for $n \geq 5$, by constructing an adversary that deals with components $\bullet$, $\longrightarrow$, $\rightarrow$, $\rightarrow\!\!\!\!\rightarrow$. (c) Therefore we have $U_t(n) \geq n + t + \min(\lfloor (n-t)/2 \rfloor, t) - 3$ for $1 \leq t \leq n/2$. [The inequalities in (a) and (b) apply also when V or W replaces U, thereby establishing the optimality of several entries in Table 1.]

▸ **27.** [*M34*] A *randomized adversary* is an adversary algorithm that is allowed to flip coins as it makes decisions.

a) Let A be a randomized adversary and let $\Pr(l)$ be the probability that A reaches leaf l of a given comparison tree. Show that if $\Pr(l) \leq p$ for all l, the height of the comparison tree is $\geq \lg(1/p)$.

b) Consider the following adversary for the problem of selecting the tth largest of n elements, given integer parameters q and r to be selected later:

A1. Choose a random set T of t elements; all $\binom{n}{t}$ possibilities are equally likely. (We will ensure that the $t-1$ largest elements belong to T.) Let $S = \{1, \ldots, n\} \setminus T$ be the other elements, and set $S_0 \leftarrow S$, $T_0 \leftarrow T$; S_0 and T_0 will represent elements that might become the tth largest.

A2. While $|T_0| > r$, decide all comparisons $x{:}y$ as follows: If $x \in S$ and $y \in T$, say that $x < y$. If $x \in S$ and $y \in S$, flip a coin to decide, and remove the smaller element from S_0 if it was in S_0. If $x \in T$ and $y \in T$, flip a coin to decide, and remove the larger element from T_0 if it was in T_0.

A3. As soon as $|T_0| = r$, partition the elements into three classes P, Q, R as follows: If $|S_0| < q$, let $P = S$, $Q = T_0$, $R = T \setminus T_0$. Otherwise, for each $y \in T_0$, let $C(y)$ be the elements of S already compared with y, and choose y_0 so that $|C(y_0)|$ is minimum. Let $P = (S \setminus S_0) \cup C(y_0)$, $Q = (S_0 \setminus C(y_0)) \cup \{y_0\}$, $R = T \setminus \{y_0\}$. Decide all future comparisons $x{:}y$ by saying that elements of P are less than elements of Q, and elements of Q are less than elements of R; flip a coin when x and y are in the same class.

Prove that if $1 \leq r \leq t$ and if $|C(y_0)| \leq q - r$ at the beginning of step A3, each leaf is reached with probability $\leq (n+1-t)/\bigl(2^{n-q}\binom{n}{t}\bigr)$. *Hint:* Show that at least $n-q$ coin flips are made.

c) Continuing (b), show that we have

$$V_t(n) \geq \min\bigl(n - 1 + (r-1)(q+1-r),\ n - q + \lg\bigl(\tbinom{n}{t}/(n+1-t)\bigr)\bigr),$$

for all integers q and r.

d) Establish (14) by choosing q and r.

*5.3.4. Networks for Sorting

In this section we shall study a constrained type of sorting that is particularly interesting because of its applications and its rich underlying theory. The new constraint is to insist on an *oblivious* sequence of comparisons, in the sense that whenever we compare K_i versus K_j the subsequent comparisons for the case $K_i < K_j$ are exactly the same as for the case $K_i > K_j$, but with i and j interchanged.

Figure 43(a) shows a comparison tree in which this homogeneity condition is satisfied. Notice that every level has the same number of comparisons, so there are 2^m outcomes after m comparisons have been made. But $n!$ is not a power of 2; some of the comparisons must therefore be redundant, in the sense that

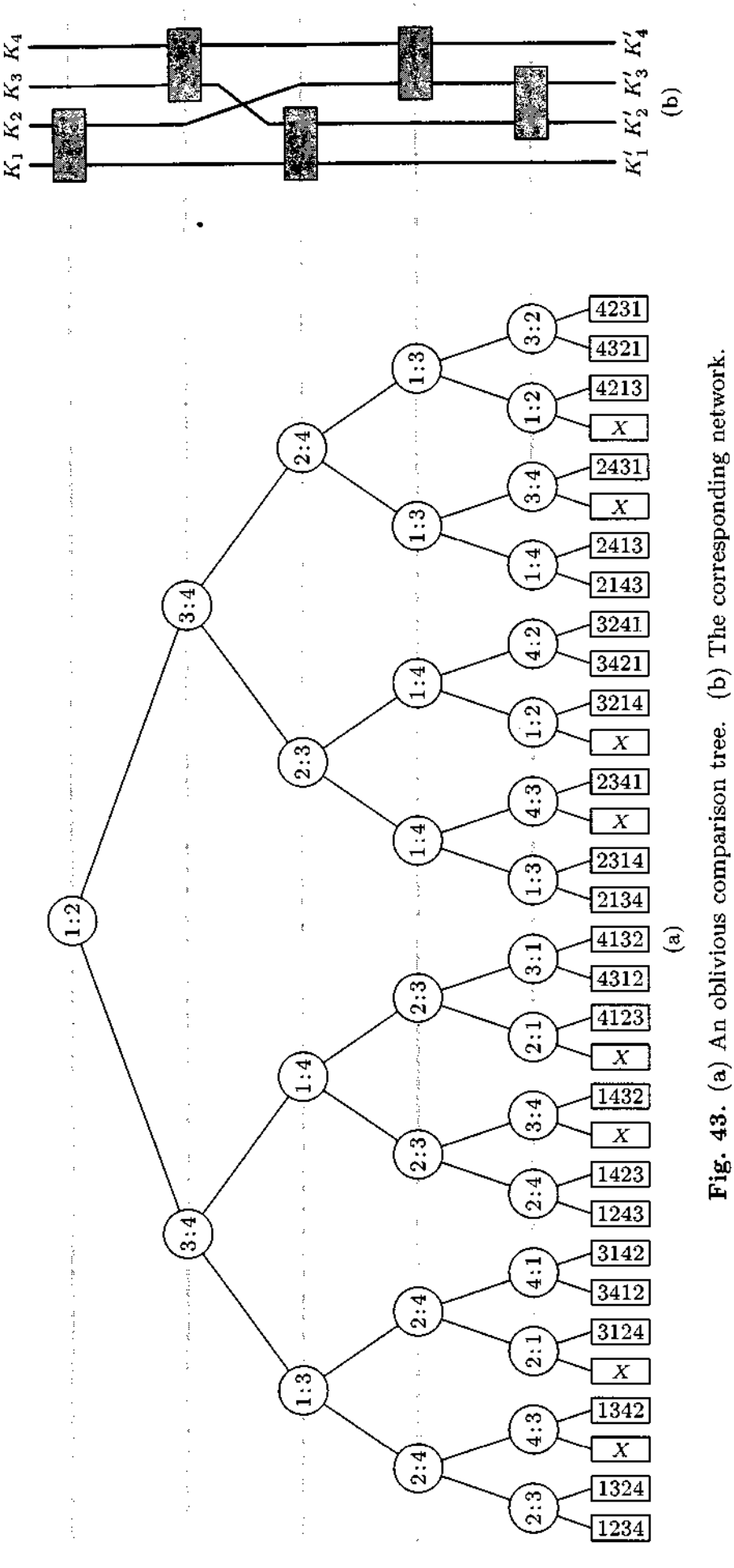


Fig. 43. (a) An oblivious comparison tree. (b) The corresponding network.

one of their subtrees can never arise in practice. In other words, some branches of the tree must make more comparisons than necessary, in order to ensure that all of the corresponding branches of the tree will sort properly.

Since each path from top to bottom of such a tree determines the entire tree, such a sorting scheme is most easily represented as a *network*; see Fig. 43(b). The boxes in such a network represent "comparator modules" that have two inputs (represented as lines coming into the module from above) and two outputs (represented as lines leading downward); the left-hand output is the smaller of the two inputs, and the right-hand output is the larger. At the bottom of the network, K_1' is the smallest of $\{K_1, K_2, K_3, K_4\}$, K_2' the second smallest, etc. It is not difficult to prove that any sorting network corresponds to an oblivious comparison tree in the sense above, and any oblivious tree corresponds to a network of comparator modules.

Incidentally, we may note that comparator modules are fairly easy to manufacture, from an engineering point of view. For example, assume that the lines contain binary numbers, where one bit enters each module per unit time, most significant bit first. Each comparator module has three states, and behaves as follows:

Time t		Time $(t+1)$	
State	Inputs	State	Outputs
0	0 0	0	0 0
0	0 1	1	0 1
0	1 0	2	0 1
0	1 1	0	1 1
1	x y	1	x y
2	x y	2	y x

Initially all modules are in state 0 and are outputting 0 0. A module enters either state 1 or state 2 as soon as its inputs differ. Numbers that begin to be transmitted at the top of Fig. 43(b) at time t will begin to be output at the bottom, in sorted order, at time $t + 3$, if a suitable delay element is attached to the K_1' and K_4' lines.

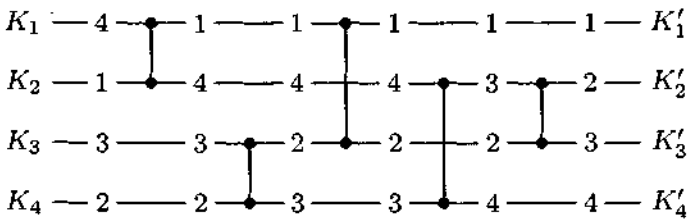


Fig. 44. Another way to represent the network of Fig. 43, as it sorts the sequence of four numbers $\langle 4, 1, 3, 2 \rangle$.

In order to develop the theory of sorting networks it is convenient to represent them in a slightly different way, illustrated in Fig. 44. Here numbers enter at the *left*, and comparator modules are represented by vertical connections between two lines; each comparator causes an interchange of its inputs, if necessary, so that the larger number sinks to the *lower* line after passing the comparator. At the right of the diagram all the numbers are in order from top to bottom.

Our previous studies of optimal sorting have concentrated on minimizing the number of comparisons, with little or no regard for any underlying data movement or for the complexity of the decision structure that may be necessary. In this respect sorting networks have obvious advantages, since the data can be maintained in n locations and the decision structure is "straight line" — there is no need to remember the results of previous comparisons, since the plan is immutably fixed in advance. Another important advantage of sorting networks is that we can usually overlap several of the operations, performing them simultaneously (on a suitable machine). For example, the five steps in Figs. 43 and 44 can be collapsed into three when simultaneous nonoverlapping comparisons are allowed, since the first two and the second two can be combined. We shall exploit this property of sorting networks later in this section. Thus sorting networks can be very useful, although it is not at all obvious that efficient n-element sorting networks can be constructed for large n; we may find that many additional comparisons are needed in order to keep the decision structure oblivious.

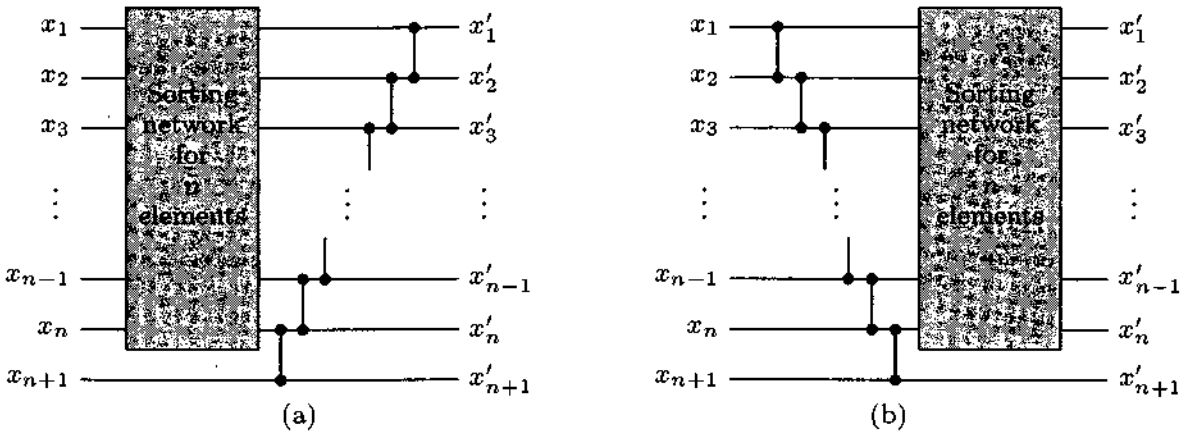


Fig. 45. Making $(n+1)$-sorters from n-sorters: (a) insertion, (b) selection.

There are two simple ways to construct a sorting network for $n+1$ elements when an n-element network is given, using either the principle of *insertion* or the principle of *selection*. Figure 45(a) shows how the $(n+1)$st element can be inserted into its proper place after the first n elements have been sorted; and part (b) of the figure shows how the largest element can be selected before we proceed to sort the remaining ones. Repeated application of Fig. 45(a) gives the network analog of straight insertion sorting (Algorithm 5.2.1S), and repeated application of Fig. 45(b) yields the network analog of the bubble sort (Algorithm 5.2.2B). Figure 46 shows the corresponding six-element networks.

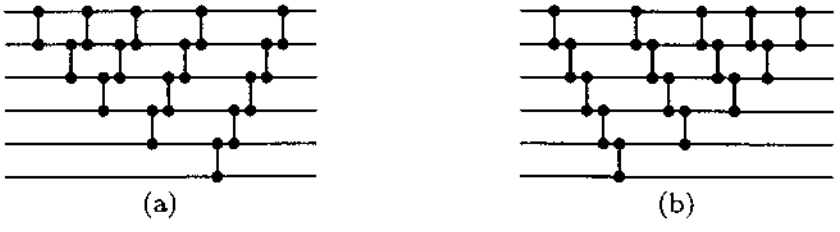


Fig. 46. Network analogs of elementary internal sorting schemes, obtained by applying the constructions of Fig. 45 repeatedly: (a) straight insertion, (b) bubble sort.

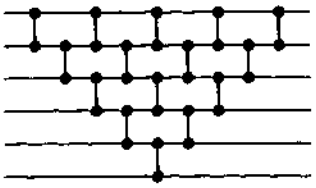
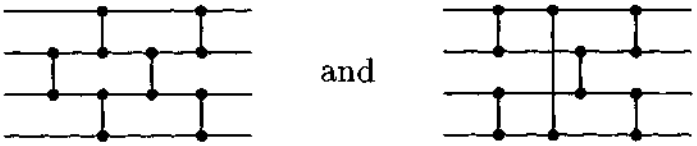


Fig. 47. With parallelism, straight insertion = bubble sort!

Notice that when we collapse either network together to allow simultaneous operations, both methods actually reduce to the same "triangular" $(2n-3)$-stage procedure (Fig. 47).

It is easy to prove that the network of Figs. 43 and 44 will sort any set of four numbers into order, since the first four comparators route the smallest and the largest elements to the correct places, and the last comparator puts the remaining two elements in order. But it is not always so easy to tell whether or not a given network will sort all possible input sequences; for example, both



are valid 4-element sorting networks, but the proofs of their validity are not trivial. It would be sufficient to test each n-element network on all $n!$ permutations of n distinct numbers, but in fact we can get by with far fewer tests:

Theorem Z (Zero-one principle). *If a network with n input lines sorts all 2^n sequences of 0s and 1s into nondecreasing order, it will sort any arbitrary sequence of n numbers into nondecreasing order.*

Proof. (This is a special case of Bouricius's theorem, exercise 5.3.1–12.) If $f(x)$ is any monotonic function, with $f(x) \le f(y)$ whenever $x \le y$, and if a given network transforms $\langle x_1, \ldots, x_n \rangle$ into $\langle y_1, \ldots, y_n \rangle$, then it is easy to see that the network will transform $\langle f(x_1), \ldots, f(x_n) \rangle$ into $\langle f(y_1), \ldots, f(y_n) \rangle$. If $y_i > y_{i+1}$ for some i, consider the monotonic function f that takes all numbers $< y_i$ into 0 and all numbers $\ge y_i$ into 1; this defines a sequence $\langle f(x_1), \ldots, f(x_n) \rangle$ of 0s and 1s that is not sorted by the network. Hence if all 0–1 sequences are sorted, we have $y_i \le y_{i+1}$ for $1 \le i < n$. ∎

The zero-one principle is quite helpful in the construction of sorting networks. As a nontrivial example, we can derive a generalized version of Batcher's "merge exchange" sort (Algorithm 5.2.2M). The idea is to sort $m+n$ elements by sorting the first m and the last n independently, then applying an (m,n)-*merging network* to the result. An (m,n)-merging network can be constructed inductively as follows:

a) If $m = 0$ or $n = 0$, the network is empty. If $m = n = 1$, the network is a single comparator module.

b) If $mn > 1$, let the sequences to be merged be $\langle x_1, \ldots, x_m \rangle$ and $\langle y_1, \ldots, y_n \rangle$. Merge the "odd sequences" $\langle x_1, x_3, \ldots, x_{2\lceil m/2 \rceil - 1} \rangle$ and $\langle y_1, y_3, \ldots, y_{2\lceil n/2 \rceil - 1} \rangle$,

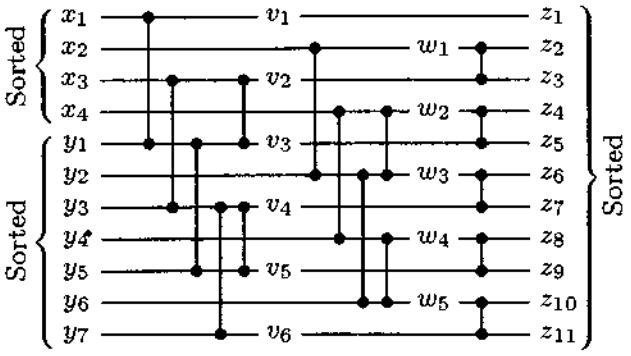


Fig. 48. The odd-even merge, when $m = 4$ and $n = 7$.

obtaining the sorted result $\langle v_1, v_2, \ldots, v_{\lceil m/2\rceil+\lceil n/2\rceil}\rangle$; and merge the "even sequences" $\langle x_2, x_4, \ldots, x_{2\lfloor m/2\rfloor}\rangle$ and $\langle y_2, y_4, \ldots, y_{2\lfloor n/2\rfloor}\rangle$, obtaining the sorted result $\langle w_1, w_2, \ldots, w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor}\rangle$. Finally, apply the comparison-interchange operations

$$w_1 : v_2, \quad w_2 : v_3, \quad w_3 : v_4, \quad \ldots, \quad w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor} : v^* \tag{1}$$

to the sequence

$$\langle v_1, w_1, v_2, w_2, v_3, w_3, \ldots, v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor}, w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor}, v^*, v^{**}\rangle; \tag{2}$$

the result will be sorted. (!) Here $v^* = v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor+1}$ does not exist if both m and n are even, and $v^{**} = v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor+2}$ does not exist unless both m and n are odd; the total number of comparator modules indicated in (1) is $\lfloor (m+n-1)/2\rfloor$. Batcher's (m, n)-merging network is called the *odd-even merge*. A $(4, 7)$-merge constructed according to these principles is illustrated in Fig. 48.

To prove that this rather strange merging procedure actually works, when $mn > 1$, we use the zero-one principle, testing it on all sequences of 0s and 1s. After the initial m-sort and n-sort, the sequence $\langle x_1, \ldots, x_m\rangle$ will consist of k 0s followed by $m - k$ 1s, and the sequence $\langle y_1, \ldots, y_m\rangle$ will be l 0s followed by $n - l$ 1s, for some k and l. Hence the sequence $\langle v_1, v_2, \ldots\rangle$ will consist of exactly $\lceil k/2\rceil + \lceil l/2\rceil$ 0s, followed by 1s; and $\langle w_1, w_2, \ldots\rangle$ will consist of $\lfloor k/2\rfloor + \lfloor l/2\rfloor$ 0s, followed by 1s. Now here's the point:

$$\big(\lceil k/2\rceil + \lceil l/2\rceil\big) - \big(\lfloor k/2\rfloor + \lfloor l/2\rfloor\big) = 0,\ 1,\ \text{or } 2. \tag{3}$$

If this difference is 0 or 1, the sequence (2) is already in order, and if the difference is 2 one of the comparison-interchanges in (1) will fix everything up. This completes the proof. (Note that the zero-one principle reduces the merging problem from a consideration of $\binom{m+n}{m}$ cases to only $(m+1)(n+1)$, represented by the two parameters k and l.)

Let $C(m, n)$ be the number of comparator modules used in the odd-even merge for m and n, not counting the initial m-sort and n-sort; we have

$$C(m,n) = \begin{cases} mn, & \text{if } mn \le 1; \\ C\big(\lceil m/2\rceil, \lceil n/2\rceil\big) + C\big(\lfloor m/2\rfloor, \lfloor n/2\rfloor\big) + \lfloor (m+n-1)/2\rfloor, & \text{if } mn > 1. \end{cases} \tag{4}$$

This is not an especially simple function of m and n, in general, but by noting that $C(1,n) = n$ and that

$$C(m+1,n+1) - C(m,n) \\ = 1 + C(\lfloor m/2\rfloor + 1, \lfloor n/2\rfloor + 1) - C(\lfloor m/2\rfloor, \lfloor n/2\rfloor), \quad \text{if } mn \geq 1,$$

we can derive the relation

$$C(m+1,n+1) - C(m,n) = \lfloor \lg m\rfloor + 2 + \lfloor n/2^{\lfloor \lg m\rfloor+1}\rfloor, \quad \text{if } n \geq m \geq 1. \tag{5}$$

Consequently

$$C(m,m+r) = B(m) + m + R_m(r), \qquad \text{for } m \geq 0 \text{ and } r \geq 0, \tag{6}$$

where $B(m)$ is the "binary insertion" function $\sum_{k=1}^{m}\lceil \lg k\rceil$ of Eq. 5.3.1–(3), and where $R_m(r)$ denotes the sum of the first m terms of the series

$$\left\lfloor\frac{r+0}{1}\right\rfloor + \left\lfloor\frac{r+1}{2}\right\rfloor + \left\lfloor\frac{r+2}{4}\right\rfloor + \left\lfloor\frac{r+3}{4}\right\rfloor + \left\lfloor\frac{r+4}{8}\right\rfloor + \cdots + \left\lfloor\frac{r+j}{2^{\lfloor \lg j\rfloor+1}}\right\rfloor + \cdots. \tag{7}$$

In particular, when $r = 0$ we have the important special case

$$C(m,m) = B(m) + m. \tag{8}$$

Furthermore if $t = \lceil \lg m\rceil$,

$$R_m(r+2^t) = R_m(r) + 1\cdot 2^{t-1} + 2\cdot 2^{t-2} + \cdots + 2^{t-1}\cdot 2^0 + m \\ = R_m(r) + m + t\cdot 2^{t-1}.$$

Hence $C(m,n+2^t) - C(m,n)$ has a simple form, and

$$C(m,n) = \left(\frac{t}{2} + \frac{m}{2^t}\right)n + O(1), \qquad \text{for } m \text{ fixed}, n\to\infty, t = \lceil \lg m\rceil; \tag{9}$$

the $O(1)$ term is an eventually periodic function of n, with period length 2^t. As $n \to \infty$ we have $C(n,n) = n\lg n + O(n)$, by Eq. (8) and exercise 5.3.1–15.

Minimum-comparison networks. Let $\hat{S}(n)$ be the minimum number of comparators needed in a sorting network for n elements; clearly $\hat{S}(n) \geq S(n)$, where $S(n)$ is the minimum number of comparisons needed in an not-necessarily-oblivious sorting procedure (see Section 5.3.1). We have $\hat{S}(4) = 5 = S(4)$, so the new constraint causes no loss of efficiency when $n = 4$; but already when $n = 5$ it turns out that $\hat{S}(5) = 9$ while $S(5) = 7$. The problem of determining $\hat{S}(n)$ seems to be even harder than the problem of determining $S(n)$; even the asymptotic behavior of $\hat{S}(n)$ is still unknown.

It is interesting to trace the history of this problem, since each step was forged with some difficulty. Sorting networks were first explored by P. N. Armstrong, R. J. Nelson, and D. J. O'Connor, about 1954 [see *U.S. Patent 3029413*]; in the words of their patent attorney, "By the use of skill, it is possible to design economical n-line sorting switches using a reduced number of two-line sorting switches." After observing that $\hat{S}(n+1) \leq \hat{S}(n) + n$, they gave special constructions for $4 \leq n \leq 8$, using 5, 9, 12, 18, and 19 comparators, respectively.

Then Nelson worked together with R. C. Bose to show that $\hat{S}(2^n) \le 3^n - 2^n$ for all n; hence $\hat{S}(n) = O(n^{\lg 3}) = O(n^{1.585})$. Bose and Nelson published their interesting method in *JACM* **9** (1962), 282–296, where they conjectured that it was best possible; T. N. Hibbard [*JACM* **10** (1963), 142–150] found a similar but slightly simpler construction that used the same number of comparisons, thereby reinforcing the conjecture.

In 1964, R. W. Floyd and D. E. Knuth found a new way to approach the problem, leading to an asymptotic bound of the form $\hat{S}(n) = O\bigl(n^{1+c/\sqrt{\log n}}\bigr)$. Working independently, K. E. Batcher discovered the general merging strategy outlined above. Using a number of comparators defined by the recursion

$$c(1) = 0, \quad c(n) = c\bigl(\lceil n/2 \rceil\bigr) + c\bigl(\lfloor n/2 \rfloor\bigr) + C\bigl(\lceil n/2 \rceil, \lfloor n/2 \rfloor\bigr) \quad \text{for } n \ge 2, \tag{10}$$

he proved (see exercise 5.2.2–14) that

$$c(2^t) = (t^2 - t + 4)2^{t-2} - 1;$$

consequently $\hat{S}(n) = O\bigl(n(\log n)^2\bigr)$. Neither Floyd and Knuth nor Batcher published their constructions until some time later [*Notices of the Amer. Math. Soc.* **14** (1967), 283; *Proc. AFIPS Spring Joint Computer Conf.* **32** (1968), 307–314].

Several people have found ways to reduce the number of comparators used by Batcher's merge-exchange construction; the following table shows the best upper bounds currently known for $\hat{S}(n)$:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$c(n)$	0	1	3	5	9	12	16	19	26	31	37	41	48	53	59	63
$\hat{S}(n) \le$	0	1	3	5	9	12	16	19	25	29	35	39	45	51	56	60

(11)

Since $\hat{S}(n) < c(n)$ for $8 < n \le 16$, merge exchange is nonoptimal for all $n > 8$. When $n \le 8$, merge exchange uses the same number of comparators as the construction of Bose and Nelson. Floyd and Knuth proved in 1964–1966 that the values listed for $\hat{S}(n)$ are *exact* when $n \le 8$ [see *A Survey of Combinatorial Theory* (North-Holland, 1973), 163–172]; the values of $\hat{S}(n)$ for $n > 8$ are still not known.

Constructions that lead to the values in (11) are shown in Fig. 49. The network for $n = 9$, based on an interesting three-way merge, was found by R. W. Floyd in 1964; its validity can be established by using the general principle described in exercise 27. The network for $n = 10$ was discovered by A. Waksman in 1969, by regarding the inputs as permutations of $\{1, 2, \ldots, 10\}$ and trying to reduce as much as possible the number of values that can appear on each line at a given stage, while maintaining some symmetry.

The network shown for $n = 13$ has quite a different pedigree: Hughes Juillé [*Lecture Notes in Comp. Sci.* **929** (1995), 246–260] used a computer program to construct it, by simulating an evolutionary process of genetic breeding. The network exhibits no obvious rhyme or reason, but it works — and it's shorter than any other construction devised so far by human ratiocination.

A 62-comparator sorting network for 16 elements was found by G. Shapiro in 1969, and this was rather surprising since Batcher's method (63 comparisons)

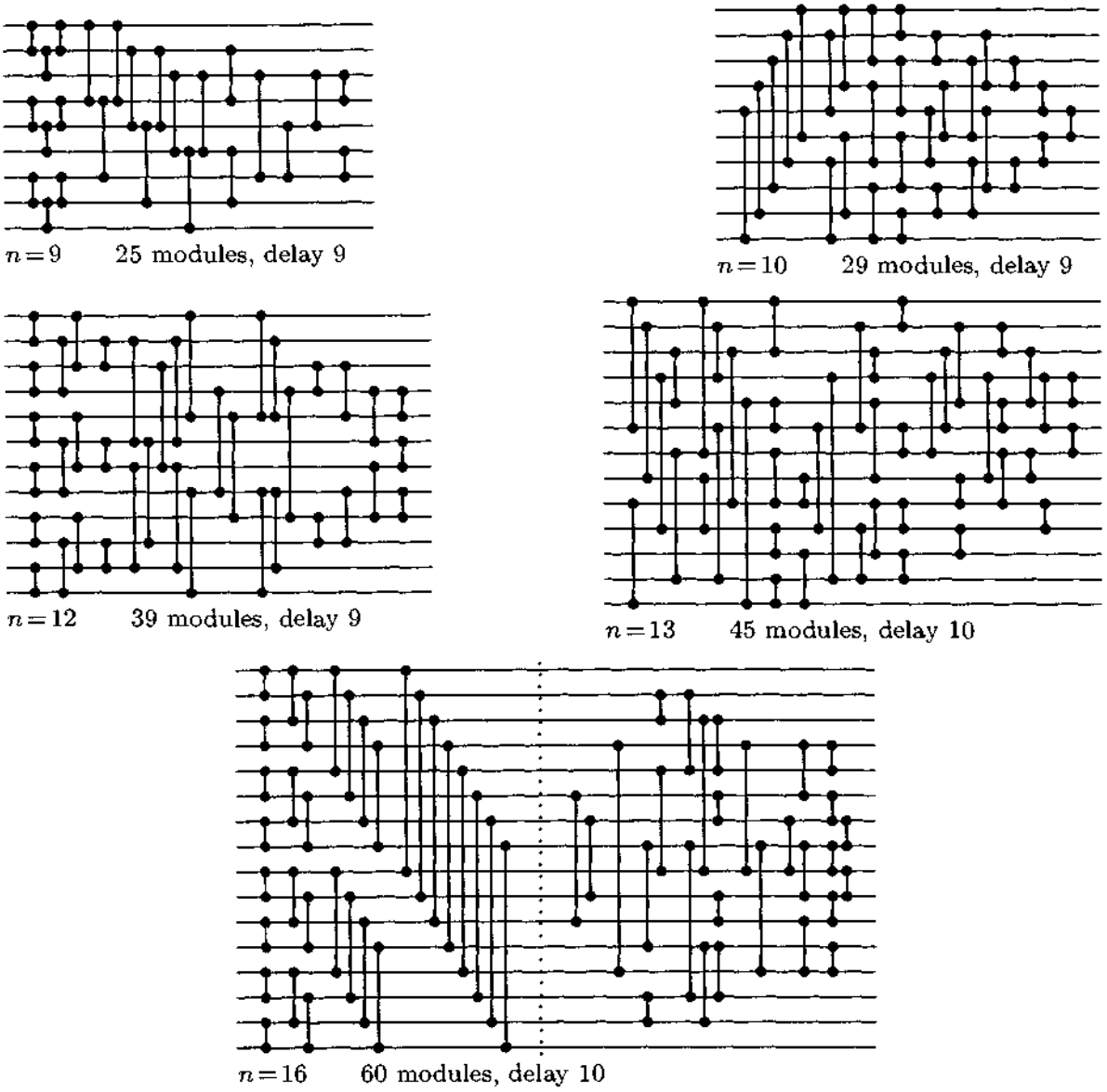


Fig. 49. Efficient sorting networks.

would appear to be at its best when n is a power of 2. Soon after hearing of Shapiro's construction, M. W. Green tripled the amount of surprise by finding the 60-comparison sorter in Fig. 49. The first portion of Green's construction is fairly easy to understand; after the 32 comparison/interchanges to the left of the dotted line have been made, the lines can be labeled with the 16 subsets of $\{a, b, c, d\}$, in such a way that the line labeled s is known to contain a number less than or equal to the contents of the line labeled t whenever s is a subset of t. The state of the sort at this point is discussed further in exercise 32. Comparisons made on subsequent levels of Green's network become increasingly mysterious, however, and as yet nobody has seen how to generalize the construction in order to obtain correspondingly efficient networks for higher values of n.

Shapiro and Green also discovered the network shown for $n = 12$. When $n = 11$, 14, or 15, good networks can be found by removing the bottom line of the network for $n + 1$, together with all comparators touching that line.

The best sorting network currently known for 256 elements, due to D. Van Voorhis, shows that $\hat{S}(256) \leq 3651$, compared to 3839 by Batcher's method. [See R. L. Drysdale and F. H. Young, *SICOMP* **4** (1975), 264–270.] As $n \to \infty$, it turns out in fact that $\hat{S}(n) = O(n \log n)$; this astonishing upper bound was proved by Ajtai, Komlós, and Szemerédi in *Combinatorica* **3** (1983), 1–19. The networks they constructed are not of practical interest, since many comparators were introduced just to save a factor of $\log n$; Batcher's method is much better, unless n exceeds the total memory capacity of all computers on earth! But the theorem of Ajtai, Komlós, and Szemerédi does establish the true asymptotic growth rate of $\hat{S}(n)$, up to a constant factor.

Minimum-time networks. In physical realizations of sorting networks, and on parallel computers, it is possible to do nonoverlapping comparison-exchanges at the same time; therefore it is natural to try to minimize the delay time. A moment's reflection shows that the delay time of a sorting network is equal to the maximum number of comparators in contact with any "path" through the network, if we define a path to consist of any left-to-right route that possibly switches lines at the comparators. We can put a sequence number on each comparator indicating the earliest time it can be executed; this is one higher than the maximum of the sequence numbers of the comparators that occur earlier on its input lines. (See Fig. 50(a); part (b) of the figure shows the same network redrawn so that each comparison is done at the earliest possible moment.)

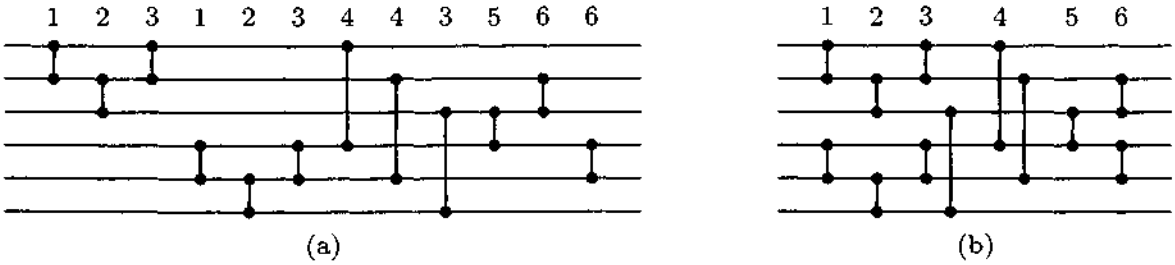


Fig. 50. Doing each comparison at the earliest possible time.

Batcher's odd-even merging network described above takes $T_B(m, n)$ units of time, where $T_B(m, 0) = T_B(0, n) = 0$, $T_B(1, 1) = 1$, and

$$T_B(m, n) = 1 + \max\bigl(T_B(\lfloor m/2 \rfloor, \lfloor n/2 \rfloor), T_B(\lceil m/2 \rceil, \lceil n/2 \rceil)\bigr) \quad \text{for } mn \geq 2.$$

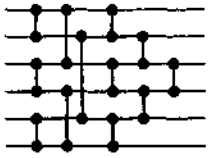
We can use these relations to prove that $T_B(m, n+1) \geq T_B(m, n)$, by induction; hence $T_B(m, n) = 1 + T_B\bigl(\lceil m/2 \rceil, \lceil n/2 \rceil\bigr)$ for $mn \geq 2$, and it follows that

$$T_B(m, n) = 1 + \bigl\lceil \lg \max(m, n) \bigr\rceil, \qquad \text{for } mn \geq 1. \tag{12}$$

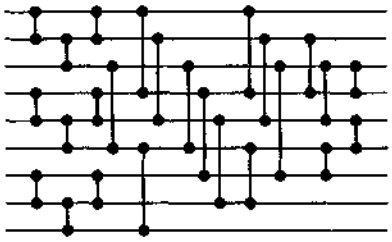
Exercise 5 shows that Batcher's sorting method therefore has a delay time of

$$\binom{1 + \lceil \lg n \rceil}{2}. \tag{13}$$

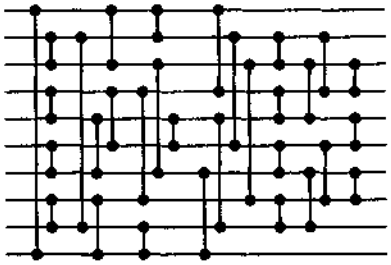
Let $\hat{T}(n)$ be the minimum achievable delay time in any sorting network for n elements. It is possible to improve some of the networks described above so



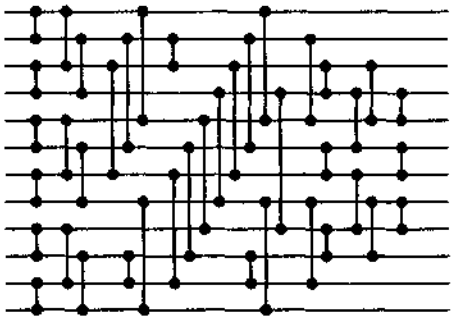
$n=6$ 12 modules, delay 5



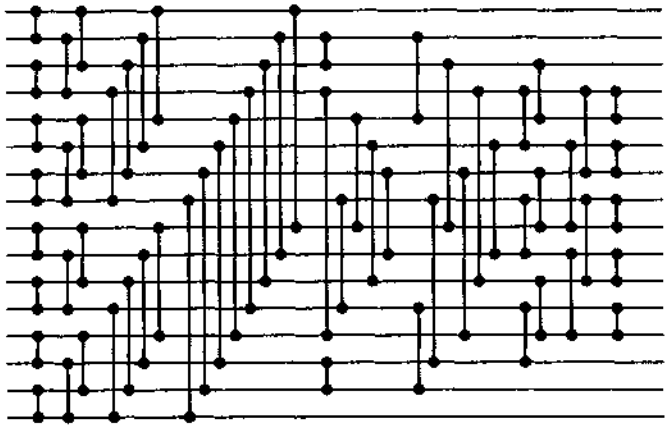
$n=9$ 25 modules, delay 8



$n=10$ 31 modules, delay 7



$n=12$ 40 modules, delay 8



$n=16$ 61 modules, delay 9

Fig. 51. Sorting networks that are the fastest known, when comparisons are performed in parallel.

that they have smaller delay time but use no more comparators, as shown for $n = 6$ and $n = 9$ in Fig. 51, and for $n = 10$ in exercise 7. Still smaller delay time can be achieved if we add one or two extra comparator modules, as shown in the remarkable networks for $n = 10$, 12, and 16 in Fig. 51. These constructions yield the following upper bounds on $\hat{T}(n)$ for small n:

$$\begin{array}{rl} n & = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16 \\ \hat{T}(n) & \le 0\ 1\ 3\ 3\ 5\ 5\ 6\ 6\ 7\ \ 7\ \ 8\ \ 8\ \ 9\ \ 9\ \ 9\ \ 9 \end{array} \tag{14}$$

For $n \le 10$ the values given here are known to be exact (see exercise 4). The networks in Fig. 51 merit careful study, because it is by no means obvious that they always sort; they were discovered in 1969–1971 by G. Shapiro ($n = 6$, 9, 12) and D. Van Voorhis ($n = 10$, 16).

Merging networks. Let $\hat{M}(m,n)$ denote the minimum number of comparator modules needed in a network that merges m elements $x_1 \le \cdots \le x_m$ with n elements $y_1 \le \cdots \le y_n$ to form the sorted sequence $z_1 \le \cdots \le z_{m+n}$. At present no merging networks have been discovered that are superior to the odd-even merge described above; hence the function $C(m,n)$ in (6) represents the best upper bound known for $\hat{M}(m,n)$.

R. W. Floyd has discovered an interesting way to find *lower* bounds for this merging problem.

Theorem F. $\hat{M}(2n,2n) \ge 2\hat{M}(n,n) + n$, *for all* $n \ge 1$.

Proof. Consider a network with $\hat{M}(2n,2n)$ comparator modules, capable of sorting all input sequences $\langle z_1, \ldots, z_{4n}\rangle$ such that $z_1 \le z_3 \le \cdots \le z_{4n-1}$ and $z_2 \le z_4 \le \cdots \le z_{4n}$. We may assume that each module replaces (z_i, z_j) by $\big(\min(z_i,z_j), \max(z_i,z_j)\big)$, for some $i < j$ (see exercise 16). The comparators can therefore be divided into three classes:

a) $i \le 2n$ and $j \le 2n$.

b) $i > 2n$ and $j > 2n$.

c) $i \le 2n$ and $j > 2n$.

Class (a) must contain at least $\hat{M}(n,n)$ comparators, since $z_{2n+1}, z_{2n+2}, \ldots, z_{4n}$ may be already in their final position when the merge starts; similarly, there are at least $\hat{M}(n,n)$ comparators in class (b). Furthermore the input sequence $\langle 0,1,0,1,\ldots,0,1\rangle$ shows that class (c) contains at least n comparators, since n zeros must move from $\{z_{2n+1},\ldots,z_{4n}\rangle$ to $\{z_1,\ldots,z_{2n}\rangle$. ∎

Repeated use of Theorem F proves that $\hat{M}(2^m,2^m) \ge \frac{1}{2}(m+2)2^m$; hence $\hat{M}(n,n) \ge \frac{1}{2}n\lg n + O(n)$. We know from Theorem 5.3.2M that merging *without* the network restriction requires only $M(n,n) = 2n-1$ comparisons; hence we have proved that merging with networks is intrinsically harder than merging in general.

The odd-even merge shows that

$$\hat{M}(m,n) \le C(m,n) = \tfrac{1}{2}(m+n)\lg\min(m,n) + O(m+n).$$

P. B. Miltersen, M. Paterson, and J. Tarui [*JACM* **43** (1996), 147–165] have improved Theorem F by establishing the lower bound

$$\hat{M}(m,n) \ge \tfrac{1}{2}\big((m+n)\lg(m+1) - m/\ln 2\big) \qquad \text{for } 1 \le m \le n.$$

Consequently $\hat{M}(m,n) = \frac{1}{2}(m+n)\lg\min(m,n) + O(m+n)$.

The exact formula $\hat{M}(2,n) = C(2,n) = \lceil \frac{3}{2}n \rceil$ has been proved by A. C. Yao and F. F. Yao [*JACM* **23** (1976), 566–571]. The value of $\hat{M}(m,n)$ is also known to equal $C(m,n)$ for $m = n \le 5$; see exercise 9.

Bitonic sorting. When simultaneous comparisons are allowed, we have seen in Eq. (12) that the odd-even merge uses $\lceil \lg(2n) \rceil$ units of delay time, when $1 \le m \le n$. Batcher has devised another type of network for merging, called a

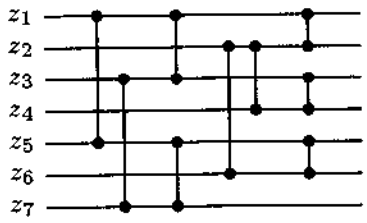


Fig. 52. Batcher's bitonic sorter of order 7.

bitonic sorter, which lowers the delay time to $\lceil \lg(m+n) \rceil$ although it requires more comparator modules. [See *U.S. Patent 3428946* (1969).]

Let us say that a sequence $\langle z_1, \ldots, z_p \rangle$ of p numbers is *bitonic* if $z_1 \geq \cdots \geq z_k \leq \cdots \leq z_p$ for some k, $1 \leq k \leq p$. (Compare this with the ordinary definition of "monotonic" sequences.) A bitonic sorter of order p is a comparator network that is capable of sorting any bitonic sequence of length p into nondecreasing order. The problem of merging $x_1 \leq \cdots \leq x_m$ with $y_1 \leq \cdots \leq y_n$ is a special case of the bitonic sorting problem, since merging can be done by applying a bitonic sorter of order $m+n$ to the sequence $\langle x_m, \ldots, x_1, y_1, \ldots, y_n \rangle$.

Notice that when a sequence $\langle z_1, \ldots, z_p \rangle$ is bitonic, so are all of its subsequences. Shortly after Batcher discovered the odd-even merging networks, he observed that we can construct a bitonic sorter of order p in an analogous way, by first sorting the bitonic subsequences $\langle z_1, z_3, z_5, \ldots \rangle$ and $\langle z_2, z_4, z_6, \ldots \rangle$ independently, then comparing and interchanging $z_1 : z_2$, $z_3 : z_4$, ... (See exercise 10 for a proof.) If $C'(p)$ is the corresponding number of comparator modules, we have

$$C'(p) = C'(\lceil p/2 \rceil) + C'(\lfloor p/2 \rfloor) + \lfloor p/2 \rfloor, \qquad \text{for } p \geq 2; \tag{15}$$

and the delay time is clearly $\lceil \lg p \rceil$. Figure 52 shows the bitonic sorter of order 7 constructed in this way: It can be used as a (3,4)- as well as a (2,5)-merging network, with three units of delay; the odd-even merge for $m = 2$ and $n = 5$ saves one comparator but adds one more level of delay.

Batcher's bitonic sorter of order 2^t is particularly interesting; it consists of t levels of 2^{t-1} comparators each. If we number the input lines $z_0, z_1, \ldots, z_{2^t-1}$, element z_i is compared to z_j on level l if and only if i and j differ only in the lth most significant bit of their binary representations. This simple structure leads to parallel sorting networks that are as fast as merge exchange, Algorithm 5.2.2M, but considerably easier to implement. (See exercises 11 and 13.)

Bitonic merging is optimum, in the sense that no parallel merging method based on simultaneous disjoint comparisons can sort in fewer than $\lceil \lg(m+n) \rceil$ stages, whether it works obliviously or not. (See exercise 46.) Another way to achieve this optimum time, with fewer comparisons but a slightly more complicated control logic, is discussed in exercise 57.

When $1 \leq m \leq n$, the nth smallest output of an (m,n) merging network depends on $2m + [m < n]$ of the inputs (see exercise 29). If it can be computed by comparators with l levels of delay, it involves at most 2^l of the inputs; hence $2^l \geq 2m + [m < n]$, and $l \geq \lceil \lg(2m + [m < n]) \rceil$. Batcher has shown [Report GER-14122 (Akron, Ohio: Goodyear Aerospace Corporation, 1968)] that this

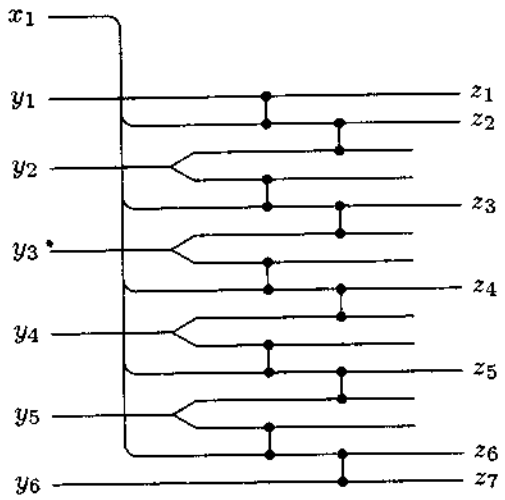


Fig. 53. Merging one item with six others, with multiple fanout, in order to achieve the minimum possible delay time.

minimum delay time is achievable if we allow "multiple fanout" in the network, namely the splitting of lines so that the same number is fed to many modules at once. For example, one of his networks, capable of merging one item with n others after only two levels of delay, is illustrated for $n = 6$ in Fig. 53. Of course, networks with multiple fanout do not conform to our conventions, and it is fairly easy to see that any $(1, n)$ merging network without multiple fanout must have a delay time of $\lg(n + 1)$ or more. (See exercise 45.)

Selection networks. We can also use networks to approach the problem of Section 5.3.3. Let $\hat{U}_t(n)$ denote the minimum number of comparators required in a network that moves the t largest of n distinct inputs into t specified output lines; the numbers are allowed to appear in any order on these output lines. Let $\hat{V}_t(n)$ denote the minimum number of comparators required to move the tth largest of n distinct inputs into a specified output line; and let $\hat{W}_t(n)$ denote the minimum number of comparators required to move the t largest of n distinct inputs into t specified output lines in nondecreasing order. It is not difficult to deduce (see exercise 17) that

$$\hat{U}_t(n) \le \hat{V}_t(n) \le \hat{W}_t(n). \tag{16}$$

Suppose first that we have $2t$ elements $\langle x_1, \ldots, x_{2t}\rangle$ and we wish to select the largest t. V. E. Alekseev [*Kibernetika* **5**, 5 (1969), 99–103] has observed that we can do the job by first sorting $\langle x_1, \ldots, x_t\rangle$ and $\langle x_{t+1}, \ldots, x_{2t}\rangle$, then comparing and interchanging

$$x_1 : x_{2t}, \qquad x_2 : x_{2t-1}, \qquad \ldots, \qquad x_t : x_{t+1}. \tag{17}$$

Since none of these pairs can contain more than one of the largest t elements (why?), Alekseev's procedure must select the largest t elements.

If we want to select the t largest of nt elements, we can apply Alekseev's procedure $n - 1$ times, eliminating t elements each time; hence

$$\hat{U}_t(nt) \le (n - 1)\big(2\hat{S}(t) + t\big). \tag{18}$$

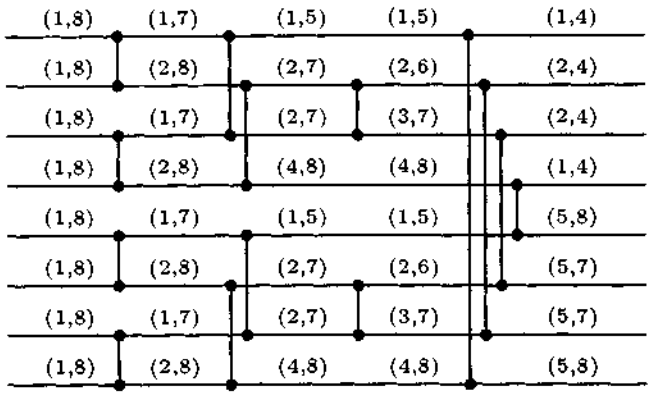


Fig. 54. Separating the largest four from the smallest four. (Numbers on these lines are used in the proof of Theorem A.)

Alekseev also derived an interesting *lower* bound for the selection problem:

Theorem A. $\hat{U}_t(n) \geq (n-t)\lceil \lg(t+1) \rceil$.

Proof. It is most convenient to consider the equivalent problem of selecting the *smallest* t elements. We can attach numbers (l, u) to each line of a comparator network, as shown in Fig. 54, where l and u denote respectively the minimum and maximum values that can appear at that position when the input is a permutation of $\{1, 2, \ldots, n\}$. Let l_i and l_j be the lower bounds on lines i and j before a comparison of $x_i : x_j$, and let l_i' and l_j' be the corresponding lower bounds after the comparison. It is obvious that $l_i' = \min(l_i, l_j)$; exercise 24 proves the (nonobvious) relation

$$l_j' \leq l_i + l_j. \tag{19}$$

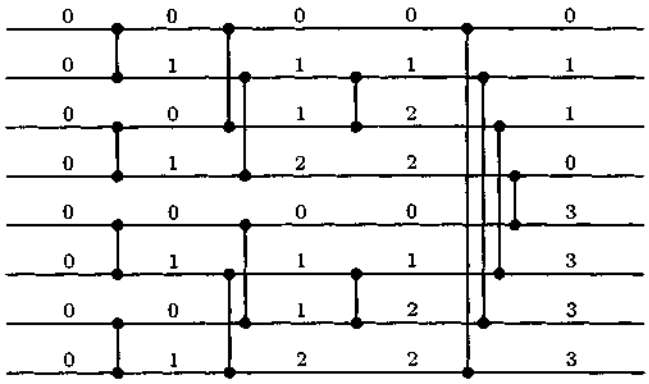


Fig. 55. Another interpretation for the network of Fig. 54.

Now let us reinterpret the network operations in another way (see Fig. 55): All input lines are assumed to contain zero, and each "comparator" now places the smaller of its inputs on the upper line and the larger *plus one* on the lower line. The resulting numbers $\langle m_1, m_2, \ldots, m_n \rangle$ have the property that

$$2^{m_i} \geq l_i \tag{20}$$

Table 1

COMPARISONS NEEDED IN SELECTION NETWORKS ($\hat{U}_t(n)$, $\hat{V}_t(n)$, $\hat{W}_t(n)$)

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
$n=1$	(0,0,0)					
$n=2$	(1,1,1)	(0,1,1)				
$n=3$	(2,2,2)	(2,3,3)	(0,2,3)			
$n=4$	(3,3,3)	(4,5,5)	(3,5,5)	(0,3,5)		
$n=5$	(4,4,4)	(6,7,7)	(6,7,8)	(4,7,9)	(0,4,9)	
$n=6$	(5,5,5)	(8,9,9)	(8,10,10)	(8,10,12)	(5,9,12)	(0,5,12)

throughout the network, since this holds initially and it is preserved by each comparator because of (19). Furthermore, the final value of

$$m_1 + m_2 + \cdots + m_n$$

is the total number of comparators in the network, since each comparator adds unity to this sum.

If the network selects the smallest t numbers, $n-t$ of the l_i are $\geq t+1$; hence $n-t$ of the m_i must be $\geq \lceil \lg(t+1) \rceil$. ∎

The lower bound in Theorem A turns out to be exact when $t=1$ and when $t=2$ (see exercise 19). Table 1 gives some values of $\hat{U}_t(n)$, $\hat{V}_t(n)$, and $\hat{W}_t(n)$ for small t and n. Andrew Yao [Ph.D. thesis, U. of Illinois (1975)] determined the asymptotic behavior of $\hat{U}_t(n)$ for fixed t, by showing that $\hat{U}_3(n) = 2n + \lg n + O(1)$ and $\hat{U}_t(n) = n\lceil \lg(t+1) \rceil + O\big((\log n)^{\lfloor \lg t \rfloor}\big)$ as $n \to \infty$; the minimum delay time is $\lg n + \lfloor \lg t \rfloor \lg\lg n + O(\log\log\log n)$. N. Pippenger [*SICOMP* **20** (1991), 878–887] has proved by nonconstructive methods that for any $\epsilon > 0$ there exist selection networks with $\hat{U}_{\lceil n/2 \rceil}(n) \leq (2+\epsilon) n \lg n$, whenever n is sufficiently large (depending on ϵ).

EXERCISES — First Set

Several of the following exercises develop the theory of sorting networks in detail, and it is convenient to introduce some notation. We let $[i:j]$ stand for a comparison/interchange module. A network with n inputs and r comparator modules is written $[i_1:j_1][i_2:j_2]\ldots[i_r:j_r]$, where each of the i's and j's is $\leq n$; we shall call it an *n-network* for short. A network is called *standard* if $i_q < j_q$ for $1 \leq q \leq r$. Thus, for example, Fig. 44 on page 221 depicts a standard 4-network, denoted by the comparator sequence [1:2][3:4][1:3][2:4][2:3].

The text's convention for drawing network diagrams represents only standard networks; all comparators $[i:j]$ are represented by a line from i to j, where $i < j$. When nonstandard networks must be drawn, we can use an *arrow* from i to j, indicating that the larger number goes to the point of the arrow. For example, Fig. 56 illustrates a nonstandard network for 16 elements, whose comparators are [1:2][4:3][5:6][8:7] etc. Exercise 11 proves that Fig. 56 is a sorting network.

If $x = \langle x_1, \ldots, x_n \rangle$ is an n-vector and α is an n-network, we write $x\alpha$ for the vector of numbers $\langle (x\alpha)_1, \ldots, (x\alpha)_n \rangle$ produced by the network. For brevity, we also let $a \vee b = \max(a,b)$, $a \wedge b = \min(a,b)$, $\bar{a} = 1-a$. Thus $(x[i:j])_i = x_i \wedge x_j$, $(x[i:j])_j = x_i \vee x_j$,