 24$

$4x_1 + x_2 + 2x_3 \leq 36$

$x_1, x_2, x_3 \geq 0$
maximize \( 3x_1 + x_2 + 2x_3 \)
subject to
\[
\begin{align*}
x_1 + x_2 + 3x_3 & \leq 30 \\
2x_1 + 2x_2 + 5x_3 & \leq 24 \\
4x_1 + x_2 + 2x_3 & \leq 36 \\
x_1, x_2, x_3 & \geq 0
\end{align*}
\]
Linear Programming and Simplex

maximize \[ 3x_1 + x_2 + 2x_3 \]
subject to
\[
\begin{align*}
    x_1 &+ x_2 + 3x_3 &\leq 30 \\
    2x_1 + 2x_2 + 5x_3 &\leq 24 \\
    4x_1 + x_2 + 2x_3 &\leq 36 \\
    x_1, x_2, x_3 &\geq 0
\end{align*}
\]
SOLUTION OF A LARGE-SCALE TRAVELING-SALESMAN PROBLEM*

G. DANTZIG, R. FULKERSON, AND S. JOHNSON
The Rand Corporation, Santa Monica, California
(Received August 9, 1954)

It is shown that a certain tour of 49 cities, one in each of the 48 states and Washington, D. C., has the shortest road distance.

THE TRAVELING-SALESMAN PROBLEM might be described as follows: Find the shortest route (tour) for a salesman starting from a given city, visiting each of a specified group of cities, and then returning to the original point of departure. More generally, given an \( n \) by \( n \) symmetric matrix \( D = (d_{ij}) \), where \( d_{ij} \) represents the 'distance' from \( I \) to \( J \), arrange the points in a cyclic order in such a way that the sum of the \( d_{ij} \) between consecutive points is minimal. Since there are only a finite number of possibilities (at most \( \frac{1}{2} (n-1)! \)) to consider, the problem is to devise a method of picking out the optimal arrangement which is reasonably efficient for fairly large values of \( n \). Although algorithms have been devised for problems of similar nature, e.g., the optimal assignment problem,\(^3,7,8\) little is known about the traveling-salesman problem. We do not claim that this note alters the situation very much; what we shall do is outline a way of approaching the problem that sometimes, at least, enables one to find an optimal path and prove it so. In particular, it will be shown that a certain arrangement of 49 cities, one in each of the 48 states and Washington, D. C., is best, the \( d_{ij} \) used representing road distances as taken from an atlas.
1. Manchester, N. H.  
4. Cleveland, Ohio  
7. Indianapolis, Ind.  
8. Chicago, Ill.  
9. Milwaukee, Wis.  
10. Minneapolis, Minn.  
11. Pierre, S. D.  
12. Bismarck, N. D.  
13. Helena, Mont.  
15. Portland, Ore.  
16. Boise, Idaho  
17. Salt Lake City, Utah  
18. Carson City, Nev.  
19. Los Angeles, Calif.  
21. Santa Fe, N. M.  
22. Denver, Colo.  
23. Cheyenne, Wyo.  
24. Omaha, Neb.  
25. Des Moines, Iowa  
26. Kansas City, Mo.  
27. Topeka, Kans.  
28. Oklahoma City, Okla.  
29. Dallas, Tex.  
30. Little Rock, Ark.  
31. Memphis, Tenn.  
32. Jackson, Miss.  
33. New Orleans, La.  
34. Birmingham, Ala.  
35. Atlanta, Ga.  
36. Jacksonville, Fla.  
37. Columbia, S. C.  
38. Raleigh, N. C.  
40. Washington, D. C.  
42. Portland, Me.  
A. Baltimore, Md.  
B. Wilmington, Del.  
C. Philadelphia, Penn.  
D. Newark, N. J.  
E. New York, N. Y.  
F. Hartford, Conn.  
G. Providence, R. I.
Road Distances

TABLE I

Road Distances between Cities in Adjusted Units

The figures in the table are mileages between the two specified numbered cities, less 11, divided by 17, and rounded to the nearest integer.
The (Unique) Optimal Tour (699 Units ≈ 12,345 miles)

Fig. 16. The optimal tour of 49 cities.

This tour has a length of 12,345 miles when the adjusted units are expressed in miles.
Outline of this Course

Some Highlights

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

A comparison network consists solely of wires and comparators.
Comparison Networks

A comparison network consists solely of wires and comparators:
- Comparator is a device with, on given two inputs, \( x \) and \( y \), returns two outputs \( x' = \min(x, y) \) and \( y' = \max(x, y) \).

**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

A comparison network consists solely of wires and comparators:
- comparator is a device with, on given two inputs, $x$ and $y$, returns two outputs $x' = \min(x, y)$ and $y' = \max(x, y)$

**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

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

A comparison network consists solely of wires and comparators:

- comparator is a device with, on given two inputs, \( x \) and \( y \), returns two outputs \( x' = \min(x, y) \) and \( y' = \max(x, y) \)
- 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.
A comparison network consists solely of wires and comparators:
- comparator is a device with, on given two inputs, \( x \) and \( y \), returns two outputs \( x' = \min(x, y) \) and \( y' = \max(x, y) \)
- 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.
A sorting network is a comparison network which works correctly (that is, it sorts every input).

A comparison network consists solely of wires and comparators:
- comparator is a device with, on given two inputs, $x$ and $y$, returns two outputs $x' = \min(x, y)$ and $y' = \max(x, y)$
- 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)

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

Depth

The maximum depth of an output wire equals the total running time.

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)

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise).

This network would not be a sorting network (Why??)

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.

Interconnections between comparators must be acyclic.

✓
Example of a Comparison Network (Figure 27.2)

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise)
This network would not be a sorting network (Why??)

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
Example of a Comparison Network (Figure 27.2)

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise).

This network would not be a sorting network (Why??)

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.

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)

A horizontal line represents a sequence of distinct wires

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

Interconnections between comparators must be acyclic

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

I. Course Intro and Sorting Networks Introduction to Sorting Networks 13
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

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

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)

Interconnections between comparators must be acyclic

This network is in fact a sorting network (Exercise)
This network would not be a sorting network (Why??)

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

Interconnections between comparators must be acyclic.
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise).
This network would not be a sorting network (Why??)

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.

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)

Interconnections between comparators must be acyclic

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

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

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)

Interconnections between comparators must be acyclic.

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise).

This network would not be a sorting network (Why??)

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.

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)

Interconnections between comparators must be acyclic

I. Course Intro and Sorting Networks

Introduction to Sorting Networks 13
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic

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

Interconnections between comparators must be acyclic
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic ✓

A horizontal line represents a sequence of distinct wires. This network is in fact a sorting network (Exercise). This network would not be a sorting network (Why??)

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.

Interconnections between comparators must be acyclic ✓
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be **acyclic**

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

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.

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)

Interconnections between comparators must be **acyclic**

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

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

Interconnections between comparators must be acyclic.
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise).

This network would not be a sorting network (Why??)

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.

Interconnections between comparators must be acyclic.

Tracing back a path must never cycle back on itself and go through the same comparator twice.
Interconnections between comparators must be acyclic

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise).

This network would not be a sorting network (Why??)

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.

Interconnections between comparators must be acyclic.
Interconnections between comparators must be acyclic.

Example of a Comparison Network (Figure 27.2)

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

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

Interconnections between comparators must be acyclic.
Interconnections between comparators must be acyclic.
Interconnections between comparators must be acyclic.

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise).

This network would not be a sorting network (Why??)

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.

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.

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

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
Example of a Comparison Network (Figure 27.2)

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

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)

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

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

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)

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

Maximum depth of an output wire equals total running time.

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)

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise).

This network would not be a sorting network (Why??)

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.

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)

This network is in fact a sorting network (Exercise)
Example of a Comparison Network (Figure 27.2)

This network would not be a sorting network (Why??)

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

Interconnections between comparators must be acyclic

✓
Example of a Comparison Network (Figure 27.2)

This network is in fact a sorting network (Exercise). This network would not be a sorting network (Why??)

Depth of a wire:

A horizontal line represents a sequence of distinct wires.

Interconnections between comparators must be acyclic. Tracing back a path must never cycle back on itself and go through the same comparator twice.

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

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise)
This network would not be a sorting network (Why??)

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)

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

**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)

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

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)**

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise)

This network would not be a sorting network (Why??)

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)

This network is in fact a sorting network (Exercise)
This network would not be a sorting network (Why??)

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)

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise).

This network would not be a sorting network (Why??)

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)

A horizontal line represents a sequence of distinct wires.

This network is in fact a sorting network (Exercise).

This network would not be a sorting network (Why??)

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.

Interconnections between comparators must be acyclic.

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

Depth
\[
\begin{array}{cccccc}
0 & 1 & 1 & 2 & 2 & 3 \\
\end{array}
\]
Zero-One Principle: A sorting network works correctly on arbitrary inputs if it works correctly on binary inputs.
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 $f(b) = \langle f(b_1), f(b_2), \ldots, f(b_n) \rangle$. 
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 \( f(b) = \langle f(b_1), f(b_2), \ldots, f(b_n) \rangle \).

\[
\begin{align*}
\min(f(x), f(y)) &= f(\min(x, y)) \\
\max(f(x), f(y)) &= f(\max(x, y))
\end{align*}
\]

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:

- 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.
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 \( f(a_j) \) is placed before \( f(a_i) \).
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.
Some Basic (Recursive) Sorting Networks

I. Course Intro and Sorting Networks

Introduction to Sorting Networks
Some Basic (Recursive) Sorting Networks

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

Bubble Sort

Insertion Sort
Some Basic (Recursive) Sorting Networks

$n$-wire Sorting Network

Bubble Sort

Insertion Sort

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

I. Course Intro and Sorting Networks
Some Basic (Recursive) Sorting Networks

I. Course Intro and Sorting Networks Introduction to Sorting Networks
Some Basic (Recursive) Sorting Networks

These are Sorting Networks, but with depth \( \Theta(n) \).

I. Course Intro and Sorting Networks

Introduction to Sorting Networks
Outline of this Course

Some Highlights

Introduction to Sorting Networks

Batcher’s Sorting Network

Counting Networks
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

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

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

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:
- \([1, 4, 6, 8, 3, 2]\)?
Bitonic Sequences

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

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

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

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:
- \( \langle 1, 4, 6, 8, 3, 2 \rangle \) ✓
- \( \langle 6, 9, 4, 2, 3, 5 \rangle \) ✓
- \( \langle 9, 8, 3, 2, 4, 6 \rangle \) ✓
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:

- \(1, 4, 6, 8, 3, 2\) ✓
- \(6, 9, 4, 2, 3, 5\) ✓
- \(9, 8, 3, 2, 4, 6\) ✓
- \(4, 5, 7, 1, 2, 6\) ?
Bitonic Sequences

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

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

- \(\langle 1, 4, 6, 8, 3, 2 \rangle\) ✓
- \(\langle 6, 9, 4, 2, 3, 5 \rangle\) ✓
- \(\langle 9, 8, 3, 2, 4, 6 \rangle\) ✓
- \(\langle 4, 5, 7, 1, 2, 6 \rangle\)
- binary sequences: \(0^i1^j0^k\), or, \(1^i0^j1^k\), 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 $i = 1, 2, \ldots, n/2$.

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 $i = 1, 2, \ldots, n/2$. 

![Diagram of a half-cleaner network]
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 $i = 1, 2, \ldots, n/2$.

---

**Figure 27.7**

The comparison network $HALFCLEANER[8]$, the half-cleaner with 8 inputs and 8 outputs. When a bitonic sequence of 0’s and 1’s is applied as input to a half-cleaner, the half-cleaner produces an output sequence in which smaller values are in the top half, larger values are in the bottom half, and both halves are bitonic. In fact, at least one of the halves is clean—consisting of either all 0’s or all 1’s—and it is from this property that we derive the name “half-cleaner.” (Note that all clean sequences are bitonic.) The next lemma proves these properties of half-cleaners.

**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 half is at least as small as every element of the bottom half, and at least one half is clean.

**Proof**

The comparison network $HALFCLEANER[n]$ compares inputs $i$ and $i + n/2$ for $i = 1, 2, \ldots, n/2$. Without loss of generality, suppose that the input is of the form $00 \ldots 011 \ldots 100 \ldots 0$. (The situation in which the input is of the form $11 \ldots 100 \ldots 011 \ldots 1$ is symmetric.) There are possible cases depending upon the block of consecutive 0’s or 1’s in which the midpoint $n/2$ falls, and one of these cases (the one in which the midpoint occurs in the block of 1’s) is further split into two cases. The four cases are shown in Figure 27.8. In each case shown, the lemma holds.
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 $i = 1, 2, \ldots, n/2$.

![Diagram of a half-cleaner network](image-url)
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 half is at least as small as every element in the bottom half.
- At least one half is clean.

**Proof**

The comparison network has $n$ inputs and $n$ outputs. Without loss of generality, suppose that the input is of the form $00\ldots 011\ldots 100\ldots 0$. (The situation in which the input is of the form $11\ldots 100\ldots 011\ldots 1$ is symmetric.) There are several possible cases depending upon the block of consecutive 0's or 1's in which the midpoint $n/2$ falls, and one of these cases (the one in which the midpoint occurs in the block of 1's) is further split into two cases. The four cases are shown in Figure 27.8. In each case shown, the lemma holds.
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

The comparison network \( \text{HALF-CLEANER} \) compares inputs \( i \) and \( i + n/2 \) for \( i = 1, 2, \ldots, n/2 \). Without loss of generality, suppose that the input is of the form \( 00 \ldots 011 \ldots 100 \ldots 0 \). (The situation in which the input is of the form \( 11 \ldots 100 \ldots 011 \ldots 1 \) is symmetric.) There are possible cases depending upon the block of consecutive 0’s or 1’s in which the midpoint \( n/2 \) falls, and one of these cases (the one in which the midpoint occurs in the block of 1’s) is further split into two cases. The four cases are shown in Figure 27.8. In each case shown, the lemma holds.
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 $0^i1^j0^k$, for some $i, j, k \geq 0$. 
Proof of Lemma 27.3

W.l.o.g. assume that the input is of the form $0^i1^j0^k$, for some $i, j, k \geq 0$.

![Diagram of sorting network division, compare, and combine stages](image1)

Figure 27.8 The possible comparisons in the H_ALFLEANER network. The input sequence is assumed to be a bitonic sequence of 0's and 1's, and without loss of generality, we assume that it is of the form $00...011...100...0$. Subsequences of 0's are white, and subsequences of 1's are gray. We can think of the $n$ inputs as being divided into two halves such that for $i = 1, 2, ..., n/2$, inputs $i$ and $i + n/2$ are compared.

(a) – (b) Cases in which the division occurs in the middle subsequence of 1's.

(c) – (d) Cases in which the division occurs in a subsequence of 0's. For all cases, every element in the top half of the output is at least as small as every element in the bottom half, both halves are bitonic, and at least one half is clean.
Proof of Lemma 27.3

W.l.o.g. assume that the input is of the form $0^i1^j0^k$, for some $i, j, k \geq 0$. 

![Diagram](image.png)

**Figure 27.8** The possible comparisons in $\text{HALF}-\text{LEANER} [n]$. The input sequence is assumed to be a bitonic sequence of 0's and 1's, and without loss of generality, we assume that it is of the form $00 \ldots 011 \ldots 100 \ldots 0$. Subsequences of 0's are white, and subsequences of 1's are gray. We can think of the $n$ inputs as being divided into two halves such that for $i = 1, 2, \ldots, n/2$, inputs $i$ and $i + n/2$ are compared. 

(a)–(b) Cases in which the division occurs in the middle subsequence of 1's. 

(c)–(d) Cases in which the division occurs in a subsequence of 0's. For all cases, every element in the top half of the output is at least as small as every element in the bottom half, both halves are bitonic, and at least one half is clean.

This suggests a recursive approach, since it now suffices to sort the top and bottom half separately.
Proof of Lemma 27.3

W.l.o.g. assume that the input is of the form $0^i1^j0^k$, for some $i, j, k \geq 0$. 

The possible comparisons in $H_{ALF-CLEANER}[n]$. The input sequence is assumed to be a bitonic sequence of 0's and 1's, and without loss of generality, we assume that it is of the form $00...011...100...0$. Subsequences of 0's are white, and subsequences of 1's are gray. We can think of the $n$ inputs as being divided into two halves such that for $i = 1, 2, ..., n/2$, inputs $i$ and $i + n/2$ are compared.

(a)–(b) Cases in which the division occurs in the middle subsequence of 1's.
(c)–(d) Cases in which the division occurs in a subsequence of 0's. For all cases, every element in the top half of the output is at least as small as every element in the bottom half, both halves are bitonic, and at least one half is clean.

This suggests a recursive approach, since it now suffices to sort the top and bottom half separately.
Proof of Lemma 27.3

W.l.o.g. assume that the input is of the form $0^i1^j0^k$, for some $i, j, k \geq 0$.

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.

Recursive Formula for depth $D(n)$:

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

Henceforth we will always assume that $n$ is a power of 2.
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 \( D(n) \):

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

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

BITONIC-SORTER[n] has depth log \( n \) and sorts any zero-one bitonic sequence.
Merging Networks

- can merge two sorted input sequences into one sorted output sequence
- will be based on a modification of BITONIC-SORTER\([n]\)
Merging Networks

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

Basic Idea:
Merging Networks

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

Basic Idea:
- consider two given sequences \(X = 00000111, \ Y = 0001111\)
Merging Networks

- can merge two sorted input sequences into one sorted output sequence
- 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 $Y$) $\Rightarrow 0000111111110000$
Merging Networks

- can merge two sorted input sequences into one sorted output sequence
- 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 \(Y\)) ⇒ \(000011111110000\)

This sequence is bitonic!
Merging Networks

- can merge two sorted input sequences into one sorted output sequence
- 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 \(Y\)) \(\Rightarrow 0000111111110000\)

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 \( \langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle \)
- 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 \( n - i + 1 \) 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 \textsc{Bitonic-Sorter}[n] compares \( i \) and \( n/2 + i \)

\( \Rightarrow \) First part of \textsc{Merger}[n] compares inputs \( i \) and \( n - i + 1 \) for \( i = 1, 2, \ldots, n/2 \)

**Figure 27.10** Comparing the first stage of \textsc{Merger}[n] with \textsc{Half-Cleaner}[n], for \( n = 8 \).

(a) The first stage of \textsc{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 \textsc{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 \text{First part of MERGER}[\( n \)] compares inputs } i \text{ and } n - i + 1 \text{ 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 \( n - i + 1 \) for \( i = 1, 2, \ldots, n/2 \)
- Remaining part is identical to BITONIC-SORTER\([n]\)

![Diagram](image)

**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 \(\log n\)

---

In the depth after separately sorting with the actual merging networks, the depth of each comparator is \(\Omega(n \log n)\).

Exercise 27.3-6: Prove that any bitonic sequence of arbitrary numbers can be sorted by this network.

Exercises

27.3-1: How many zero-one bitonic sequences of length \(n\) are there?

27.3-2: Prove that the number of comparators in this kind of merging network is \(\Omega(n \log n)\).

The bitonic sorter works. By recursively combining half-cleaners, as shown in Figure 27.9, we can sort any bitonic sequence of arbitrary numbers.

Exercise 27.3-6: that any bitonic sequence of arbitrary numbers can be sorted by this network.

Prove that any merging network, regardless of the order of inputs, requires \(\Omega(n \log n)\) comparators.
### 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 \( \log n \)

### Batcher's Sorting Network

The network decomposed into \( \mathcal{D} \) is defined recursively:

- \( n = 2 \) if \( \mathcal{D} \) consists of a single wire.
- Otherwise, use two copies of \( \mathcal{S}(k) \) to sort the two input sequences.

Partition the sequence into two monotonic sequences of half the size, and sort them using \( \mathcal{D}(k/2) \).

Merge the two sorted sequences using \( \mathcal{M} \), which, by Lemma 27.3, is a merging network.

- The first stage is followed by two parallel copies of \( \mathcal{B} \) in which the two monotonic sequences to be merged are sorted. The depth of each comparison network to verify that it is a merging network?

*Prove that any merging network, regardless of the order of inputs, requires \( \Omega(n \log n) \) comparators.*

- Figure 27.11 shows the network after unrolling the recursion. Each half-cleaner that makes up the remainder of the bitonic sorter is shaded. Sampling zero-one values are shown on the wires. We can build sequences of 0's and 1's can merge any two monotonically increasing sequences of length \( n \).

- Experiments 27.3-6 that any bitonic sequence of arbitrary numbers can be sorted by this network.

*How many zero-one bitonic sequences of length \( n \) are there?*
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

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.
Main Components

1. **BITONIC-SORTER[n]**
   - sorts any bitonic sequence
   - depth \( \log n \)

2. **MERGER[n]**
   - merges two sorted input sequences
   - depth \( \log n \)

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)

Let $S^\star$ be the depth of a sorting network with $2^k$ elements. Suppose that we have a sequence of $n$ elements and wish to partition them into $\log n$ sets, each containing $\frac{n}{\log n}$ elements. We know that every number is within $\log n$ of the smallest and the largest. Let $M(n)$ be the depth of each comparator in the actual merging networks. The depth of each comparator in the merging networks is $\frac{\log n}{2}$.

Show that the depth of $S^\star$ is $\Theta(n \log n)$, where $\Theta$ is the asymptotic notation.
Unrolling the Recursion (Figure 27.12)
Unrolling the Recursion (Figure 27.12)

Suppose that we have $2^k$ numbers to be sorted and we know that every number is within $[2^k]$. Let $S_k$ be the depth of a sorting network with $2^k$ numbers and sorts any input. The sorting network $S_k$ works.

The sorting network $S_{k+1} = S_k + 2M_k$ associates the positions of its smallest and the largest. Prove that we can do this in constant additional elements.

Unrolling the Recursion (Figure 27.12)
Unrolling the Recursion (Figure 27.12)

Recursion for $D(n)$:

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

\textbf{SORTER}[n] \text{ has depth } \Theta(\log^2 n) \text{ and sorts any input.}
A Glimpse at the AKS Network

Ajtai, Komlós, Szemerédi (1983)
There exists a sorting network with depth $O(\log n)$.

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

We will prove that such networks can be constructed in constant depth!
A Glimpse at the AKS Network

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

Quite elaborate construction, and involves huge constants.

Ajtai, Komlós, Szemerédi (1983)
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 comparison 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 $b_{n/2+1}, \ldots, b_n$. 

We will prove that such networks can be constructed in constant depth!
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 comparison 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 $\log n$ exist $\leadsto$ yields sorting networks of depth $\Theta((\log n)^2)$. 
A Glimpse at the AKS Network

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

Ajtai, Komlós, Szemerédi (1983)

Perfect Halver

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

We will prove that such networks can be constructed in constant depth!
A bipartite \((n, d, \mu)\)-expander is a graph with:
- \(G\) has \(n\) vertices \((n/2\) on each side)
- the edge-set is union of \(d\) perfect matchings
- For every subset \(S \subseteq V\) being in one part,
  \[|N(S)| > \min\{\mu \cdot |S|, n/2 - |S|\}\]
A bipartite \((n, d, \mu)\)-expander is a graph with:
- \(G\) has \(n\) vertices \((n/2 \text{ on each side})\)
- the edge-set is union of \(d\) perfect matchings
- For every subset \(S \subseteq V\) being in one part,

\[|N(S)| > \min\{\mu \cdot |S|, n/2 - |S|\}\]
Expander Graphs

A bipartite \((n, d, \mu)\)-expander is a graph with:
- \(G\) has \(n\) vertices (\(n/2\) on each side)
- the edge-set is union of \(d\) perfect matchings
- For every subset \(S \subseteq V\) being in one part,

\[ |N(S)| > \min\{\mu \cdot |S|, n/2 - |S|\} \]
Expander Graphs

A bipartite \((n, d, \mu)\)-expander is a graph with:

- \(G\) has \(n\) vertices \((n/2\) on each side)
- the edge-set is union of \(d\) perfect matchings
- For every subset \(S \subseteq V\) being in one part,

\[ |N(S)| > \min\{\mu \cdot |S|, n/2 − |S|\} \]
Expander Graphs

A bipartite \((n, d, \mu)\)-expander is a graph with:
- \(G\) has \(n\) vertices \((n/2\) on each side)
- the edge-set is union of \(d\) perfect matchings
- For every subset \(S \subseteq V\) being in one part,

\[
|N(S)| > \min\{\mu \cdot |S|, n/2 - |S|\}
\]

Specific definition tailored for sorting network - many other variants exist!
Expander Graphs

A bipartite \((n, d, \mu)\)-expander is a graph with:

- \(G\) has \(n\) vertices \((n/2\) on each side) 
- the edge-set is union of \(d\) perfect matchings 
- For every subset \(S \subseteq V\) being in one part, 

\[ |N(S)| > \min\{\mu \cdot |S|, n/2 - |S|\} \]

Expander Graphs:

- **probabilistic construction** “easy”: take \(d\) (disjoint) random matchings
- **explicit construction** is a deep mathematical problem with ties to number theory, group theory, combinatorics etc.
- **many applications** in networking, complexity theory and coding theory
From Expanders to Approximate Halvers

I. Course Intro and Sorting Networks

Batcher’s Sorting Network
From Expanders to Approximate Halvers

I. Course Intro and Sorting Networks

Batcher’s Sorting Network
From Expanders to Approximate Halvers

\[ \text{Diagram:}\]

\[ L \quad R \]
From Expanders to Approximate Halvers

I. Course Intro and Sorting Networks

Batcher’s Sorting Network
From Expanders to Approximate Halvers

I. Course Intro and Sorting Networks

Batcher's Sorting Network
From Expanders to Approximate Halvers

I. Course Intro and Sorting Networks

Batcher’s Sorting Network
From Expanders to Approximate Halvers

I. Course Intro and Sorting Networks

Batcher's Sorting Network
From Expanders to Approximate Halvers

I. Course Intro and Sorting Networks

Batcher’s Sorting Network
From Expanders to Approximate Halvers

I. Course Intro and Sorting Networks

Batcher's Sorting Network

29
Existence of Approximate Halvers (not examinable)

Proof:

\[
X := \text{keys with the } k \text{ smallest inputs}
\]

\[
Y := \text{wires in lower half with } k \text{ smallest outputs}
\]

For every \( u \in \mathbb{N}(Y) \):

\[\exists \text{ comparat. } (u, v), v \in Y \]

Let \( u_t, v_t \) be their keys after the comparator

Let \( u_d, v_d \) be their keys at the output

Note that \( v_d \in X \)

Further:

\[ u_d \leq u_t \leq v_t \leq v_d \]

\[ \Rightarrow u_d \in X \]

Since \( u \) was arbitrary:

\[ |Y| + |\mathbb{N}(Y)| \leq k. \]

Since \( G \) is a bipartite \((n, d, \mu)\)-expander:

\[ |Y| + |\mathbb{N}(Y)| > |Y| + \min\{\mu, n/2 - |Y|\} = \min\{1 + \mu |Y|, n/2\} \]

Combining the two bounds above yields:

\[ (1 + \mu) |Y| \leq k. \]

Same argument \( \Rightarrow \) at most \( \epsilon \cdot k \), \( \epsilon := 1/ (\mu + 1) \), of the \( k \) largest input keys are placed in \( b_1, \ldots, b_{n/2} \).

Here we used that \( k \leq n/2 \)
Existence of Approximate Halvers (not examinable)

Proof:

- \( X := \) keys with the \( k \) smallest inputs
Existence of Approximate Halvers (not examinable)

Proof:

- $X :=$ keys with the $k$ smallest inputs
- $Y :=$ wires in lower half with $k$ smallest outputs

Let $u, v \in Y$ be their keys after the comparator
Let $u_d, v_d$ be their keys at the output
Note that $v_d \in X$
Further:
$u_d \leq u_t \leq v_t \leq v_d \Rightarrow u_d \in X$
Since $u$ was arbitrary:
$|Y| + |N(Y)| \leq k$
Since $G$ is a bipartite $(n, d, \mu)$-expander:
$|Y| + |N(Y)| > |Y| + \min\{\mu, n/2 - |Y|\} = \min\{(1 + \mu)|Y|, n/2\}$
Combining the two bounds above yields:
$(1 + \mu)|Y| \leq k$
Same argument $\Rightarrow$ at most $\epsilon \cdot k$, $\epsilon := 1/(\mu + 1)$, of the $k$ largest input keys are placed in $b_1, \ldots, b_{n/2}$.

Here we used that $k \leq n/2$
Existence of Approximate Halvers (not examinable)

Proof:

- $X :=$ keys with the $k$ smallest inputs
- $Y :=$ wires in lower half with $k$ smallest outputs
- For every $u \in N(Y)$: $\exists$ comparat. $(u, v), v \in Y$

**Typical Application of Expander Graphs in Parallel Algorithms**

Much more work needed to construct the AKS Sorting Network
Existence of Approximate Halvers (not examinable)

Proof:

- $X := \text{keys with the } k \text{ smallest inputs}$
- $Y := \text{wires in lower half with } k \text{ smallest outputs}$
- For every $u \in N(Y)$: $\exists$ comparat. $(u, v), v \in Y$
- Let $u_t, v_t$ be their keys after the comparator
  Let $u_d, v_d$ be their keys at the output
Existence of Approximate Halvers (not examinable)

Proof:

- $X := \text{keys with the } k \text{ smallest inputs}$
- $Y := \text{wires in lower half with } k \text{ smallest outputs}$
- For every $u \in N(Y)$: $\exists$ comparat. $(u, v), v \in Y$
- Let $u_t, v_t$ be their keys after the comparator
  Let $u_d, v_d$ be their keys at the output
Existence of Approximate Halvers (not examinable)

Proof:

- \( X := \) keys with the \( k \) smallest inputs
- \( Y := \) wires in lower half with \( k \) smallest outputs
- For every \( u \in N(Y) \): \( \exists \) comparat. \( (u, v) \), \( v \in Y \)
- Let \( u_t, v_t \) be their keys after the comparator
  - Let \( u_d, v_d \) be their keys at the output
- Note that \( v_d \in X \)
Existence of Approximate Halvers (not examinable)

Proof:

- $X :=$ keys with the $k$ smallest inputs
- $Y :=$ wires in lower half with $k$ smallest outputs
- For every $u \in N(Y)$: $\exists$ comparat. $(u, v)$, $v \in Y$
- Let $u_t, v_t$ be their keys after the comparator
  Let $u_d, v_d$ be their keys at the output
- Note that $v_d \in X$
- Further: $u_d \leq u_t \leq v_t \leq v_d$
Existence of Approximate Halvers (not examinable)

Proof:

- $X := \text{keys with the } k \text{ smallest inputs}$
- $Y := \text{wires in lower half with } k \text{ smallest outputs}$
- For every $u \in N(Y)$: $\exists$ comparat. $(u, v), v \in Y$
- Let $u_t, v_t$ be their keys after the comparator
- Let $u_d, v_d$ be their keys at the output
- Note that $v_d \in X$
- Further: $u_d \leq u_t \leq v_t \leq v_d \Rightarrow u_d \in X$
- Since $u$ was arbitrary:

$$|Y| + |N(Y)| \leq k.$$
Existence of Approximate Halvers (not examinable)

Proof:

- $X := \text{keys with the } k \text{ smallest inputs}$
- $Y := \text{wires in lower half with } k \text{ smallest outputs}$
- For every $u \in N(Y)$: $\exists$ comparat. $(u, v)$, $v \in Y$
- Let $u_t, v_t$ be their keys after the comparator
- Let $u_d, v_d$ be their keys at the output
- Note that $v_d \in X$
- Further: $u_d \leq u_t \leq v_t \leq v_d \Rightarrow u_d \in X$
- Since $u$ was arbitrary:
  \[ |Y| + |N(Y)| \leq k. \]
- Since $G$ is a bipartite $(n, d, \mu)$-expander:

\[ |Y| + |N(Y)| \leq k. \]

\[ |Y| + |N(Y)| \leq k. \]
Existence of Approximate Halvers (not examinable)

Proof:

- $X := \text{keys with the } k \text{ smallest inputs}$
- $Y := \text{wires in lower half with } k \text{ smallest outputs}$
- For every $u \in N(Y)$: $\exists$ comparat. $(u, v)$, $v \in Y$
- Let $u_t, v_t$ be their keys after the comparator
  - Let $u_d, v_d$ be their keys at the output
- Note that $v_d \in X$
- Further: $u_d \leq u_t \leq v_t \leq v_d \Rightarrow u_d \in X$
- Since $u$ was arbitrary:
  \[
  |Y| + |N(Y)| \leq k.
  \]
- Since $G$ is a bipartite $(n, d, \mu)$-expander:
  \[
  |Y| + |N(Y)| > |Y| + \min\{\mu |Y|, n/2 - |Y|\} = \min\{(1 + \mu)|Y|, n/2\}.
  \]

\[\text{Combining the two bounds above yields:} \quad (1 + \mu)|Y| \leq k.\]

\[\text{Same argument} \Rightarrow \text{at most} \ \epsilon \cdot k, \quad \epsilon := 1/(\mu + 1), \text{of the } k \text{ largest input keys are placed in } b_1, \ldots, b_{n/2}.\]
Existence of Approximate Halvers (not examinable)

Proof:

- $X :=$ keys with the $k$ smallest inputs
- $Y :=$ wires in lower half with $k$ smallest outputs
- For every $u \in N(Y)$: $\exists$ comparat. $(u, v)$, $v \in Y$
- Let $u_t, v_t$ be their keys after the comparator
  - Let $u_d, v_d$ be their keys at the output
- Note that $v_d \in X$
- Further: $u_d \leq u_t \leq v_t \leq v_d \Rightarrow u_d \in X$
- Since $u$ was arbitrary:
  \[ |Y| + |N(Y)| \leq k. \]

- Since $G$ is a bipartite $(n, d, \mu)$-expander:
  \[ |Y| + |N(Y)| > |Y| + \min\{\mu |Y|, n/2 - |Y|\} \]
Existence of Approximate Halvers (not examinable)

Proof:

- \( X \) := keys with the \( k \) smallest inputs
- \( Y \) := wires in lower half with \( k \) smallest outputs
- For every \( u \in N(Y) \): \( \exists \) comparat. \( (u, v) \), \( v \in Y \)
- Let \( u_t, v_t \) be their keys after the comparator
  - Let \( u_d, v_d \) be their keys at the output
- Note that \( v_d \in X \)
- Further: \( u_d \leq u_t \leq v_t \leq v_d \Rightarrow u_d \in X \)
- Since \( u \) was arbitrary:
  \[
  |Y| + |N(Y)| \leq k.
  \]
- Since \( G \) is a bipartite \((n, d, \mu)\)-expander:
  \[
  |Y| + |N(Y)| > |Y| + \min\{\mu|Y|, n/2 - |Y|\}
  = \min\{(1 + \mu)|Y|, n/2\}.
  \]
Existence of Approximate Halvers (not examinable)

Proof:

- \( X := \) keys with the \( k \) smallest inputs
- \( Y := \) wires in lower half with \( k \) smallest outputs
- For every \( u \in N(Y) \): \( \exists \) comparat. \((u, v), v \in Y\)
- Let \( u_t, v_t \) be their keys after the comparator
- Let \( u_d, v_d \) be their keys at the output
- Note that \( v_d \in X \)
- Further: \( u_d \leq u_t \leq v_t \leq v_d \Rightarrow u_d \in X \)
- Since \( u \) was arbitrary:
  \[
  |Y| + |N(Y)| \leq k.
  \]

- Since \( G \) is a bipartite \((n, d, \mu)\)-expander:
  \[
  |Y| + |N(Y)| > |Y| + \min\{\mu |Y|, n/2 - |Y|\} = \min\{(1 + \mu)|Y|, n/2\}.
  \]

- Combining the two bounds above yields:
  \[
  (1 + \mu)|Y| \leq k.
  \]
Existence of Approximate Halvers (not examinable)

Proof:

- \( X := \) keys with the \( k \) smallest inputs
- \( Y := \) wires in lower half with \( k \) smallest outputs
- For every \( u \in N(Y) \): \( \exists \) comparat. \((u, v), v \in Y\)
- Let \( u_t, v_t \) be their keys after the comparator
  Let \( u_d, v_d \) be their keys at the output
- Note that \( v_d \in X \)
- Further: \( u_d \leq u_t \leq v_t \leq v_d \Rightarrow u_d \in X \)
- Since \( u \) was arbitrary:
  \[
  |Y| + |N(Y)| \leq k.
  \]
- Since \( G \) is a bipartite \((n, d, \mu)\)-expander:
  \[
  |Y| + |N(Y)| > |Y| + \min\{\mu|Y|, n/2 - |Y|\}
  \]
  \[
  = \min\{(1 + \mu)|Y|, n/2\}.
  \]
- Combining the two bounds above yields:
  \[
  (1 + \mu)|Y| \leq k.
  \]

Here we used that \( k \leq n/2 \)
Existence of Approximate Halvers (not examinable)

Proof:

- \( X := \) keys with the \( k \) smallest inputs
- \( Y := \) wires in lower half with \( k \) smallest outputs
- For every \( u \in N(Y) \): \( \exists \) comparat. \( (u, v), v \in Y \)
- Let \( u_t, v_t \) be their keys after the comparator
  Let \( u_d, v_d \) be their keys at the output
- Note that \( v_d \in X \)
- Further: \( u_d \leq u_t \leq v_t \leq v_d \Rightarrow u_d \in X \)
- Since \( u \) was arbitrary:

\[
|Y| + |N(Y)| \leq k.
\]

Since \( G \) is a bipartite \((n, d, \mu)\)-expander:

\[
|Y| + |N(Y)| > |Y| + \min\{|\mu|Y|, n/2 - |Y|\} = \min\{(1 + \mu)|Y|, n/2\}.
\]

Combining the two bounds above yields:

\[
(1 + \mu)|Y| \leq k.
\]

Same argument \( \Rightarrow \) at most \( \epsilon \cdot k \),

\( \epsilon := 1/（\mu + 1）\), of the \( k \) largest input keys are placed in \( b_1, \ldots, b_{n/2} \).

- typical application of expander graphs in parallel algorithms
- Much more work needed to construct the AKS sorting network
Donald E. Knuth (Stanford)

“Batcher’s method is much better, unless $n$ exceeds the total memory capacity of all computers on earth!”

Richard J. Lipton (Georgia Tech)

“The AKS sorting network is galactic: it needs that $n$ be larger than $2^{78}$ or so to finally be smaller than Batcher’s network for $n$ items.”
Siblings of Sorting Network

Sorting Networks

- sorts any input of size $n$
- special case of Comparison Networks
Siblings of Sorting Network

Sorting Networks
- sorts any input of size $n$
- special case of Comparison Networks

Switching (Shuffling) Networks
- creates a random permutation of $n$ items
- special case of Permutation Networks
Siblings of Sorting Network

### Sorting Networks
- sorts any input of size $n$
- special case of Comparison Networks

### Switching (Shuffling) Networks
- creates a random permutation of $n$ items
- special case of Permutation Networks

### Counting Networks
- balances any stream of tokens over $n$ wires
- special case of Balancing Networks
Outline of this Course

Some Highlights

Introduction to Sorting Networks

Batcher’s Sorting Network

Counting Networks
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.
Distributed Counting

Processors collectively assign successive values from a given range.

Values could represent addresses in memories or destinations on an interconnection network.
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top, ...)

I. Course Intro and Sorting Networks

Counting Networks
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top, ...)

![Diagram of a balancing network showing an arrow indicating the direction of token flow.](image)
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top, ...)

I. Course Intro and Sorting Networks  Counting Networks
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,...)
Counting Network

Distributed Counting
Processors collectively assign successive values from a given range.

Balancing Networks
- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top, . . . )

I. Course Intro and Sorting Networks  Counting Networks
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top, ...)

---

I. Course Intro and Sorting Networks  Counting Networks 34
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top, ...)

I. Course Intro and Sorting Networks  Counting Networks

34
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,...)
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,...)
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,...)
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top, ...)

![Diagram of a balancing network with balancers and tokens]

I. Course Intro and Sorting Networks  Counting Networks 34
Counting Network

Distributed Counting

Processors collectively assign successive values from a given range.

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top, ...)

Number of tokens differs by at most one
1. Let $x_1, x_2, \ldots, x_n$ be the number of tokens (ever received) on the designated input wires.

2. Let $y_1, y_2, \ldots, y_n$ be the number of tokens (ever received) on the designated output wires.

Counting Network (Formal Definition)
Bitonic Counting Network

Counting Network (Formal Definition)

1. Let $x_1, x_2, \ldots, x_n$ be the number of tokens (ever received) on the designated input wires.
2. Let $y_1, y_2, \ldots, y_n$ be the number of tokens (ever received) on the designated output wires.
3. In a quiescent state: $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$.
4. A counting network is a balancing network with the **step-property**: $0 \leq y_i - y_j \leq 1$ for any $i < j$. 
1. Let $x_1, x_2, \ldots, x_n$ be the number of tokens (ever received) on the designated input wires.

2. Let $y_1, y_2, \ldots, y_n$ be the number of tokens (ever received) on the designated output wires.

3. In a quiescent state: $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$

4. A counting network is a balancing network with the **step-property**:

   \[ 0 \leq y_i - y_j \leq 1 \text{ for any } i < j. \]

**Bitonic Counting Network**: Take Batcher’s Sorting Network and replace each comparator by a balancer.
Correctness of the Bitonic Counting Network (not examinable)

<table>
<thead>
<tr>
<th>Facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let ( x_1, \ldots, x_n ) and ( y_1, \ldots, y_n ) have the step property. Then:</td>
</tr>
<tr>
<td>1. We have ( \sum_{i=1}^{n/2} x_{2i-1} = \lceil \frac{1}{2} \sum_{i=1}^{n} x_i \rceil ) and ( \sum_{i=1}^{n/2} x_{2i} = \lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \rfloor )</td>
</tr>
<tr>
<td>2. If ( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i ), then ( x_i = y_i ) for ( i = 1, \ldots, n ).</td>
</tr>
<tr>
<td>3. If ( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1 ), then ( \exists! j = 1, 2, \ldots, n ) with ( x_j = y_j + 1 ) and ( x_i = y_i ) for ( j \neq i ).</td>
</tr>
</tbody>
</table>
Correctness of the Bitonic Counting Network (not examinable)

Facts

Let \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_n \) have the step property. Then:

1. We have \( \sum_{i=1}^{n/2} x_{2i-1} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor \) and \( \sum_{i=1}^{n/2} x_{2i} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil \)

2. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \), then \( x_i = y_i \) for \( i = 1, \ldots, n \).

3. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1 \), then \( \exists j = 1, 2, \ldots, n \) with \( x_j = y_j + 1 \) and \( x_i = y_i \) for \( j \neq i \).

Key Lemma

Consider a `MERGER[n]`. Then if the inputs \( x_1, \ldots, x_{n/2} \) and \( x_{n/2+1}, \ldots, x_n \) have the step property, then so does the output \( y_1, \ldots, y_n \).

Proof (by induction on \( n \) being a power of 2)
Correctness of the Bitonic Counting Network (not examinable)

Let \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_n \) have the step property. Then:

1. We have 
   \[
   \sum_{i=1}^{n/2} x_{2i-1} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor, \quad \text{and} \quad \sum_{i=1}^{n/2} x_{2i} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil
   \]

2. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \), then \( x_i = y_i \) for \( i = 1, \ldots, n \).

3. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1 \), then \( \exists! j = 1, 2, \ldots, n \) with \( x_j = y_j + 1 \) and \( x_i = y_i \) for \( j \neq i \).

---

Proof (by induction on \( n \) being a power of 2)
Correctness of the Bitonic Counting Network (not examinable)

Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ have the step property. Then:

1. We have $\sum_{i=1}^{n/2} x_{2i-1} = \lceil \frac{1}{2} \sum_{i=1}^{n} x_i \rceil$, and $\sum_{i=1}^{n/2} x_{2i} = \lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \rfloor$

2. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, then $x_i = y_i$ for $i = 1, \ldots, n$.

3. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1$, then $\exists! j = 1, 2, \ldots, n$ with $x_j = y_j + 1$ and $x_i = y_i$ for $j \neq i$.

---

**Facts**

<table>
<thead>
<tr>
<th>$z_1$</th>
<th>$z_2$</th>
<th>$z_3$</th>
<th>$z_4$</th>
<th>$z'_1$</th>
<th>$z'_2$</th>
<th>$z'_3$</th>
<th>$z'_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_4$</td>
<td>$x_5$</td>
<td>$x_6$</td>
<td>$x_7$</td>
<td>$x_8$</td>
</tr>
</tbody>
</table>

**Proof (by induction on $n$ being a power of 2)**

- Case $n = 2$ is clear, since MERGER[2] is a single balancer
Correctness of the Bitonic Counting Network (not examinable)

Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ have the step property. Then:

1. We have $\sum_{i=1}^{n/2} x_{2i-1} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil$, and $\sum_{i=1}^{n/2} x_{2i} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor$.
2. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, then $x_i = y_i$ for $i = 1, \ldots, n$.
3. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1$, then $\exists j = 1, 2, \ldots, n$ with $x_j = y_j + 1$ and $x_i = y_i$ for $j \neq i$.

Proof (by induction on $n$ being a power of 2)

- Case $n = 2$ is clear, since MERGER[2] is a single balancer.
- $n > 2$:
Correctness of the Bitonic Counting Network (not examinable)

Facts

Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ have the step property. Then:

1. We have $\sum_{i=1}^{n/2} x_{2i-1} = \lceil \frac{1}{2} \sum_{i=1}^{n} x_i \rceil$, and $\sum_{i=1}^{n/2} x_{2i} = \lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \rfloor$
2. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, then $x_i = y_i$ for $i = 1, \ldots, n$.
3. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1$, then $\exists j = 1, 2, \ldots, n$ with $x_j = y_j + 1$ and $x_i = y_i$ for $j \neq i$.

Proof (by induction on $n$ being a power of 2)

- Case $n = 2$ is clear, since MERGER[2] is a single balancer
- $n > 2$: Let $z_1, \ldots, z_{n/2}$ and $z'_1, \ldots, z'_{n/2}$ be the outputs of the MERGER[$n/2$] subnetworks
Correctness of the Bitonic Counting Network (not examinable)

Let \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_n \) have the step property. Then:

1. We have \( \sum_{i=1}^{n/2} x_{2i-1} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor \), and \( \sum_{i=1}^{n/2} x_{2i} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil \)

2. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \), then \( x_i = y_i \) for \( i = 1, \ldots, n \).

3. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1 \), then \( \exists! j = 1, 2, \ldots, n \) with \( x_j = y_j + 1 \) and \( x_i = y_i \) for \( j \neq i \).

---

**Proof (by induction on \( n \) being a power of 2)**

- Case \( n = 2 \) is clear, since \( \text{MERGER}[2] \) is a single balancer
- \( n > 2 \): Let \( z_1, \ldots, z_{n/2} \) and \( z'_1, \ldots, z'_{n/2} \) be the outputs of the \( \text{MERGER}[n/2] \) subnetworks
Correctness of the Bitonic Counting Network (not examinable)

Facts
Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ have the step property. Then:

1. We have $\sum_{i=1}^{n/2} x_{2i-1} = \lceil \frac{1}{2} \sum_{i=1}^{n} x_i \rceil$, and $\sum_{i=1}^{n/2} x_{2i} = \lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \rfloor$.
2. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, then $x_i = y_i$ for $i = 1, \ldots, n$.
3. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1$, then $\exists! j = 1, 2, \ldots, n$ with $x_j = y_j + 1$ and $x_i = y_i$ for $j \neq i$.

Proof (by induction on $n$ being a power of 2)
- Case $n = 2$ is clear, since MERGER[2] is a single balancer
- $n > 2$: Let $z_1, \ldots, z_{n/2}$ and $z'_1, \ldots, z'_{n/2}$ be the outputs of the MERGER[n/2] subnetworks
Correctness of the Bitonic Counting Network (not examinable)

Let \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_n \) have the step property. Then:

1. We have \( \sum_{i=1}^{n/2} x_{2i-1} = \lceil \frac{1}{2} \sum_{i=1}^{n} x_i \rceil \), and \( \sum_{i=1}^{n/2} x_{2i} = \lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \rfloor \).

2. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \), then \( x_i = y_i \) for \( i = 1, \ldots, n \).

3. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1 \), then \( \exists! j = 1, 2, \ldots, n \) with \( x_j = y_j + 1 \) and \( x_i = y_i \) for \( j \neq i \).

Proof (by induction on \( n \) being a power of 2)

- Case \( n = 2 \) is clear, since MERGER[2] is a single balancer
- \( n > 2 \): Let \( z_1, \ldots, z_{n/2} \) and \( z'_1, \ldots, z'_{n/2} \) be the outputs of the MERGER[\( n/2 \)] subnetworks
Correctness of the Bitonic Counting Network (not examinable)

Let \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_n \) have the step property. Then:

1. We have \( \sum_{i=1}^{n/2} x_{2i-1} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor \) and \( \sum_{i=1}^{n/2} x_{2i} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil \)
2. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \), then \( x_i = y_i \) for \( i = 1, \ldots, n \).
3. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1 \), then \( \exists! j = 1, 2, \ldots, n \) with \( x_j = y_j + 1 \) and \( x_i = y_i \) for \( j \neq i \).

### Facts

Let \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_n \) have the step property. Then:

- Case \( n = 2 \) is clear, since MERGER[2] is a single balancer
- \( n > 2 \): Let \( z_1, \ldots, z_{n/2} \) and \( z'_1, \ldots, z'_{n/2} \) be the outputs of the MERGER[n/2] subnetworks

![Diagram](image)

Proof (by induction on \( n \) being a power of 2)
Correctness of the Bitonic Counting Network (not examinable)

Facts

Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ have the step property. Then:

1. We have $\sum_{i=1}^{n/2} x_{2i-1} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor$, and $\sum_{i=1}^{n/2} x_{2i} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil$

2. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, then $x_i = y_i$ for $i = 1, \ldots, n$.

3. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1$, then $\exists! j = 1, 2, \ldots, n$ with $x_j = y_j + 1$ and $x_i = y_i$ for $j \neq i$.

Proof (by induction on $n$ being a power of 2)

- Case $n = 2$ is clear, since MERGER[2] is a single balancer
- $n > 2$: Let $z_1, \ldots, z_{n/2}$ and $z'_1, \ldots, z'_{n/2}$ be the outputs of the MERGER[$n/2$] subnetworks
- IH $\Rightarrow z_1, \ldots, z_{n/2}$ and $z'_1, \ldots, z'_{n/2}$ have the step property
Correctness of the Bitonic Counting Network (not examinable)

Facts

Let \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_n \) have the step property. Then:

1. We have
   \[
   \sum_{i=1}^{n/2} x_{2i-1} = \lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \rfloor, \quad \text{and} \quad \sum_{i=1}^{n/2} x_{2i} = \lceil \frac{1}{2} \sum_{i=1}^{n} x_i \rceil
   \]
2. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \), then \( x_i = y_i \) for \( i = 1, \ldots, n \).
3. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1 \), then \( \exists! j = 1, 2, \ldots, n \) with \( x_j = y_j + 1 \) and \( x_i = y_i \) for \( j \neq i \).

Proof (by induction on \( n \) being a power of 2)

- Case \( n = 2 \) is clear, since MERGER[2] is a single balancer
- \( n > 2 \): Let \( z_1, \ldots, z_{n/2} \) and \( z'_1, \ldots, z'_{n/2} \) be the outputs of the MERGER[\( n/2 \)] subnetworks
- IH \( \Rightarrow \) \( z_1, \ldots, z_{n/2} \) and \( z'_1, \ldots, z'_{n/2} \) have the step property
- Let \( Z := \sum_{i=1}^{n/2} z_i \) and \( Z' := \sum_{i=1}^{n/2} z'_i \)
Let \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_n \) have the step property. Then:

1. We have
   \[
   \sum_{i=1}^{n/2} x_{2i-1} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor, \quad \text{and} \quad \sum_{i=1}^{n/2} x_{2i} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil
   \]

2. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \), then \( x_i = y_i \) for \( i = 1, \ldots, n \).

3. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1 \), then \( \exists! j = 1, 2, \ldots, n \) with \( x_j = y_j + 1 \) and \( x_i = y_i \) for \( j \neq i \).

---

**Facts**

- Let \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_n \) have the step property. Then:
  - \( \sum_{i=1}^{n/2} x_{2i-1} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor \)
  - \( \sum_{i=1}^{n/2} x_{2i} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil \)

**Proof (by induction on \( n \) being a power of 2)**

- Case \( n = 2 \) is clear, since MERGER[2] is a single balancer
- \( n > 2 \): Let \( z_1, \ldots, z_{n/2} \) and \( z'_1, \ldots, z'_{n/2} \) be the outputs of the MERGER[n/2] subnetworks
- IH \( \Rightarrow z_1, \ldots, z_{n/2} \) and \( z'_1, \ldots, z'_{n/2} \) have the step property
- Let \( Z := \sum_{i=1}^{n/2} z_i \) and \( Z' := \sum_{i=1}^{n/2} z'_i \)
- Claim: \( |Z - Z'| \leq 1 \) (since \( Z' = \left\lfloor \frac{1}{2} \sum_{i=1}^{n/2} x_i \right\rfloor + \left\lceil \frac{1}{2} \sum_{i=n/2+1}^{n} x_i \right\rceil \))
Correctness of the Bitonic Counting Network (not examinable)

Let \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_n \) have the step property. Then:

1. We have \( \sum_{i=1}^{n/2} x_{2i-1} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor \), and \( \sum_{i=1}^{n/2} x_{2i} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil \).
2. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \), then \( x_i = y_i \) for \( i = 1, \ldots, n \).
3. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1 \), then \( \exists! j = 1, 2, \ldots, n \) with \( x_j = y_j + 1 \) and \( x_i = y_i \) for \( j \neq i \).

Proof (by induction on \( n \) being a power of 2)

- Case \( n = 2 \) is clear, since \( \text{MERGER}[2] \) is a single balancer.
- \( n > 2 \): Let \( z_1, \ldots, z_{n/2} \) and \( z'_1, \ldots, z'_{n/2} \) be the outputs of the \( \text{MERGER}[n/2] \) subnetworks.
- \( \text{IH} \Rightarrow z_1, \ldots, z_{n/2} \) and \( z'_1, \ldots, z'_{n/2} \) have the step property.
- Let \( Z := \sum_{i=1}^{n/2} z_i \) and \( Z' := \sum_{i=1}^{n/2} z'_i \).
- Claim: \( |Z - Z'| \leq 1 \) (since \( Z' = \left\lfloor \frac{1}{2} \sum_{i=1}^{n/2} x_i \right\rfloor + \left\lceil \frac{1}{2} \sum_{i=n/2+1}^{n} x_i \right\rceil \)).
Correctness of the Bitonic Counting Network (not examinable)

Let \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_n \) have the step property. Then:

1. We have \( \sum_{i=1}^{n/2} x_{2i-1} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor \) and \( \sum_{i=1}^{n/2} x_{2i} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil \)
2. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \), then \( x_i = y_i \) for \( i = 1, \ldots, n \).
3. If \( \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1 \), then \( \exists! j = 1, 2, \ldots, n \) with \( x_j = y_j + 1 \) and \( x_i = y_i \) for \( j \neq i \).

Proof (by induction on \( n \) being a power of 2)

- Case \( n = 2 \) is clear, since MERGER[2] is a single balancer
- \( n > 2 \): Let \( z_1, \ldots, z_{n/2} \) and \( z'_1, \ldots, z'_{n/2} \) be the outputs of the MERGER[\( n/2 \)] subnetworks
- IH \( \Rightarrow z_1, \ldots, z_{n/2} \) and \( z'_1, \ldots, z'_{n/2} \) have the step property
- Let \( Z := \sum_{i=1}^{n/2} z_i \) and \( Z' := \sum_{i=1}^{n/2} z'_i \)
- Claim: \( |Z - Z'| \leq 1 \) (since \( Z' = \left\lfloor \frac{1}{2} \sum_{i=1}^{n/2} x_i \right\rfloor + \left\lceil \frac{1}{2} \sum_{i=n/2+1}^{n} x_i \right\rceil \))
- Case 1: If \( Z = Z' \), then F2 implies the output of MERGER[\( n \)] is \( y_i = z_{1+[(i-1)/2]} \)
Correctness of the Bitonic Counting Network (not examinable)

Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ have the step property. Then:

1. We have $\sum_{i=1}^{n/2} x_{2i-1} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor$, and $\sum_{i=1}^{n/2} x_{2i} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil$

2. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, then $x_i = y_i$ for $i = 1, \ldots, n$.

3. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1$, then $\exists! j = 1, 2, \ldots, n$ with $x_j = y_j + 1$ and $x_i = y_i$ for $j \neq i$.

Proof (by induction on $n$ being a power of 2)

- Case $n = 2$ is clear, since MERGER[2] is a single balancer
- $n > 2$: Let $z_1, \ldots, z_{n/2}$ and $z'_1, \ldots, z'_{n/2}$ be the outputs of the MERGER[n/2] subnetworks
- IH $\Rightarrow z_1, \ldots, z_{n/2}$ and $z'_1, \ldots, z'_{n/2}$ have the step property
- Let $Z := \sum_{i=1}^{n/2} z_i$ and $Z' := \sum_{i=1}^{n/2} z'_i$
- Claim: $|Z - Z'| \leq 1$ (since $Z' = \left\lfloor \frac{1}{2} \sum_{i=1}^{n/2} x_i \right\rfloor + \left\lceil \frac{1}{2} \sum_{i=n/2+1}^{n} x_i \right\rceil$)

- Case 1: If $Z = Z'$, then F2 implies the output of MERGER[n] is $y_i = z_{1+\left\lfloor (i-1)/2 \right\rfloor}$ ✓
- Case 2: If $|Z - Z'| = 1$, F3 implies $z_i = z'_i$ for $i = 1, \ldots, n/2$ except a unique $j$ with $z_j \neq z'_j$.

Balancer between $z_j$ and $z'_j$ will ensure that the step property holds.
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$.
Counting can be done as follows:
Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:
Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

I. Course Intro and Sorting Networks Counting Networks
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2 \cdot n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2n$, ...
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:
Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$.
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2 \cdot n$, ...
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2 \cdot n$, ...
Counting can be done as follows:

Add local counter to each output wire \( i \), to assign consecutive numbers \( i, i+n, i+2\cdot n, \ldots \).
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:

Add local counter to each output wire \(i\), to assign consecutive numbers \(i, i+n, i+2n, \ldots\)
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

$I$. Course Intro and Sorting Networks Counting Networks
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

$I$. Course Intro and Sorting Networks Counting Networks
Counting can be done as follows:

Add local counter to each output wire \( i \), to assign consecutive numbers \( i, i+n, i+2n, \ldots \).
Counting can be done as follows:

Add local counter to each output wire \( i \), to assign consecutive numbers \( i, i + n, i + 2n, \ldots \).
Counting can be done as follows:

Add local counter to each output wire to assign consecutive numbers.
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:
Add local counter to each output wire. 

Counting can be done as follows: add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$
Counting can be done as follows:
Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

I. Course Intro and Sorting Networks

Counting Networks
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:
Add local counter to each output wire \(i\), to assign consecutive numbers \(i\), \(i+n\), \(i+2n\), ...
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2\cdot n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:
Add local counter to each output wire \( i \), to assign consecutive numbers \( i, i + n, i + 2 \cdot n, \ldots \)
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:
Add local counter to each output wire \(i\), to assign consecutive numbers \(i, i+n, i+2n, \ldots\)
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$.
Counting can be done as follows: Add local counter to each output wire \( i \), to assign consecutive numbers \( i, i+n, i+2n, \ldots \).
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2 \cdot n$, ...

I. Course Intro and Sorting Networks

Counting Networks
Counting can be done as follows:
Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:
Add local counter to each output wire \( i \), to assign consecutive numbers \( i, i + n, i + 2n, \ldots \).
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:
Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$

I. Course Intro and Sorting Networks

Counting Networks
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:
Add local counter to each output wire \(i\), to assign consecutive numbers \(i, i+n, i+2n, \ldots\)
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2 \cdot n$, ...

I. Course Intro and Sorting Networks  Counting Networks
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:
Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:
Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

I. Course Intro and Sorting Networks Counting Networks
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:

1. Add local counter to each output wire $i$.
2. Assign consecutive numbers $i$, $i+n$, $i+2n$, ...

**Diagram:**

```
  x1  ->  y1
     |   |   |
     1  2  3
     |   |   |
  x2  ->  y2
     |   |   |
     4  5  6
     |   |   |
  x3  ->  y3
     |   |   |
  x4  ->  y4
```
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

Illustration of the bitonic counting network.
Counting can be done as follows:
Add local counter to each output wire \( i \), to assign consecutive numbers \( i, i + n, i + 2n, \ldots \)
Counting can be done as follows:

Add local counter to each output wire, to assign consecutive numbers, ..., 2·n, ...,
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

I. Course Intro and Sorting Networks

Counting Networks
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:

Add local counter to each output wire to assign consecutive numbers.
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

I. Course Intro and Sorting Networks

Counting Networks
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2 \cdot n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2 \cdot n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$
Counting can be done as follows:

1. Add local counter to each output wire $i$.
2. Assign consecutive numbers $i, i+n, i+2n, \ldots$.
Counting can be done as follows:

Add the local counter to each output wire \( i \), to assign consecutive numbers \( i \), \( i + n \), \( i + 2n \), ...
Counting can be done as follows:

Add local counter to each output wire \( i \), to assign consecutive numbers \( i, i+n, i+2n, \ldots \)
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n,...$
Counting can be done as follows:

1. Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2n$, ...

I. Course Intro and Sorting Networks

Counting Networks
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

I. Course Intro and Sorting Networks Counting Networks
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2 \cdot n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:

Add local counter to each output wire \(i\), to assign consecutive numbers \(i, i+n, i+2\cdot n, \ldots\)
Counting can be done as follows:

Add local counter to each output wire to assign consecutive numbers.
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

I. Course Intro and Sorting Networks Counting Networks
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i + n$, $i + 2 \cdot n$, ...

I. Course Intro and Sorting Networks Counting Networks
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:
Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

I. Course Intro and Sorting Networks
Counting Networks
Counting can be done as follows:

1. Add local counter to each output wire
2. Assign consecutive numbers
3. Counting Networks
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2\cdot n$, ...

I. Course Intro and Sorting Networks

Counting Networks
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$.

I. Course Intro and Sorting Networks  Counting Networks
Counting can be done as follows:
Add local counter to each output wire \( i \), to assign consecutive numbers \( i, i + n, i + 2n, \ldots \).
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...

I. Course Intro and Sorting Networks

Counting Networks
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows: Add local counter to each output wire, to assign consecutive numbers $i, i + n, i + 2 \cdot n, \ldots$. 

I. Course Intro and Sorting Networks

Counting Networks
Counting can be done as follows: Add local counter to each output wire \( i \), to assign consecutive numbers \( i, i+n, i+2n, \ldots \).
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:

Add local counter to each output wire to assign consecutive numbers.

I. Course Intro and Sorting Networks

Counting Networks
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$
Counting can be done as follows:

- Add local counter to each output wire \( i \), to assign consecutive numbers \( i, i+n, i+2n, \ldots \).
Counting can be done as follows:

Add local counter to each output wire $i$ to assign consecutive numbers $i$, $i + n$, $i + 2n$, ...
Counting can be done as follows:
Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2\cdot n$, ...
Counting can be done as follows: Add local counter to each output wire $i$, to assign consecutive numbers $i, i+n, i+2n, ...$. 

1. Course Intro and Sorting Networks
2. Counting Networks
Counting can be done as follows: Add local counter to each output wire \(i\), to assign consecutive numbers \(i, i+n, i+2n, ...\)
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:

Add local counter to each output wire $i$, to assign consecutive numbers $i$, $i+n$, $i+2n$, ...
Counting can be done as follows:
Add local counter to each output wire $i$ to assign consecutive numbers $i, i+n, i+2n, ...$
Bitonic Counting Network in Action (Asynchronous Execution)

Counting can be done as follows:
Add local counter to each output wire \(i\), to assign consecutive numbers
\(i, i+n, i+2n, \ldots\)
Counting can be done as follows:
Add **local counter** to each output wire \( i \), to assign consecutive numbers \( i, i + n, i + 2 \cdot n, \ldots \)
A Periodic Counting Network [Aspnes, Herlihy, Shavit, JACM 1994]

Consists of $\log n$ LOCK networks each of which has depth $\log n$. 

I. Course Intro and Sorting Networks

Counting Networks
A Periodic Counting Network [Aspnes, Herlihy, Shavit, JACM 1994]

Consists of $\log n$ BLOCK[$n$] networks each of which has depth $\log n$
Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.
The converse is not true!

**Counting vs. Sorting**

If a network is a counting network, then it is also a sorting network.
Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.
If a network is a counting network, then it is also a sorting network.

Proof.

- Let $C$ be a counting network, and $S$ be the corresponding sorting network.

Diagram:

```
  C
```

The converse is not true!
Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let $C$ be a counting network, and $S$ be the corresponding sorting network.
From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.
- Let $C$ be a counting network, and $S$ be the corresponding sorting network
- Consider an input sequence $a_1, a_2, \ldots, a_n \in \{0, 1\}^n$ to $S$
From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let $C$ be a counting network, and $S$ be the corresponding sorting network
- Consider an input sequence $a_1, a_2, \ldots, a_n \in \{0, 1\}^n$ to $S$
Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let $C$ be a counting network, and $S$ be the corresponding sorting network.
- Consider an input sequence $a_1, a_2, \ldots, a_n \in \{0, 1\}^n$ to $S$.
- Define an input $x_1, x_2, \ldots, x_n \in \{0, 1\}^n$ to $C$ by $x_i = 1$ iff $a_i = 0$. 

![Diagram](image)
From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let \( C \) be a counting network, and \( S \) be the corresponding sorting network.
- Consider an input sequence \( a_1, a_2, \ldots, a_n \in \{0, 1\}^n \) to \( S \).
- Define an input \( x_1, x_2, \ldots, x_n \in \{0, 1\}^n \) to \( C \) by \( x_i = 1 \) iff \( a_i = 0 \).
- \( C \) is a counting network \( \Rightarrow \) all ones will be routed to the lower wires.

\[
\begin{array}{c}
\text{C} \\
0 \\
1 \\
1 \\
0 \\
\end{array} \quad \begin{array}{c}
\text{S} \\
1 \\
0 \\
0 \\
1 \\
\end{array}
\]
From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let $C$ be a counting network, and $S$ be the corresponding sorting network.
- Consider an input sequence $a_1, a_2, \ldots, a_n \in \{0, 1\}^n$ to $S$.
- Define an input $x_1, x_2, \ldots, x_n \in \{0, 1\}^n$ to $C$ by $x_i = 1$ iff $a_i = 0$.
- $C$ is a counting network $\Rightarrow$ all ones will be routed to the lower wires.

![Diagram of counting network $C$ and sorting network $S$]

I. Course Intro and Sorting Networks

Counting Networks
From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let \( C \) be a counting network, and \( S \) be the corresponding sorting network.
- Consider an input sequence \( a_1, a_2, \ldots, a_n \in \{0, 1\}^n \) to \( S \).
- Define an input \( x_1, x_2, \ldots, x_n \in \{0, 1\}^n \) to \( C \) by \( x_i = 1 \) iff \( a_i = 0 \).
- \( C \) is a counting network \( \Rightarrow \) all ones will be routed to the lower wires.

\[
\begin{array}{cccc}
& 0 & 1 & 1 \\
\hline
1 & 0 & 0 \\
\hline
1 & 1 & 1 \\
\hline
0 & 0 & 0 \\
\end{array} \quad \begin{array}{cccc}
1 \\
\hline
0 \\
\hline
1 \\
\hline
1 \\
\end{array}
\]

C \quad S
From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let $C$ be a counting network, and $S$ be the corresponding sorting network.
- Consider an input sequence $a_1, a_2, \ldots, a_n \in \{0, 1\}^n$ to $S$.
- Define an input $x_1, x_2, \ldots, x_n \in \{0, 1\}^n$ to $C$ by $x_i = 1$ iff $a_i = 0$.
- $C$ is a counting network $\Rightarrow$ all ones will be routed to the lower wires.

\[ C \]
\[
\begin{array}{cccc}
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 \\
\end{array}
\]

\[ S \]
\[
\begin{array}{cccc}
1 \\
0 \\
0 \\
1 \\
\end{array}
\]
From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let $C$ be a counting network, and $S$ be the corresponding sorting network.
- Consider an input sequence $a_1, a_2, \ldots, a_n \in \{0, 1\}^n$ to $S$.
- Define an input $x_1, x_2, \ldots, x_n \in \{0, 1\}^n$ to $C$ by $x_i = 1$ iff $a_i = 0$.
- $C$ is a counting network $\Rightarrow$ all ones will be routed to the lower wires.
From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let $C$ be a counting network, and $S$ be the corresponding sorting network.
- Consider an input sequence $a_1, a_2, \ldots, a_n \in \{0, 1\}^n$ to $S$.
- Define an input $x_1, x_2, \ldots, x_n \in \{0, 1\}^n$ to $C$ by $x_i = 1$ iff $a_i = 0$.
- $C$ is a counting network $\Rightarrow$ all ones will be routed to the lower wires.
- $S$ corresponds to $C$ $\Rightarrow$ all zeros will be routed to the lower wires.

\[ 
\begin{array}{cccccc}
0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{array} 
\]

\[ 
\begin{array}{cc}
1 & \\
0 & \\
0 & \\
1 & \\
\end{array} 
\]
From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let $C$ be a counting network, and $S$ be the corresponding sorting network.
- Consider an input sequence $a_1, a_2, \ldots, a_n \in \{0, 1\}^n$ to $S$.
- Define an input $x_1, x_2, \ldots, x_n \in \{0, 1\}^n$ to $C$ by $x_i = 1$ iff $a_i = 0$.
- $C$ is a counting network $\Rightarrow$ all ones will be routed to the lower wires.
- $S$ corresponds to $C$ $\Rightarrow$ all zeros will be routed to the lower wires.
From Counting to Sorting

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let $C$ be a counting network, and $S$ be the corresponding sorting network.
- Consider an input sequence $a_1, a_2, \ldots, a_n \in \{0, 1\}^n$ to $S$.
- Define an input $x_1, x_2, \ldots, x_n \in \{0, 1\}^n$ to $C$ by $x_i = 1$ iff $a_i = 0$.
- $C$ is a counting network $\Rightarrow$ all ones will be routed to the lower wires.
- $S$ corresponds to $C$ $\Rightarrow$ all zeros will be routed to the lower wires.
- By the Zero-One Principle, $S$ is a sorting network.

\[\begin{array}{cccccc}
0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}\]

\[\begin{array}{cccccc}
1 \\
0 \\
0 \\
1 \\
\end{array}\]