27 Sorting Networks

In Part II, we examined sorting algorithms for serial computers (random-access machines, or RAM's) that allow only one operation to be executed at a time. In this chapter, we investigate sorting algorithms based on a comparison-network model of computation, in which many comparison operations can be performed simultaneously.

Comparison networks differ from RAM's in two important respects. First, they can only perform comparisons. Thus, an algorithm such as counting sort (see Section 8.2) cannot be implemented on a comparison network. Second, unlike the RAM model, in which operations occur serially—that is, one after another—operations in a comparison network may occur at the same time, or "in parallel." As we shall see, this characteristic allows the construction of comparison networks that sort n values in sublinear time.

We begin in Section 27.1 by defining comparison networks and sorting networks. We also give a natural definition for the "running time" of a comparison network in terms of the depth of the network. Section 27.2 proves the "zero-one principle," which greatly eases the task of analyzing the correctness of sorting networks.

The efficient sorting network that we shall design is essentially a parallel version of the merge-sort algorithm from Section 2.3.1. Our construction will have three steps. Section 27.3 presents the design of a "bitonic" sorter that will be our basic building block. We modify the bitonic sorter slightly in Section 27.4 to produce a merging network that can merge two sorted sequences into one sorted sequence. Finally, in Section 27.5, we assemble these merging networks into a sorting network that can sort n values in $O(\lg^2 n)$ time.

27.1 Comparison networks

Sorting networks are comparison networks that always sort their inputs, so it makes sense to begin our discussion with comparison networks and their characteristics.

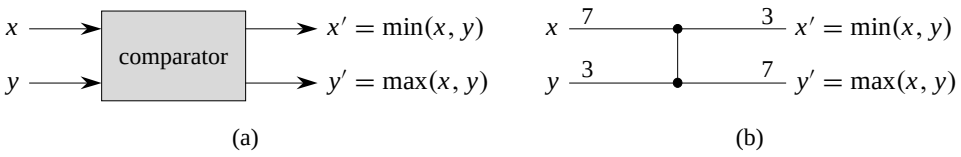

Figure 27.1 **(a)** A comparator with inputs x and y and outputs x' and y'. **(b)** The same comparator, drawn as a single vertical line. Inputs $x = 7$, $y = 3$ and outputs $x' = 3$, $y' = 7$ are shown.

A comparison network is composed solely of wires and comparators. A ***comparator***, shown in Figure 27.1(a), is a device with two inputs, x and y, and two outputs, x' and y', that performs the following function:

$$\begin{aligned} x' &= \min(x, y) , \\ y' &= \max(x, y) . \end{aligned}$$

Because the pictorial representation of a comparator in Figure 27.1(a) is too bulky for our purposes, we shall adopt the convention of drawing comparators as single vertical lines, as shown in Figure 27.1(b). Inputs appear on the left and outputs on the right, with the smaller input value appearing on the top output and the larger input value appearing on the bottom output. We can thus think of a comparator as sorting its two inputs.

We shall assume that each comparator operates in $O(1)$ time. In other words, we assume that the time between the appearance of the input values x and y and the production of the output values x' and y' is a constant.

A ***wire*** transmits a value from place to place. Wires can connect the output of one comparator to the input of another, but otherwise they are either network input wires or network output wires. Throughout this chapter, we shall assume that a comparison network contains n ***input wires*** $a_1, a_2, \ldots, a_n$, through which the values to be sorted enter the network, and n ***output wires*** $b_1, b_2, \ldots, b_n$, which produce the results computed by the network. Also, we shall speak of the ***input sequence*** $\langle a_1, a_2, \ldots, a_n \rangle$ and the ***output sequence*** $\langle b_1, b_2, \ldots, b_n \rangle$, referring to the values on the input and output wires. That is, we use the same name for both a wire and the value it carries. Our intention will always be clear from the context.

Figure 27.2 shows a ***comparison network***, which is a set of comparators interconnected by wires. We draw a comparison network on n inputs as a collection of n horizontal ***lines*** with comparators stretched vertically. Note that a line does *not* represent a single wire, but rather a sequence of distinct wires connecting various comparators. The top line in Figure 27.2, for example, represents three wires: input wire a_1, which connects to an input of comparator A; a wire connecting the top output of comparator A to an input of comparator C; and output wire b_1, which comes from the top output of comparator C. Each comparator input is connected

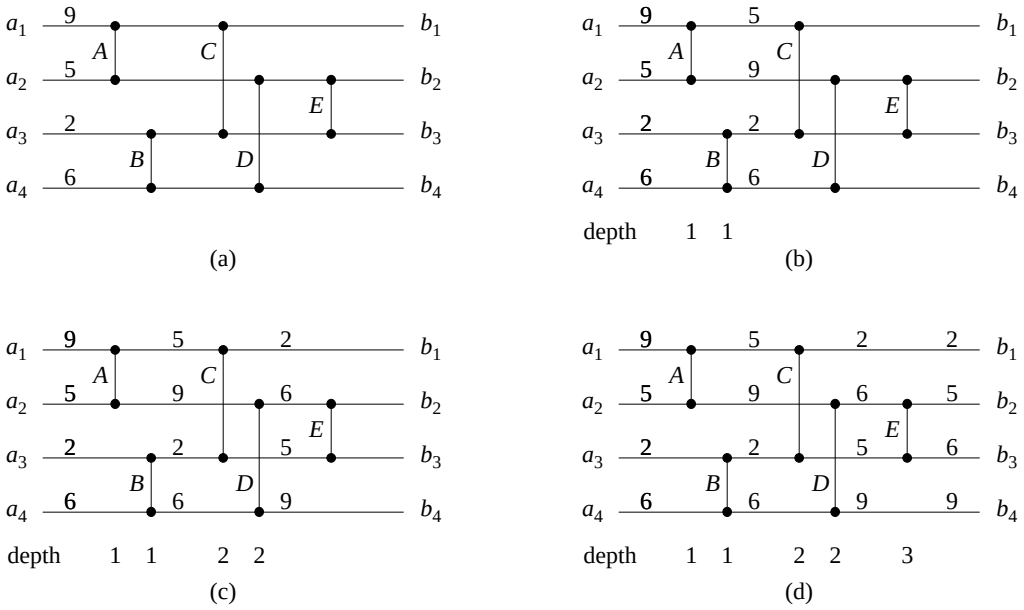

Figure 27.2 **(a)** A 4-input, 4-output comparison network, which is in fact a sorting network. At time 0, the input values shown appear on the four input wires. **(b)** At time 1, the values shown appear on the outputs of comparators A and B, which are at depth 1. **(c)** At time 2, the values shown appear on the outputs of comparators C and D, at depth 2. Output wires b_1 and b_4 now have their final values, but output wires b_2 and b_3 do not. **(d)** At time 3, the values shown appear on the outputs of comparator E, at depth 3. Output wires b_2 and b_3 now have their final values.

to a wire that is either one of the network's n input wires $a_1, a_2, \ldots, a_n$ or is connected to the output of another comparator. Similarly, each comparator output is connected to a wire that is either one of the network's n output wires $b_1, b_2, \ldots, b_n$ or is connected to the input of another comparator. The main requirement for interconnecting comparators is that the graph of interconnections must be acyclic: if we trace a path from the output of a given comparator to the input of another to an output to an input, etc., the path we trace must never cycle back on itself and go through the same comparator twice. Thus, as in Figure 27.2, we can draw a comparison network with network inputs on the left and network outputs on the right; data move through the network from left to right.

Each comparator produces its output values only when both of its input values are available to it. In Figure 27.2(a), for example, suppose that the sequence $\langle 9, 5, 2, 6 \rangle$ appears on the input wires at time 0. At time 0, then, only comparators A and B have all their input values available. Assuming that each comparator requires one time unit to compute its output values, comparators A and B produce their outputs at time 1; the resulting values are shown in Figure 27.2(b). Note

that comparators A and B produce their values at the same time, or "in parallel." Now, at time 1, comparators C and D, but not E, have all their input values available. One time unit later, at time 2, they produce their outputs, as shown in Figure 27.2(c). Comparators C and D operate in parallel as well. The top output of comparator C and the bottom output of comparator D connect to output wires b_1 and b_4, respectively, of the comparison network, and these network output wires therefore carry their final values at time 2. Meanwhile, at time 2, comparator E has its inputs available, and Figure 27.2(d) shows that it produces its output values at time 3. These values are carried on network output wires b_2 and b_3, and the output sequence $\langle 2, 5, 6, 9 \rangle$ is now complete.

Under the assumption that each comparator takes unit time, we can define the "running time" of a comparison network, that is, the time it takes for all the output wires to receive their values once the input wires receive theirs. Informally, this time is the largest number of comparators that any input element can pass through as it travels from an input wire to an output wire. More formally, we define the ***depth*** of a wire as follows. An input wire of a comparison network has depth 0. Now, if a comparator has two input wires with depths d_x and d_y, then its output wires have depth $\max(d_x, d_y) + 1$. Because there are no cycles of comparators in a comparison network, the depth of a wire is well defined, and we define the depth of a comparator to be the depth of its output wires. Figure 27.2 shows comparator depths. The depth of a comparison network is the maximum depth of an output wire or, equivalently, the maximum depth of a comparator. The comparison network of Figure 27.2, for example, has depth 3 because comparator E has depth 3. If each comparator takes one time unit to produce its output value, and if network inputs appear at time 0, a comparator at depth d produces its outputs at time d; the depth of the network therefore equals the time for the network to produce values at all of its output wires.

A ***sorting network*** is a comparison network for which the output sequence is monotonically increasing (that is, $b_1 \leq b_2 \leq \cdots \leq b_n$) for *every* input sequence. Of course, not every comparison network is a sorting network, but the network of Figure 27.2 is. To see why, observe that after time 1, the minimum of the four input values has been produced by either the top output of comparator A or the top output of comparator B. After time 2, therefore, it must be on the top output of comparator C. A symmetrical argument shows that after time 2, the maximum of the four input values has been produced by the bottom output of comparator D. All that remains is for comparator E to ensure that the middle two values occupy their correct output positions, which happens at time 3.

A comparison network is like a procedure in that it specifies how comparisons are to occur, but it is unlike a procedure in that its ***size***—the number of comparators that it contains—depends on the number of inputs and outputs. Therefore, we shall actually be describing "families" of comparison networks. For example, the goal

of this chapter is to develop a family SORTER of efficient sorting networks. We specify a given network within a family by the family name and the number of inputs (which equals the number of outputs). For example, the n-input, n-output sorting network in the family SORTER is named SORTER[n].

Exercises

27.1-1
Show the values that appear on all the wires of the network of Figure 27.2 when it is given the input sequence $\langle 9, 6, 5, 2 \rangle$.

27.1-2
Let n be an exact power of 2. Show how to construct an n-input, n-output comparison network of depth $\lg n$ in which the top output wire always carries the minimum input value and the bottom output wire always carries the maximum input value.

27.1-3
It is possible to take a sorting network and add a comparator to it, resulting in a comparison network that is not a sorting network. Show how to add a comparator to the network of Figure 27.2 in such a way that the resulting network does not sort every input permutation.

27.1-4
Prove that any sorting network on n inputs has depth at least $\lg n$.

27.1-5
Prove that the number of comparators in any sorting network is $\Omega(n \lg n)$.

27.1-6
Consider the comparison network shown in Figure 27.3. Prove that it is in fact a sorting network, and describe how its structure is related to that of insertion sort (Section 2.1).

27.1-7
We can represent an n-input comparison network with c comparators as a list of c pairs of integers in the range from 1 to n. If two pairs contain an integer in common, the order of the corresponding comparators in the network is determined by the order of the pairs in the list. Given this representation, describe an $O(n + c)$-time (serial) algorithm for determining the depth of a comparison network.

27.1-8 $\star$
Suppose that in addition to the standard kind of comparator, we introduce an "upside-down" comparator that produces its minimum output on the bottom wire

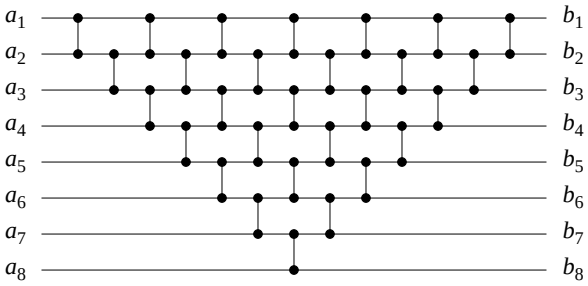

Figure 27.3 A sorting network based on insertion sort for use in Exercise 27.1-6.

and its maximum output on the top wire. Show how to convert any sorting network that uses a total of c standard and upside-down comparators to one that uses c standard ones. Prove that your conversion method is correct.

27.2 The zero-one principle

The ***zero-one principle*** says that if a sorting network works correctly when each input is drawn from the set $\{0, 1\}$, then it works correctly on arbitrary input numbers. (The numbers can be integers, reals, or, in general, any set of values from any linearly ordered set.) As we construct sorting networks and other comparison networks, the zero-one principle will allow us to focus on their operation for input sequences consisting solely of 0's and 1's. Once we have constructed a sorting network and proved that it can sort all zero-one sequences, we shall appeal to the zero-one principle to show that it properly sorts sequences of arbitrary values.

The proof of the zero-one principle relies on the notion of a monotonically increasing function (Section 3.2).

Lemma 27.1
If a comparison network transforms the input sequence $a = \langle a_1, a_2, \ldots, a_n \rangle$ into the output sequence $b = \langle b_1, b_2, \ldots, b_n \rangle$, then for any monotonically increasing function f, the network transforms the input sequence $f(a) = \langle f(a_1), f(a_2), \ldots, f(a_n) \rangle$ into the output sequence $f(b) = \langle f(b_1), f(b_2), \ldots, f(b_n) \rangle$.

Proof We shall first prove the claim that if f is a monotonically increasing function, then a single comparator with inputs $f(x)$ and $f(y)$ produces outputs $f(\min(x, y))$ and $f(\max(x, y))$. We then use induction to prove the lemma.

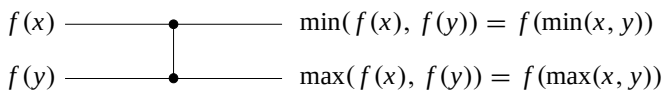

Figure 27.4 The operation of the comparator in the proof of Lemma 27.1. The function f is monotonically increasing.

To prove the claim, consider a comparator whose input values are x and y. The upper output of the comparator is $\min(x, y)$ and the lower output is $\max(x, y)$. Suppose we now apply $f(x)$ and $f(y)$ to the inputs of the comparator, as is shown in Figure 27.4. The operation of the comparator yields the value $\min(f(x), f(y))$ on the upper output and the value $\max(f(x), f(y))$ on the lower output. Since f is monotonically increasing, $x \leq y$ implies $f(x) \leq f(y)$. Consequently, we have the identities

$$\begin{aligned} \min(f(x), f(y)) &= f(\min(x, y)) , \\ \max(f(x), f(y)) &= f(\max(x, y)) . \end{aligned}$$

Thus, the comparator produces the values $f(\min(x, y))$ and $f(\max(x, y))$ when $f(x)$ and $f(y)$ are its inputs, which completes the proof of the claim.

We can use induction on the depth of each wire in a general comparison network to prove a stronger result than the statement of the lemma: if a wire assumes the value a_i when the input sequence a is applied to the network, then it assumes the value $f(a_i)$ when the input sequence $f(a)$ is applied. Because the output wires are included in this statement, proving it will prove the lemma.

For the basis, consider a wire at depth 0, that is, an input wire a_i. The result follows trivially: when $f(a)$ is applied to the network, the input wire carries $f(a_i)$. For the inductive step, consider a wire at depth d, where $d \geq 1$. The wire 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 the inductive hypothesis, therefore, if the input wires to the comparator carry values a_i and a_j when the input sequence a is applied, then they carry $f(a_i)$ and $f(a_j)$ when the input sequence $f(a)$ is applied. By our earlier claim, the output wires of this comparator then carry $f(\min(a_i, a_j))$ and $f(\max(a_i, a_j))$. Since they carry $\min(a_i, a_j)$ and $\max(a_i, a_j)$ when the input sequence is a, the lemma is proved. ■

As an example of the application of Lemma 27.1, Figure 27.5(b) shows the sorting network from Figure 27.2 (repeated in Figure 27.5(a)) with the monotonically increasing function $f(x) = \lceil x/2 \rceil$ applied to the inputs. The value on every wire is f applied to the value on the same wire in Figure 27.2.

When a comparison network is a sorting network, Lemma 27.1 allows us to prove the following remarkable result.

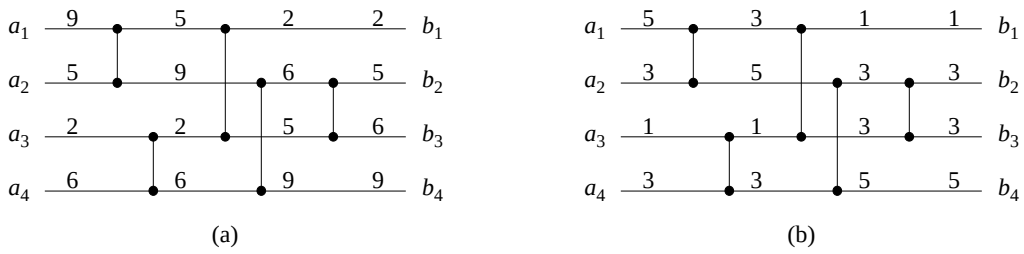

Figure 27.5 **(a)** The sorting network from Figure 27.2 with input sequence $\langle 9, 5, 2, 6\rangle$. **(b)** The same sorting network with the monotonically increasing function $f(x) = \lceil x/2 \rceil$ applied to the inputs. Each wire in this network has the value of f applied to the value on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

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 Suppose for the purpose of contradiction that the network sorts all zero-one sequences, but there exists a sequence of arbitrary numbers that the network does not correctly sort. That is, there exists an input sequence $\langle a_1, a_2, \ldots, a_n\rangle$ 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. We define a monotonically increasing function f as

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

Since the network places a_j before a_i in the output sequence when $\langle a_1, a_2, \ldots, a_n\rangle$ is input, it follows from Lemma 27.1 that it places $f(a_j)$ before $f(a_i)$ in the output sequence when $\langle f(a_1), f(a_2), \ldots, f(a_n)\rangle$ is input. But since $f(a_j) = 1$ and $f(a_i) = 0$, we obtain the contradiction that the network fails to sort the zero-one sequence $\langle f(a_1), f(a_2), \ldots, f(a_n)\rangle$ correctly. ■

Exercises

27.2-1
Prove that applying a monotonically increasing function to a sorted sequence produces a sorted sequence.

27.2-2
Prove that a comparison network with n inputs correctly sorts the input sequence $\langle n, n-1, \ldots, 1\rangle$ if and only if it correctly sorts the $n-1$ zero-one sequences $\langle 1, 0, 0, \ldots, 0, 0\rangle$, $\langle 1, 1, 0, \ldots, 0, 0\rangle$, ..., $\langle 1, 1, 1, \ldots, 1, 0\rangle$.

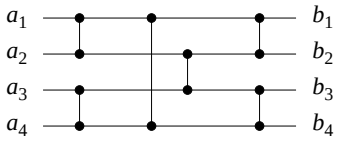

Figure 27.6 A sorting network for sorting 4 numbers.

27.2-3
Use the zero-one principle to prove that the comparison network shown in Figure 27.6 is a sorting network.

27.2-4
State and prove an analog of the zero-one principle for a decision-tree model. (*Hint:* Be sure to handle equality properly.)

27.2-5
Prove that an n-input sorting network must contain at least one comparator between the ith and $(i+1)$st lines for all $i = 1, 2, \ldots, n-1$.

27.3 A bitonic sorting network

The first step in our construction of an efficient sorting network is to construct a comparison network that can sort any ***bitonic sequence***: a sequence that monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing. For example, the sequences $\langle 1, 4, 6, 8, 3, 2\rangle$, $\langle 6, 9, 4, 2, 3, 5\rangle$, and $\langle 9, 8, 3, 2, 4, 6\rangle$ are all bitonic. As a boundary condition, we say that any sequence of just 1 or 2 numbers is bitonic. The zero-one sequences that are bitonic have a simple structure. They have the form $0^i 1^j 0^k$ or the form $1^i 0^j 1^k$, for some $i, j, k \geq 0$. Note that a sequence that is either monotonically increasing or monotonically decreasing is also bitonic.

The bitonic sorter that we shall construct is a comparison network that sorts bitonic sequences of 0's and 1's. Exercise 27.3-6 asks you to show that the bitonic sorter can sort bitonic sequences of arbitrary numbers.

The half-cleaner

A bitonic sorter is composed of several stages, each of which is called a ***half-cleaner***. Each half-cleaner is a comparison network of depth 1 in which input line i is compared with line $i + n/2$ for $i = 1, 2, \ldots, n/2$. (We assume that n is

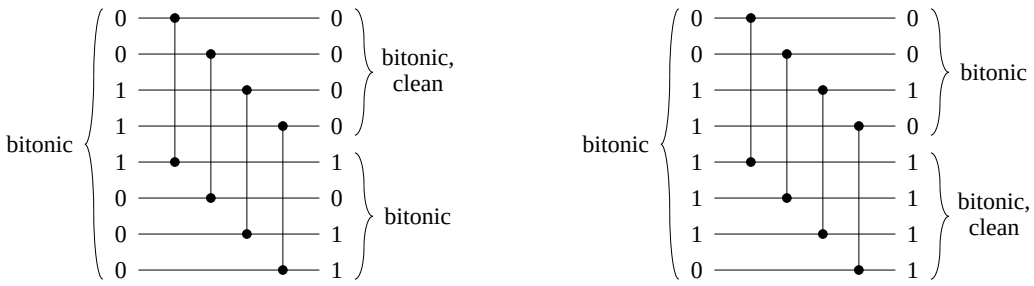

Figure 27.7 The comparison network HALF-CLEANER[8]. Two different sample zero-one input and output values are shown. The input is assumed to be bitonic. A half-cleaner ensures that every output element of the top half is at least as small as every output element of the bottom half. Moreover, both halves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

even.) Figure 27.7 shows HALF-CLEANER[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 HALF-CLEANER[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 $00\ldots011\ldots100\ldots0$. (The situation in which the input is of the form $11\ldots100\ldots011\ldots1$ is symmetric.) There are three 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. ■

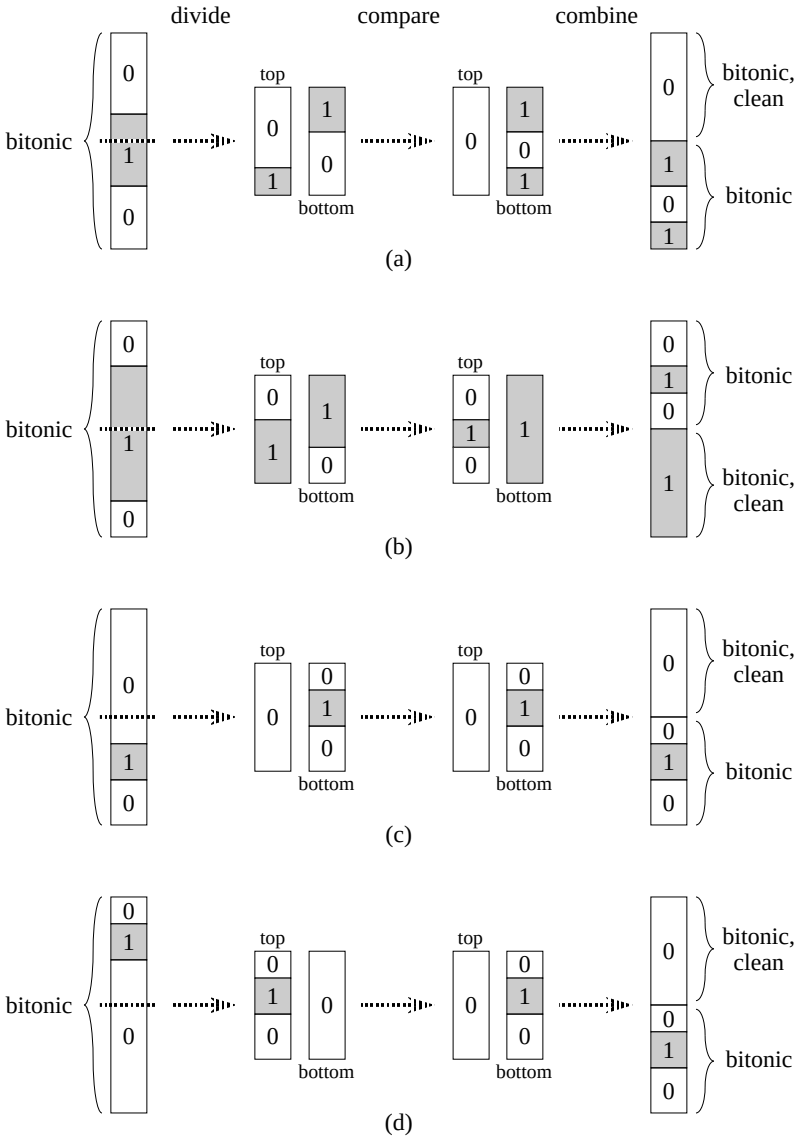

Figure 27.8 The possible comparisons in HALF-CLEANER[n]. The input sequence is assumed to be a bitonic sequence of 0's and 1's, and without loss of generality, we assume that it is of the form $00\ldots011\ldots100\ldots0$. Subsequences of 0's are white, and subsequences of 1's are gray. We can think 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.

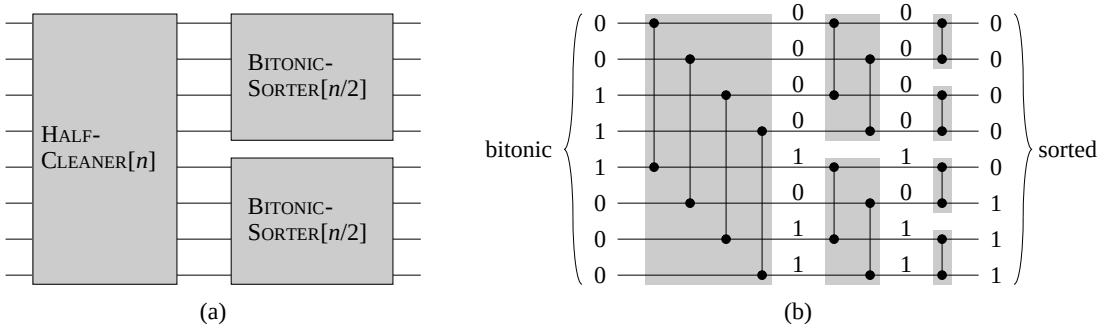

Figure 27.9 The comparison network BITONIC-SORTER[n], shown here for $n = 8$. **(a)** The recursive construction: HALF-CLEANER[n] followed by two copies of BITONIC-SORTER[$n/2$] that operate in parallel. **(b)** The network after unrolling the recursion. Each half-cleaner is shaded. Sample zero-one values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

The bitonic sorter

By recursively combining half-cleaners, as shown in Figure 27.9, we can build a ***bitonic sorter***, which is a network that sorts bitonic sequences. The first stage of BITONIC-SORTER[n] consists of HALF-CLEANER[n], which, by Lemma 27.3, produces two bitonic sequences of half the size such that every element in the top half is at least as small as every element in the bottom half. Thus, we can complete the sort by using two copies of BITONIC-SORTER[$n/2$] to sort the two halves recursively. In Figure 27.9(a), the recursion has been shown explicitly, and in Figure 27.9(b), the recursion has been unrolled to show the progressively smaller half-cleaners that make up the remainder of the bitonic sorter. The depth $D(n)$ of BITONIC-SORTER[n] is given by the recurrence

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

whose solution is $D(n) = \lg n$.

Thus, a zero-one bitonic sequence can be sorted by BITONIC-SORTER, which has a depth of $\lg n$. It follows by the analog of the zero-one principle given as Exercise 27.3-6 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
Show that BITONIC-SORTER[n], where n is an exact power of 2, contains $\Theta(n \lg n)$ comparators.

27.3-3
Describe how an $O(\lg n)$-depth bitonic sorter can be constructed when the number n of inputs is not an exact power of 2.

27.3-4
If the input to a half-cleaner is a bitonic sequence of arbitrary numbers, prove that the output satisfies the following properties: both the top half and the bottom half are bitonic, and every element in the top half is at least as small as every element in the bottom half.

27.3-5
Consider two sequences of 0's and 1's. Prove that if every element in one sequence is at least as small as every element in the other sequence, then one of the two sequences is clean.

27.3-6
Prove the following analog of the zero-one principle for bitonic sorting networks: a comparison network that can sort any bitonic sequence of 0's and 1's can sort any bitonic sequence of arbitrary numbers.

27.4 A merging network

Our sorting network will be constructed from ***merging networks***, which are networks that can merge two sorted input sequences into one sorted output sequence. We modify BITONIC-SORTER[n] to create the merging network MERGER[n]. As with the bitonic sorter, we shall prove the correctness of the merging network only for inputs that are zero-one sequences. Exercise 27.4-1 asks you to show how the proof can be extended to arbitrary input values.

The merging network is based on the following intuition. 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 example, given the sorted zero-one sequences $X = 00000111$ and $Y = 00001111$, we reverse Y to get $Y^{\mathrm{R}} = 11110000$. Concatenating X and Y^{R} yields 0000011111110000, which is bitonic. Thus, to merge the two input sequences X and Y, it suffices to perform a bitonic sort on X concatenated with Y^{R}.

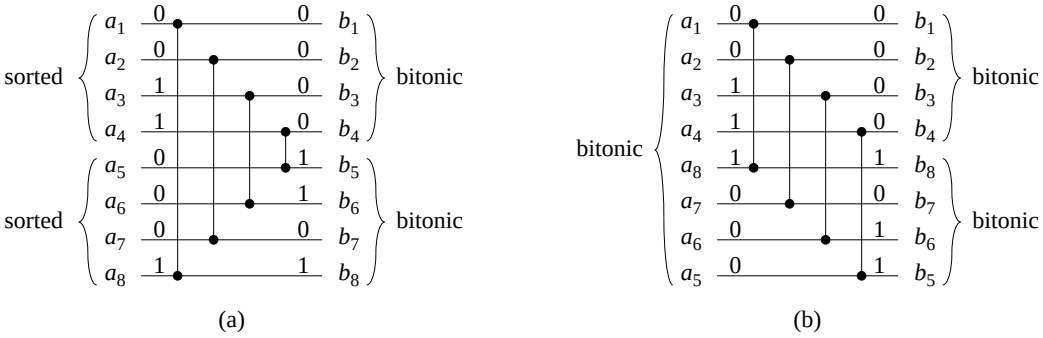

Figure 27.10 Comparing the first stage of MERGER[n] with HALF-CLEANER[n], for $n = 8$. **(a)** The first stage of MERGER[n] transforms the two monotonic input sequences $\langle a_1, a_2, \ldots, a_{n/2}\rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n\rangle$ into two bitonic sequences $\langle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

We can construct MERGER[n] by modifying the first half-cleaner of BITONIC-SORTER[n]. The key is to perform the reversal of the second half of the inputs implicitly. Given two sorted sequences $\langle a_1, a_2, \ldots, a_{n/2}\rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n\rangle$ to be merged, we want the effect of bitonically sorting the sequence $\langle a_1, a_2, \ldots, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+1}\rangle$. Since the first half-cleaner of BITONIC-SORTER[n] compares inputs i and $n/2 + i$, for $i = 1, 2, \ldots, n/2$, we make the first stage of the merging network compare inputs i and $n - i + 1$. Figure 27.10 shows the correspondence. The only subtlety is that 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. Since the reversal of a bitonic sequence is bitonic, however, the top and bottom outputs of the first stage of the merging network satisfy the properties in Lemma 27.3, and thus the top and bottom can be bitonically sorted in parallel to produce the sorted output of the merging network.

The resulting merging network is shown in Figure 27.11. Only the first stage of MERGER[n] is different from BITONIC-SORTER[n]. Consequently, the depth of MERGER[n] is $\lg n$, the same as that of BITONIC-SORTER[n].

Exercises

27.4-1

Prove an analog of the zero-one principle for merging networks. Specifically, show that a comparison network that can merge any two monotonically increasing se-

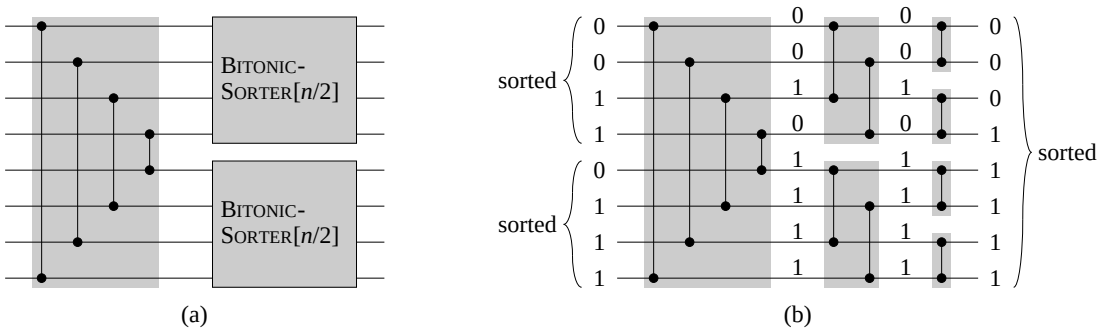

Figure 27.11 A network that merges two sorted input sequences into one sorted output sequence. The network MERGER[n] can be viewed as BITONIC-SORTER[n] with the first half-cleaner altered to compare inputs i and $n-i+1$ for $i = 1, 2, \ldots, n/2$. Here, $n = 8$. **(a)** The network decomposed into the 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.

quences of 0's and 1's can merge any two monotonically increasing sequences of arbitrary numbers.

27.4-2

How many different zero-one input sequences must be applied to the input of a comparison network to verify that it is a merging network?

27.4-3

Show that any network that can merge 1 item with $n-1$ sorted items to produce a sorted sequence of length n must have depth at least $\lg n$.

27.4-4 $\star$

Consider a merging network with inputs $a_1, a_2, \ldots, a_n$, for n an exact power of 2, in which the two monotonic sequences to be merged are $\langle a_1, a_3, \ldots, a_{n-1}\rangle$ and $\langle a_2, a_4, \ldots, a_n\rangle$. Prove that the number of comparators in this kind of merging network is $\Omega(n \lg n)$. Why is this an interesting lower bound? (*Hint:* Partition the comparators into three sets.)

27.4-5 $\star$

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

27.5 A sorting network

We now have all the necessary tools to construct a network that can sort any input sequence. The sorting network SORTER[n] uses the merging network to implement a parallel version of merge sort from Section 2.3.1. The construction and operation of the sorting network are illustrated in Figure 27.12.

Figure 27.12(a) shows the recursive construction of SORTER[n]. The n input elements are sorted by using two copies of SORTER[$n/2$] recursively to sort (in parallel) two subsequences of length $n/2$ each. The two resulting sequences are then merged by MERGER[n]. The boundary case for the recursion is when $n = 1$, in which case we can use a wire to sort the 1-element sequence, since a 1-element sequence is already sorted. Figure 27.12(b) shows the result of unrolling the recursion, and Figure 27.12(c) shows the actual network obtained by replacing the MERGER boxes in Figure 27.12(b) with the actual merging networks.

Data pass through $\lg n$ stages in the network SORTER[n]. Each of the individual inputs to the network is already a sorted 1-element sequence. The first stage of SORTER[n] consists of $n/2$ copies of MERGER[2] that work in parallel to merge pairs of 1-element sequences to produce sorted sequences of length 2. The second stage consists of $n/4$ copies of MERGER[4] that merge pairs of these 2-element sorted sequences to produce sorted sequences of length 4. In general, for $k = 1, 2, \ldots, \lg n$, stage k consists of $n/2^k$ copies of MERGER[2^k] that merge pairs of the 2^{k-1}-element sorted sequences to produce sorted sequences of length 2^k. At the final stage, one sorted sequence consisting of all the input values is produced. This sorting network can be shown by induction to sort zero-one sequences, and consequently, by the zero-one principle (Theorem 27.2), it can sort arbitrary values.

We can analyze the depth of the sorting network recursively. The depth $D(n)$ of SORTER[n] is the depth $D(n/2)$ of SORTER[$n/2$] (there are two copies of SORTER[$n/2$], but they operate in parallel) plus the depth $\lg n$ of MERGER[n]. Consequently, the depth of SORTER[n] is given by the recurrence

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

whose solution is $D(n) = \Theta(\lg^2 n)$. (Use the version of the master method given in Exercise 4.4-2.) Thus, we can sort n numbers in parallel in $O(\lg^2 n)$ time.

Exercises

27.5-1

How many comparators are there in SORTER[n]?

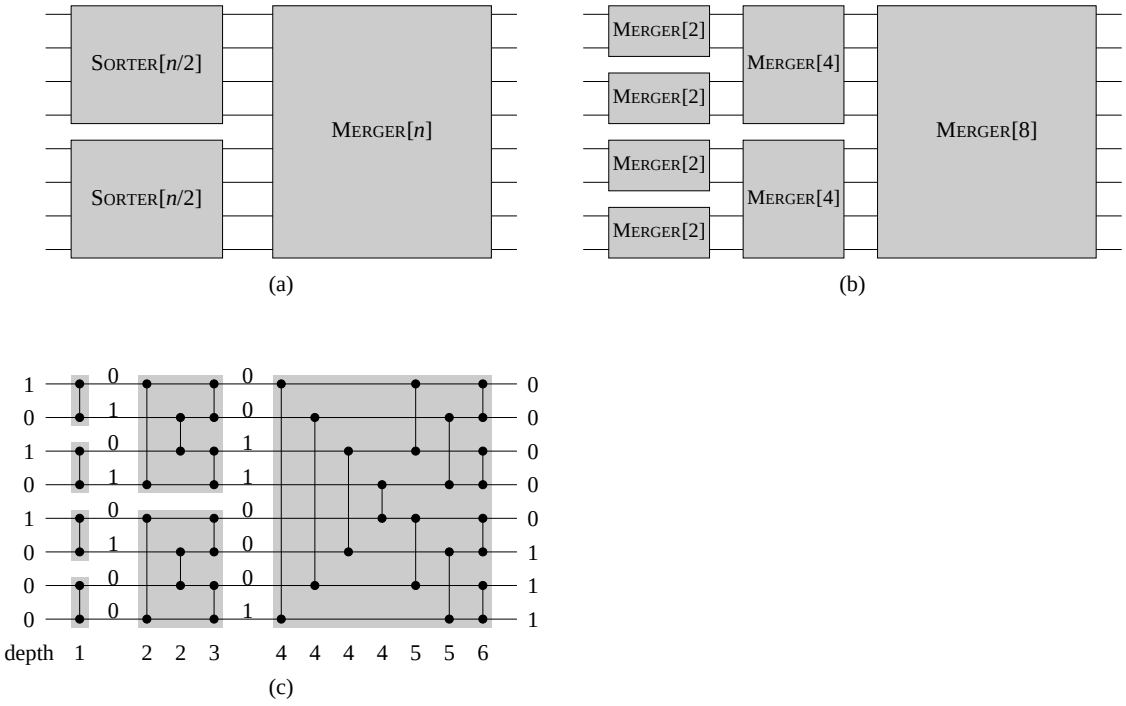

Figure 27.12 The sorting network SORTER[n] constructed by recursively combining merging networks. **(a)** The recursive construction. **(b)** Unrolling the recursion. **(c)** Replacing the MERGER boxes with the actual merging networks. The depth of each comparator is indicated, and sample zero-one values are shown on the wires.

27.5-2

Show that the depth of SORTER[n] is exactly $(\lg n)(\lg n + 1)/2$.

27.5-3

Suppose that we have $2n$ elements $\langle a_1, a_2, \ldots, a_{2n}\rangle$ and wish to partition them into the n smallest and the n largest. Prove that we can do this in constant additional depth after separately sorting $\langle a_1, a_2, \ldots, a_n\rangle$ and $\langle a_{n+1}, a_{n+2}, \ldots, a_{2n}\rangle$.

27.5-4 ⋆

Let $S(k)$ be the depth of a sorting network with k inputs, and let $M(k)$ be the depth of a merging network with $2k$ inputs. Suppose that we have a sequence of n numbers to be sorted and we know that every number is within k positions of its correct position in the sorted order. Show that we can sort the n numbers in depth $S(k) + 2M(k)$.

27.5-5 $\star$

We can sort the entries of an $m \times m$ matrix by repeating the following procedure k times:

1. Sort each odd-numbered row into monotonically increasing order.
2. Sort each even-numbered row into monotonically decreasing order.
3. Sort each column into monotonically increasing order.

How many iterations k are required for this procedure to sort, and in what order should we read the matrix entries after the k iterations to obtain the sorted output?

Problems

27-1 Transposition sorting networks

A comparison network is a ***transposition network*** if each comparator connects adjacent lines, as in the network in Figure 27.3.

a. Show that any transposition sorting network with n inputs has $\Omega(n^2)$ comparators.

b. Prove that a transposition network with n inputs is a sorting network if and only if it sorts the sequence $\langle n, n-1, \ldots, 1\rangle$. (*Hint:* Use an induction argument analogous to the one in the proof of Lemma 27.1.)

An ***odd-even sorting network*** on n inputs $\langle a_1, a_2, \ldots, a_n\rangle$ is a transposition sorting network with n levels of comparators connected in the "brick-like" pattern illustrated in Figure 27.13. As can be seen in the figure, for $i = 1, 2, \ldots, n$ and $d = 1, 2, \ldots, n$, line i is connected by a depth-d comparator to line $j = i + (-1)^{i+d}$ if $1 \le j \le n$.

c. Prove that odd-even sorting networks actually sort.

27-2 Batcher's odd-even merging network

In Section 27.4, we saw how to construct a merging network based on bitonic sorting. In this problem, we shall construct an ***odd-even merging network***. We assume that n is an exact power of 2, and we wish to merge the sorted sequence of elements on lines $\langle a_1, a_2, \ldots, a_n\rangle$ with those on lines $\langle a_{n+1}, a_{n+2}, \ldots, a_{2n}\rangle$. If $n = 1$, we put a comparator between lines a_1 and a_2. Otherwise, we recursively construct two odd-even merging networks that operate in parallel. The first merges the sequence on lines $\langle a_1, a_3, \ldots, a_{n-1}\rangle$ with the sequence on lines $\langle a_{n+1}, a_{n+3}, \ldots, a_{2n-1}\rangle$ (the odd elements). The second merges $\langle a_2, a_4, \ldots, a_n\rangle$ with $\langle a_{n+2}, a_{n+4}, \ldots, a_{2n}\rangle$ (the

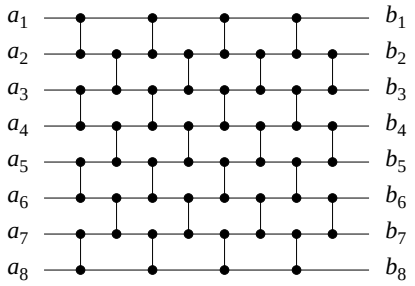

Figure 27.13 An odd-even sorting network on 8 inputs.

even elements). To combine the two sorted subsequences, we put a comparator between a_{2i} and a_{2i+1} for $i = 1, 2, \ldots, n-1$.

a. Draw a $2n$-input merging network for $n = 4$.

b. Professor Corrigan suggests that to combine the two sorted subsequences produced by the recursive merging, instead of putting a comparator between a_{2i} and a_{2i+1} for $i = 1, 2, \ldots, n-1$, one should put a comparator between a_{2i-1} and a_{2i} for $i = 1, 2, \ldots, n$. Draw such a $2n$-input network for $n = 4$, and give a counterexample to show that the professor is mistaken in thinking that the network produced is a merging network. Show that the $2n$-input merging network from part (a) works properly on your example.

c. Use the zero-one principle to prove that any $2n$-input odd-even merging network is indeed a merging network.

d. What is the depth of a $2n$-input odd-even merging network? What is its size?

27-3 Permutation networks

A ***permutation network*** on n inputs and n outputs has switches that allow it to connect its inputs to its outputs according to any of the $n!$ possible permutations. Figure 27.14(a) shows the 2-input, 2-output permutation network P_2, which consists of a single switch that can be set either to feed its inputs straight through to its outputs or to cross them.

a. Argue that if we replace each comparator in a sorting network with the switch of Figure 27.14(a), the resulting network is a permutation network. That is, for

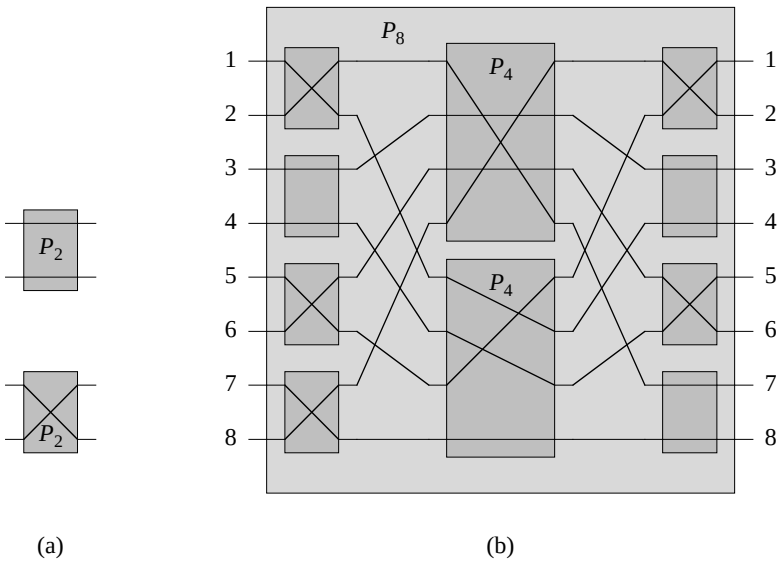

Figure 27.14 Permutation networks. **(a)** The permutation network P_2, which consists of a single switch that can be set in either of the two ways shown. **(b)** The recursive construction of P_8 from 8 switches and two P_4's. The switches and P_4's are set to realize the permutation $\pi = \langle 4, 7, 3, 5, 1, 6, 8, 2\rangle$.

any permutation π, there is a way to set the switches in the network so that input i is connected to output $\pi(i)$.

Figure 27.14(b) shows the recursive construction of an 8-input, 8-output permutation network P_8 that uses two copies of P_4 and 8 switches. The switches have been set to realize the permutation $\pi = \langle \pi(1), \pi(2), \ldots, \pi(8)\rangle = \langle 4, 7, 3, 5, 1, 6, 8, 2\rangle$, which requires (recursively) that the top P_4 realize $\langle 4, 2, 3, 1\rangle$ and the bottom P_4 realize $\langle 2, 3, 1, 4\rangle$.

b. Show how to realize the permutation $\langle 5, 3, 4, 6, 1, 8, 2, 7\rangle$ on P_8 by drawing the switch settings and the permutations performed by the two P_4's.

Let n be an exact power of 2. Define P_n recursively in terms of two $P_{n/2}$'s in a manner similar to the way we defined P_8.

c. Describe an $O(n)$-time (ordinary random-access machine) algorithm that sets the n switches connected to the inputs and outputs of P_n and specifies the permutations that must be realized by each $P_{n/2}$ in order to accomplish any given n-element permutation. Prove that your algorithm is correct.

d. What are the depth and size of P_n? How long does it take on an ordinary random-access machine to compute all switch settings, including those within the $P_{n/2}$'s?

e. Argue that for $n > 2$, any permutation network—not just P_n—must realize some permutation by two distinct combinations of switch settings.

Chapter notes

Knuth [185] contains a discussion of sorting networks and charts their history. They apparently were first explored in 1954 by P. N. Armstrong, R. J. Nelson, and D. J. O'Connor. In the early 1960's, K. E. Batcher discovered the first network capable of merging two sequences of n numbers in $O(\lg n)$ time. He used odd-even merging (see Problem 27-2), and he also showed how this technique could be used to sort n numbers in $O(\lg^2 n)$ time. Shortly afterward, he discovered an $O(\lg n)$-depth bitonic sorter similar to the one presented in Section 27.3. Knuth attributes the zero-one principle to W. G. Bouricius (1954), who proved it in the context of decision trees.

For a long time, the question remained open as to whether a sorting network with depth $O(\lg n)$ exists. In 1983, the answer was shown to be a somewhat unsatisfying yes. The AKS sorting network (named after its developers, Ajtai, Komlós, and Szemerédi [11]) can sort n numbers in depth $O(\lg n)$ using $O(n \lg n)$ comparators. Unfortunately, the constants hidden by the O-notation are quite large (many thousands), and thus it cannot be considered practical.