Sorting Network*

Xiaofeng Gao

Department of Computer Science and Engineering
Shanghai Jiao Tong University, P.R.China

Algorithm Course @ Shanghai Jiao Tong University

* Special thanks is given to Mr. Jiajun Tang from CS2015@SJTU (`https://github.com/yelantingfeng`) for drawing SortingNetwork with Python Tkinter.

Outline

1. Basic Concepts
   - History
   - Comparison Network
   - Sorting Network
2. Zero-One Principle
   - Domain Conversion Lemma
   - Zero-One Principle
3. Construction of a Sorting Network
   - Bitonic Sorter
   - Merger
   - Sorter

Source and Story

Sorting Network is from Chapter 27 of CLRS book (2nd Edition), which has been replaced by *Multithreaded Algorithm* in the 3rd Edition. Its main 0-1 principle has been merged into Problem 8-7 of the new Chapter 8 (Sorting in Linear Time).

In this class, we choose this topic because:

- It implements beautiful triple divide-and-conquer techniques to achieve a parallel sorting process.
- Its 0-1 principle introduces a reduction method to solve hard problems by easy alternatives.
- It can be exhibited interestingly by many modern visualization tools like Python Tkinter.

In Memory of Algorithm Class

CS6363 Computer Algorithm

Prof. Ivan Sudborough

Founders Professor of Erik Jonsson School of Engineering and Computer Science

The University of Texas at Dallas

"An easily described, easily communicated problem is invaluable for engaging a wide array of participants, from high school students to the most eminent mathematicians ...... I want to be remembered for contributing to the body of knowledge in my discipline. I've found solutions to open problems that nobody knew how to solve."

—-Ivan Sudborough

Introduction

We examined sorting algorithms based on

- ▷ (before) serial computers (random-access machines, RAM's)
  → allow only one operation to be executed at a time.
- ▷ (now) comparison-network
  → n comparison operations can be performed simultaneously.

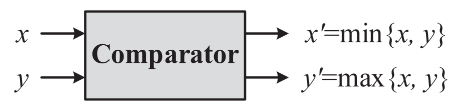
Comparison Network **VS** RAM's

- ▷ Comparison network can only perform comparisons.
  (Cannot deal with Counting Sort etc.)
- ▷ Comparison network runs parallel operations.

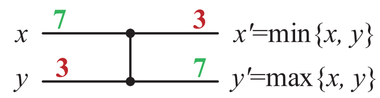
Definition

comparison network: composed solely of comparators and wires.

comparator: device with two inputs x and y, and two outputs x' and y', that performs the following function:



A comparator



Logical representation

Each comparator operates in $O(1)$ time.

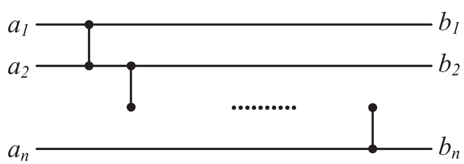
wire: transmits a value from place to place.

- ▷ Connect the output of one comparator to the input of another;
- ▷ The network input wires or output wires.

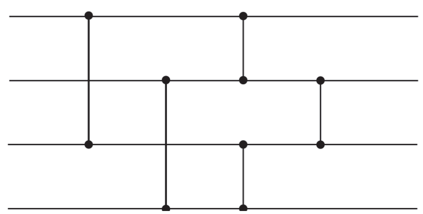
Objective of Comparison Network

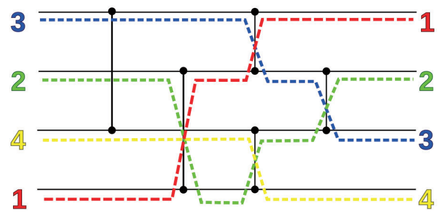
Assume a **comparison network** contains n input wires $\langle a_1, a_2, \cdots, a_n \rangle$, through which the values to be sorted enter the network, and n output wires $\langle b_1, b_2, \cdots, b_n \rangle$, which produce the results computed by the network.

Goal: Draw a comparison network on n inputs as a collection of n horizontal lines with comparators stretched vertically.



An Example





Data move from left to right.

Interconnections must be acyclic.

If a comparator has two input wires with **depths** d_x and d_y, then its output wire have depth $\max\{d_x, d_y\} + 1$. (Initially is 0)

Sorting Network

A sorting network is a comparison network for which the output sequence is monotonically increasing ($b_1 \leq b_2 \leq \cdots \leq b_n$) for **every** input sequence.

Note: We are discussing a family of comparison networks according to the input size.

Zero-One Principle

Zero-One Principle: if a sorting network works correctly with inputs drawn from **{0, 1}**, then it works correctly on arbitrary input numbers (e.g., integers, reals, or any linearly ordered set).

Domain Conversion Lemma: If a comparison network transforms the input sequence $\mathbf{a} = \langle a_1, a_2, \cdots, a_n \rangle$ into the output sequence $\mathbf{b} = \langle b_1, b_2, \cdots, b_n \rangle$, then for any monotonically increasing function f, the network transforms the input sequence $f(\mathbf{a}) = \langle f(a_1), f(a_2), \cdots, f(a_n) \rangle$ into the output sequence $f(\mathbf{b}) = \langle f(b_1), f(b_2), \cdots, f(b_n) \rangle$.

Proof (by Induction)

Basis Step: Consider a comparator whose input values are x and y. The upper output is $\min\{x, y\}$ while the lower output is $\max\{x, y\}$.

If we apply $f(x)$ and $f(y)$ as the inputs, the operation of the comparator yields the value of upper $\min\{f(x), f(y)\}$ and lower $\max\{f(x), f(y)\}$.

Since f is monotonically increasing, $x \leq y$ implies $f(x) \leq f(y)$. Thus we have

$$\min\{f(x), f(y)\} = f(\min\{x, y\}),$$
$$\max\{f(x), f(y)\} = f(\max\{x, y\}),$$

which completes the proof of the claim as the base case.

Proof (Continued)

We use induction on the depth of each wire in a general comparison network to prove a stronger result:

A Stronger Statement: If a wire assumes the value a_i when the input sequence is $\mathbf{a}$, then it assumes the value $f(a_i)$ when the input sequence is $f(\mathbf{a})$.

Since the output wires are included in this statement, proving it will prove the lemma.

Proof (Continued)

Basis: A wire at depth 0 is an input wire a_i. When $f(\mathbf{a})$ is applied to the network, the input wire carries $f(a_i)$.

Induction: A wire at depth $d \geq 1$ is the output of a comparator at depth d, and the input wires to this comparator are at a depth strictly less than d. By inductive hypothesis, if the input wires carry values a_i and a_j with input sequence $\mathbf{a}$, then they carry $f(a_i)$ and $f(a_j)$ with input sequence $f(\mathbf{a})$.

By previous claim, the output wires of this comparator then carry $f(\min\{a_i, a_j\})$ and $f(\max\{a_i, a_j\})$. Since the carry $\min\{a_i, a_j\}$ and $\max\{a_i, a_j\}$ when the input sequence is $\mathbf{a}$, the lemma is proved.

An Example

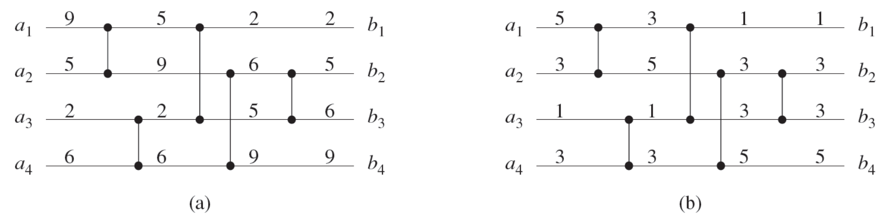


Figure 27.5 **(a)** The sorting network from Figure 27.2 with input sequence $\langle 9, 5, 2, 6\rangle$. **(b)** The same sorting network with the monotonically increasing function $f(x) = \lceil x/2 \rceil$ applied to the inputs. Each wire in this network has the value of f applied to the value on the corresponding wire in (a).

Zero-One Principle

Theorem: 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: (Contradiction) Suppose there exists a sequence of arbitrary numbers that the network does not correctly sort. That is, there exists an input sequence $\langle a_1, a_2, \cdots, a_n\rangle$ containing elements a_i and a_j, such that $a_i < a_j$, but the network places a_j before a_i in the output sequence.

Proof (Continued)

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 previous lemma, it will place $f(a_j)$ before $f(a_i)$ in the output sequence when $\langle f(a_1), f(a_2), \cdots, f(a_n)\rangle$ is input.

However, since $f(a_j) = 1$ and $f(a_i) = 0$, the network fails to sort the zero-one sequence $\langle f(a_1), f(a_2), \cdots, f(a_n)\rangle$ correctly.

A contradiction!

Construction of a Sorting Network

To construct a sorting network, we need three steps:

Step 1: Construct a Bitonic Sorter $\Rightarrow$ to sort bitonic sequence.

Step 2: Construct a Merger $\Rightarrow$ to merge two sorted sequence.

Step 3: Construct a Sorter $\Rightarrow$ to sort an arbitrary sequence.

Step 1: Construct a Bitonic Sorter

We start from bitonic sequence.

A **bitonic Sequence** is a sequence that monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Examples: $\langle 1,4,6,8,3,2\rangle$, $\langle 6,9,4,2,3,5\rangle$, $\langle 9,8,3,2,4,6\rangle$

Zero-one bitonic sequence have the form $0^i1^j0^k$ or the form $1^i0^j1^k$.

A monotonically increasing or monotonically decreasing sequence is also bitonic.

Half-Cleaner

A half-cleaner is a comparison network of depth 1, in which input line i is compared with line $i+\frac{n}{2}$ for $i=1,2,\cdots,\frac{n}{2}$ (assume n is even).

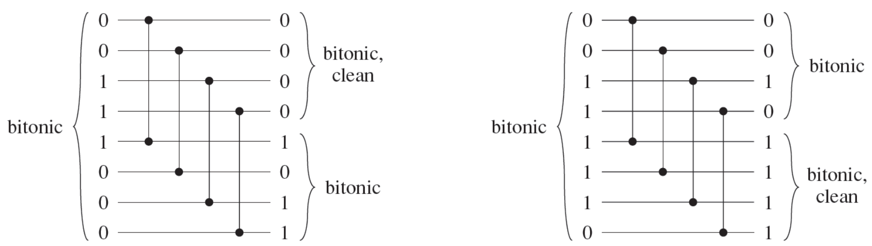


Figure 27.7 The comparison network HALF-CLEANER[8]. Two different sample zero-one input and output values are shown. The input is assumed to be bitonic. A half-cleaner ensures that every output element of the top half is at least as small as every output element of the bottom half. Moreover, both halves are bitonic, and at least one half is clean.

Half-Cleaner Lemma

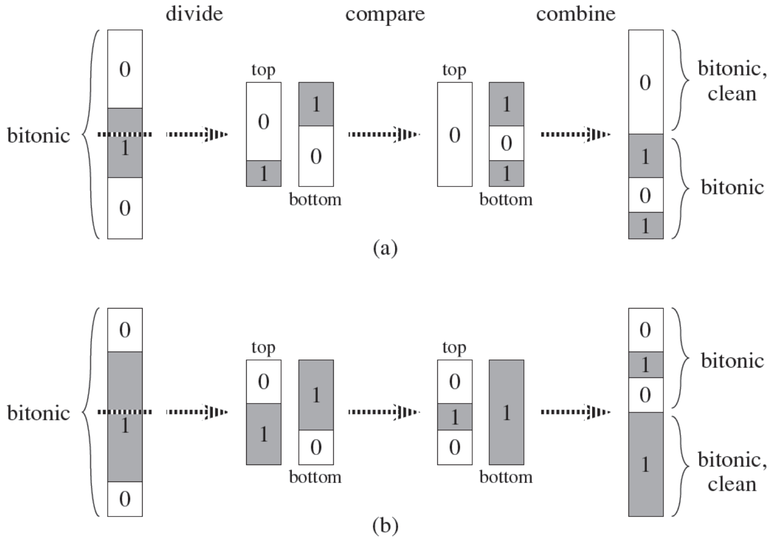
Lemma: If the input to a half-cleaner is a bitonic sequence of 0's and 1's, then the output satisfies the following properties:

- ▷ 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.

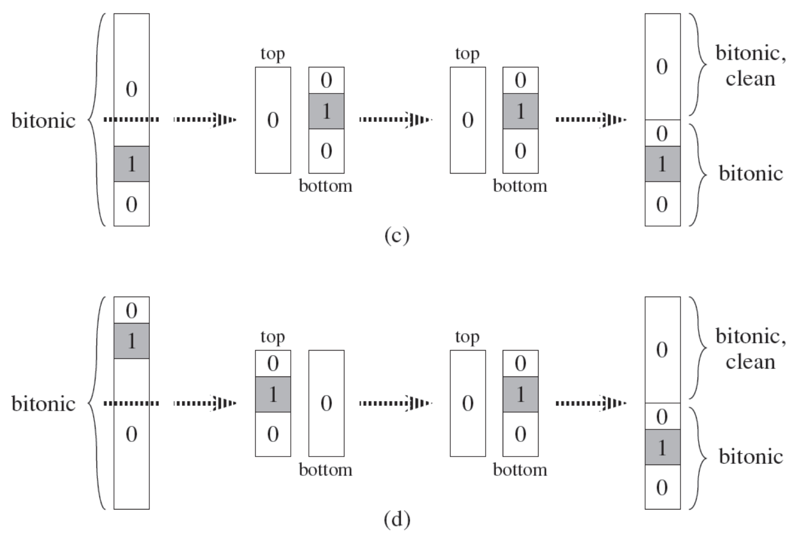
Proof: HALF-CLEANER[n] compares inputs i and $i+n/2$ for $i=1,2,\cdots,n/2$. Without loss of generality, suppose the input is of the form $00\cdots011\cdots100\cdots0$ (the situation of $11\cdots100\cdots011\cdots1$ are symmetric).

Note: There are three possible cases depending upon the block of consecutive 0's and 1's in which the midpoint $n/2$ falls, and one of these cases is further split into two cases.

Case Analysis

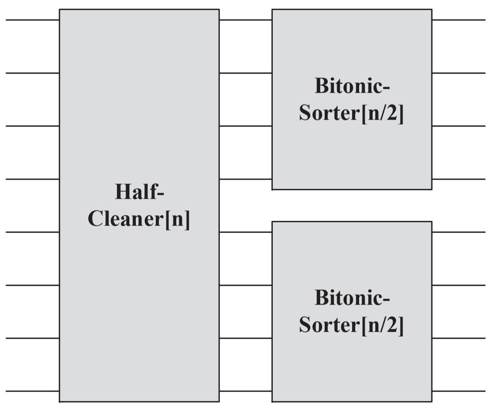


Case Analysis (Cont.)



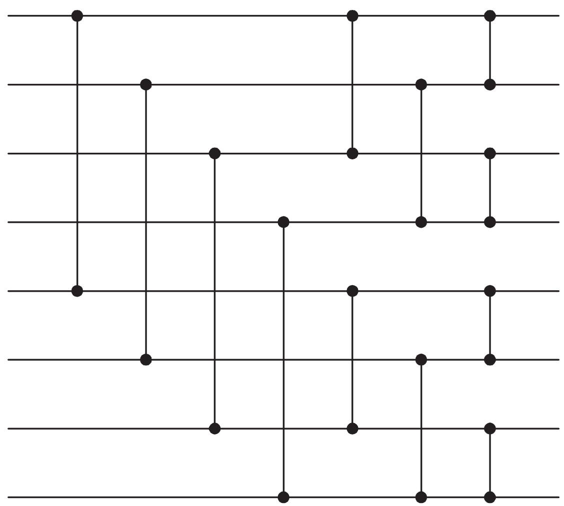
The Bitonic Sorter

By recursively combing half-cleaners, we can build a bitonic sorter, which is a network that sorts bitonic sequences.



Bitonic-Sorter[n]

An Example of $n = 8$



Depth $D(n)$

The depth $D(n)$ of BITONIC-SORTER$[n]$ is given by the recurrence

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

Easy to see, $D(n) = O(\log n)$.

Thus, a zero-one bitonic sequence can be sorted by BITONIC-SORTER$[n]$, which has a depth of $\log n$.

By zero-one principle, any bitonic sequence of arbitrary numbers can be sorted by this network.

Step 2: Construct a Merger

Merging Network can merge two sorted input sequences into one sorted output sequence.

Given two sorted sequences, if we reverse the order of the second sequence and then concatenate the two sequences, the resulting sequence is bitonic.

For instance:

$$\begin{aligned} X &= 00000111; \\ Y &= 00001111; \\ Y^R &= 11110000; \\ X \circ Y^R &= 0000011111110000. \end{aligned}$$

MERGER$[n]$

Given two sorted sequences $\langle a_1, a_2, \cdots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \cdots, a_n \rangle$, we want the effect of bitonically sorting the sequence $\langle a_1, a_2, \cdots, a_{n/2}, a_n, a_{n-1}, \cdots, a_{n/2+1} \rangle$.

Since the first half-cleaner of BITONIC-SORTER$[n]$ compares inputs i with $n/2 + i$, for $i = 1, 2, \cdots, n/2$, we make the first stage of the merging network compare inputs i and $n - i + 1$.

The order of the outputs from the bottom of the first stage of MERGER$[n]$ are reversed compared with the order of outputs from an ordinary half-cleaner.

Comparison between MERGER$[n]$ and HALF-CLEANER$[n]$

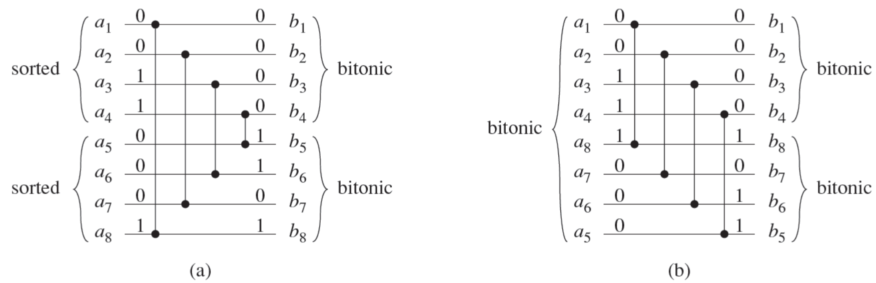
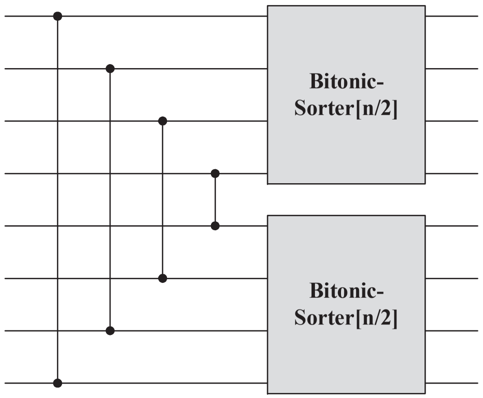


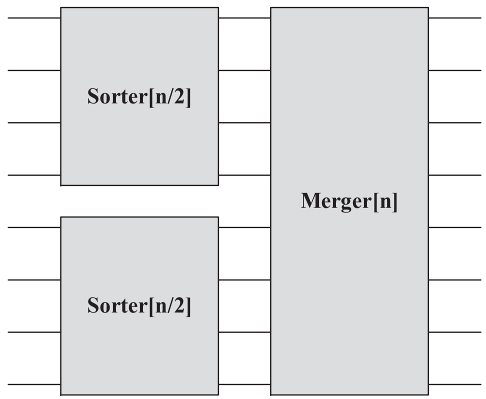
Figure 27.10 Comparing the first stage of MERGER$[n]$ with HALF-CLEANER$[n]$, for $n = 8$. **(a)** The first stage of MERGER$[n]$ transforms the two monotonic input sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$ into two bitonic sequences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

MERGER[n]

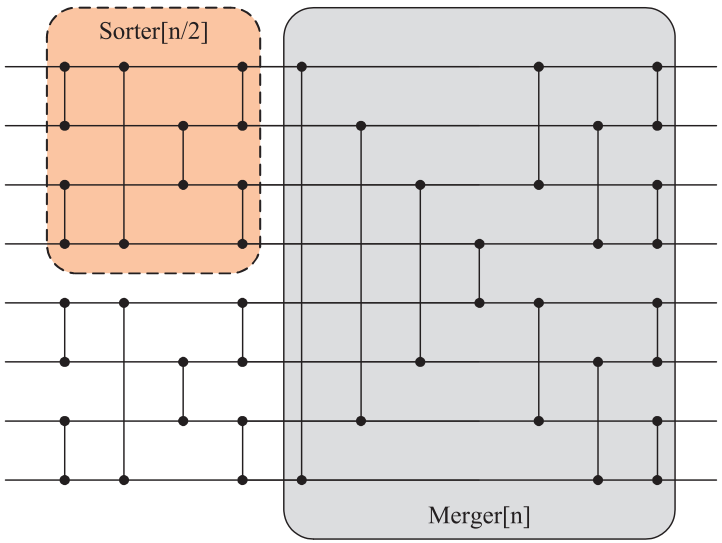


Step 3: Construct a Sorter

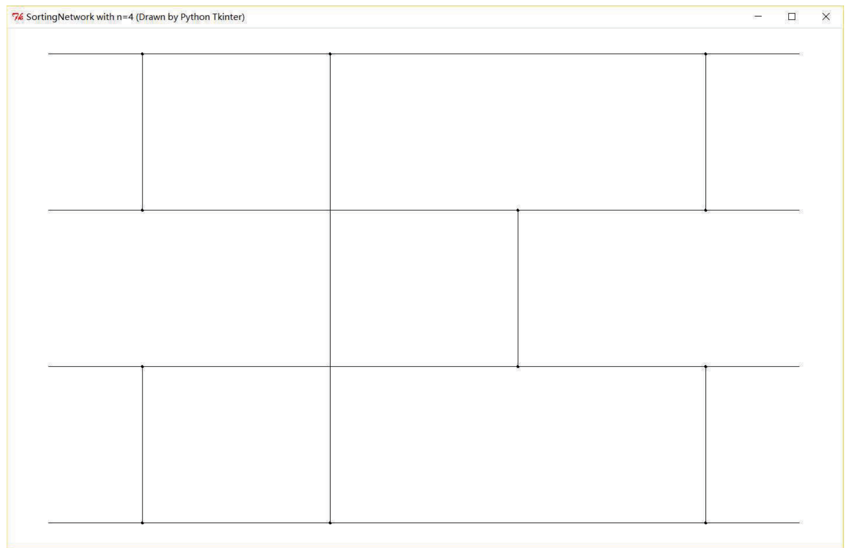
The sorting network SORTER[n] are composed by two copies of SORTER[$n/2$] and one MERGER[n] recursively.



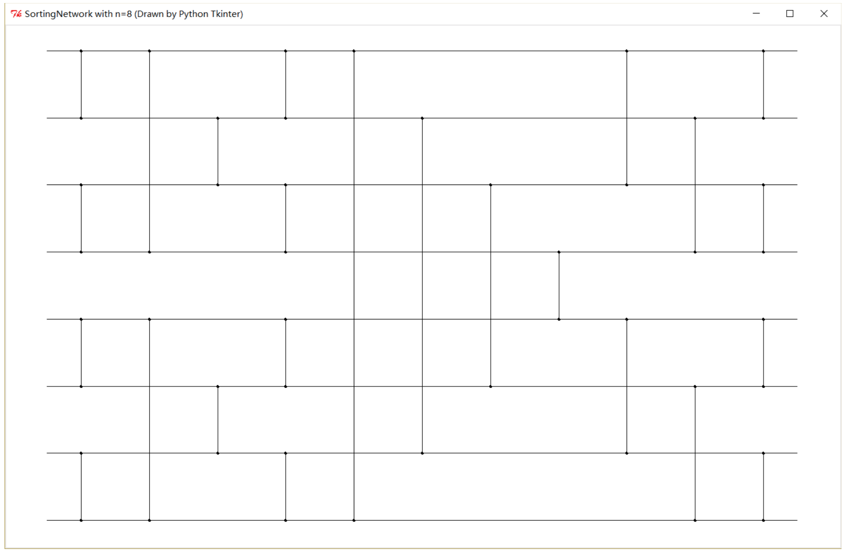
An Example with $n = 8$



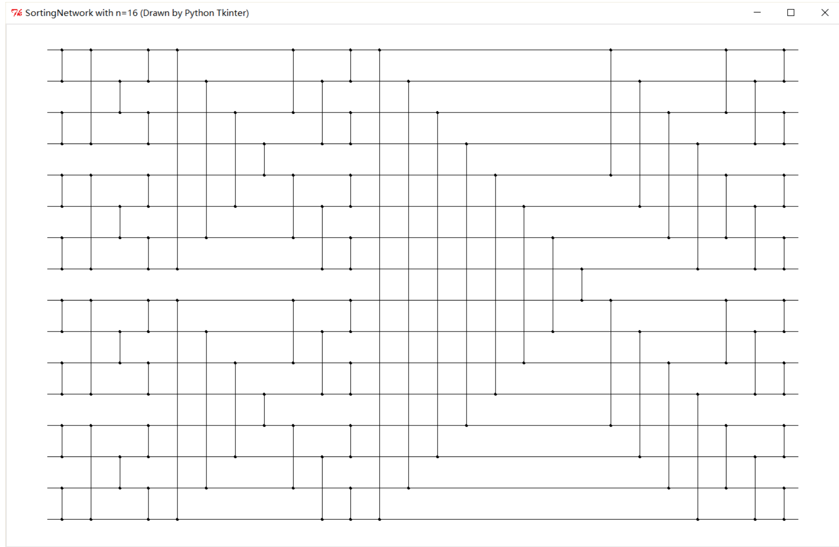
Growing Trend (Drawn by Python with $n = 4$)



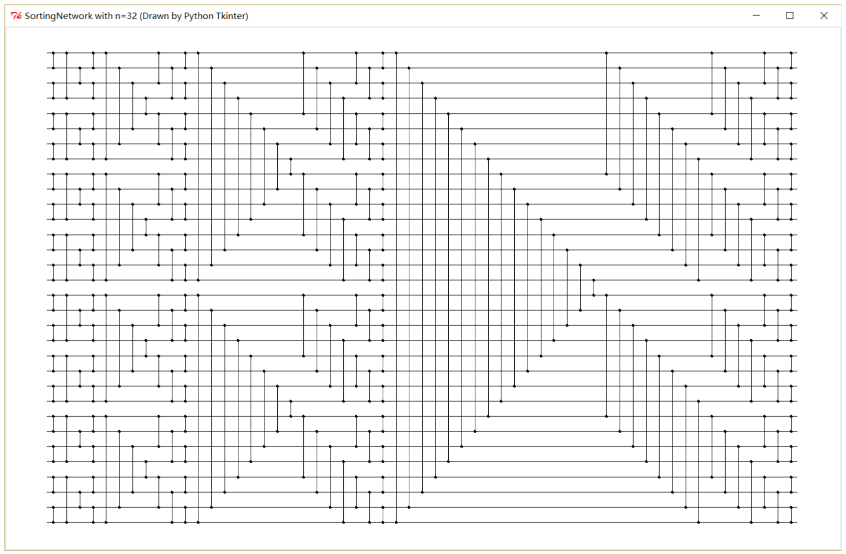
Growing Trend (Drawn by Python with $n = 8$)



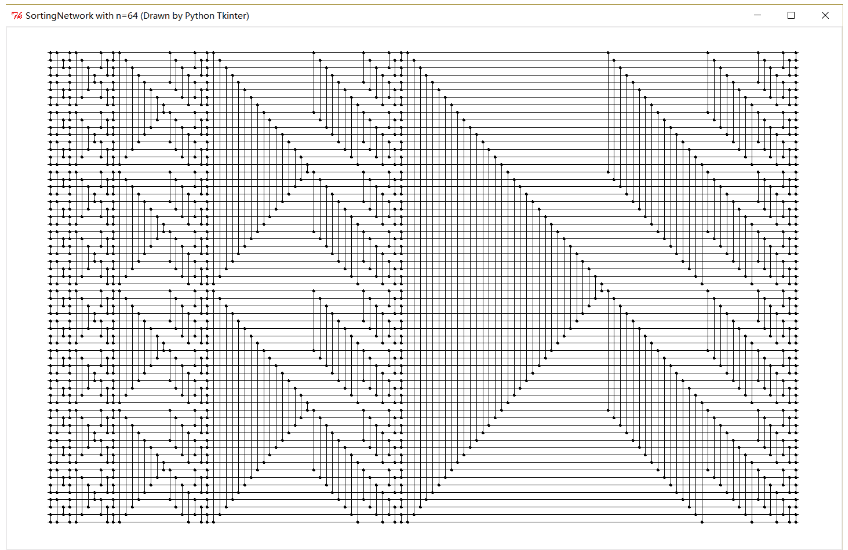
Growing Trend (Drawn by Python with $n = 16$)



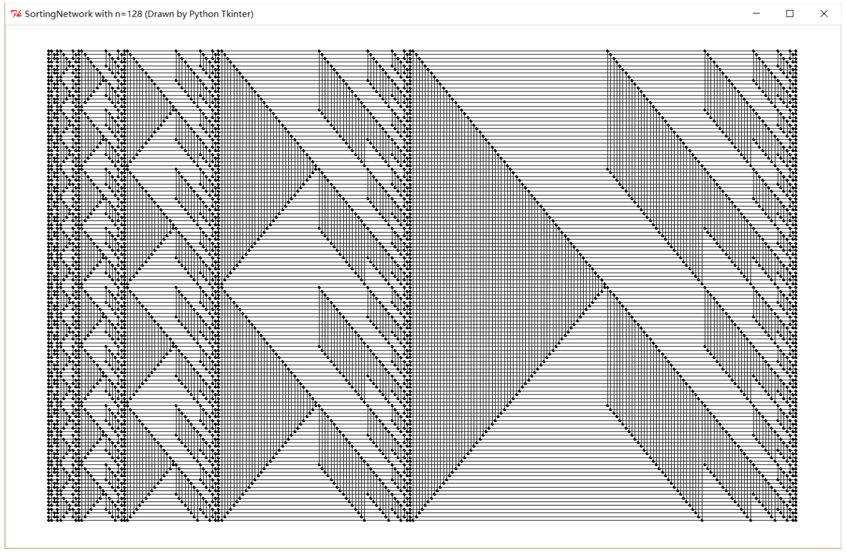
Growing Trend (Drawn by Python with $n = 32$)



Growing Trend (Drawn by Python with $n = 64$)



Growing Trend (Drawn by Python with $n = 128$)



Performance Analysis

The depth $D(n)$ of SORTER$[n]$ is the depth $D(n/2)$ of SORTER$[n/2]$ plus the depth $\ln n$ of MERGER$[n]$.

Consequently, the depth of SORTER$[n]$ is given by the recurrence

$$D(n) = \begin{cases} 0 & \text{if } n = 1; \\ D(n/2) + O(\log n) & \text{if } n \geq 1. \end{cases}$$

By recurrence computation, the solution is $D(n) = O(\log^2 n)$. Thus we can sort n numbers in parallel in $O(\log^2 n)$ time.