5.3. OPTIMUM SORTING

Now that we have analyzed a great many methods for internal sorting, it is time to turn to a broader question: *What is the best possible way to sort?* Can we place limits on the maximum sorting speeds that will ever be achievable, no matter how clever a programmer might be?

Of course there *is* no best possible way to sort; we must define precisely what is meant by "best," and there is no best possible way to define "best." We have discussed similar questions about the theoretical optimality of algorithms in Sections 4.3.3, 4.6.3, and 4.6.4, where high-precision multiplication and polynomial evaluation were considered. In each case it was necessary to formulate a rather simple definition of a "best possible" algorithm, in order to give sufficient structure to the problem to make it workable. And in each case we ran into interesting problems that are so difficult they still haven't been completely resolved. The same situation holds for sorting; some very interesting discoveries have been made, but many fascinating questions remain unanswered.

Studies of the inherent complexity of sorting have usually been directed towards minimizing the number of times we make comparisons between keys while sorting n items, or merging m items with n, or selecting the tth largest of an unordered set of n items. Sections 5.3.1, 5.3.2, and 5.3.3 discuss these questions in general, and Section 5.3.4 deals with similar issues under the interesting restriction that the pattern of comparisons must essentially be fixed in advance. Several other types of interesting theoretical questions related to optimum sorting appear in the exercises for Section 5.3.4, and in the discussion of external sorting (Sections 5.4.4, 5.4.8, and 5.4.9).

As soon as an Analytical Engine exists,
it will necessarily guide the future course of the science.
Whenever any result is sought by its aid,
the question will then arise —
By what course of calculation can these
results be arrived at by the machine
in the shortest *time?*

— CHARLES BABBAGE (1864)

5.3.1. Minimum-Comparison Sorting

The minimum number of key comparisons needed to sort n elements is obviously *zero*, because we have seen radix methods that do no comparisons at all. In fact, it is possible to write `MIX` programs that are able to sort, although they contain no conditional jump instructions at all! (See exercise 5–8 at the beginning of this chapter.) We have also seen several sorting methods that are based essentially on comparisons of keys, yet their running time in practice is dominated by other considerations such as data movement, housekeeping operations, etc.

Therefore it is clear that comparison counting is not the only way to measure the effectiveness of a sorting method. But it is fun to scrutinize the number of comparisons anyway, since a theoretical study of this subject gives us a good

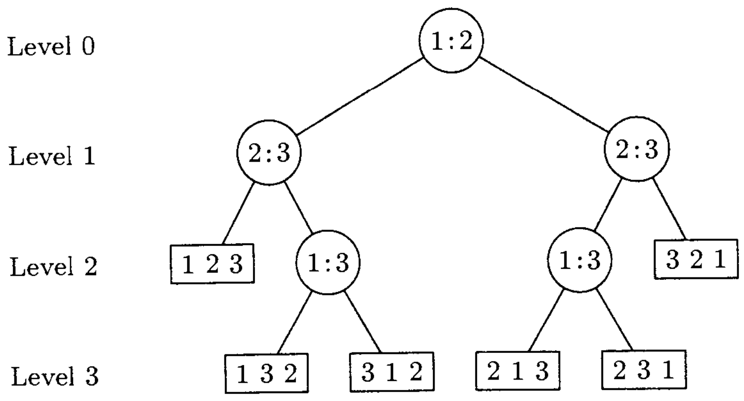


Fig. 34. A comparison tree for sorting three elements.

deal of useful insight into the nature of sorting processes, and it also helps us to sharpen our wits for the more mundane problems that confront us at other times.

In order to rule out radix-sorting methods, which do no comparisons at all, we shall restrict our discussion to sorting techniques that are based solely on an abstract linear ordering relation "$<$" between keys, as discussed at the beginning of this chapter. For simplicity, we shall also confine our discussion to the case of *distinct* keys, so that there are only two possible outcomes of any comparison of K_i versus K_j: either $K_i < K_j$ or $K_i > K_j$. (For an extension of the theory to the general case where equal keys are allowed, see exercises 3 through 12. For bounds on the worst-case running time that is needed to sort integers without the restriction to comparison-based methods, see Fredman and Willard, *J. Computer and Syst. Sci.* **47** (1993), 424–436; Ben-Amram and Galil, *J. Comp. Syst. Sci.* **54** (1997), 345–370; Thorup, *SODA* **9** (1998), 550–555.)

The problem of sorting by comparisons can also be expressed in other equivalent ways. Given a set of n distinct weights and a balance scale, we can ask for the least number of weighings necessary to completely rank the weights in order of magnitude, when the pans of the balance scale can each accommodate only one weight. Alternatively, given a set of n players in a tournament, we can ask for the smallest number of games that suffice to rank all contestants, assuming that the strengths of the players can be linearly ordered (with no ties).

All n-element sorting methods that satisfy the constraints above can be represented in terms of an extended binary tree structure such as that shown in Fig. 34. Each *internal node* (drawn as a circle) contains two indices "$i\!:\!j$" denoting a comparison of K_i versus K_j. The left subtree of this node represents the subsequent comparisons to be made if $K_i < K_j$, and the right subtree represents the actions to be taken when $K_i > K_j$. Each *external node* of the tree (drawn as a box) contains a permutation $a_1 a_2 \ldots a_n$ of $\{1, 2, \ldots, n\}$, denoting the fact that the ordering

$$K_{a_1} < K_{a_2} < \cdots < K_{a_n}$$

has been established. (If we look at the path from the root to this external node, each of the $n-1$ relationships $K_{a_i} < K_{a_{i+1}}$ for $1 \le i < n$ will be the result of some comparison $a_i\!:\!a_{i+1}$ or $a_{i+1}\!:\!a_i$ on this path.)

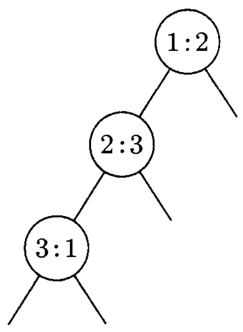


Fig. 35. Example of a redundant comparison.

Thus Fig. 34 represents a sorting method that first compares K_1 with K_2; if $K_1 > K_2$, it goes on (via the right subtree) to compare K_2 with K_3, and then if $K_2 < K_3$ it compares K_1 with K_3; finally if $K_1 > K_3$ it knows that $K_2 < K_3 < K_1$. An actual sorting algorithm will usually also move the keys around in the file, but we are interested here only in the comparisons, so we ignore all data movement. A comparison of K_i with K_j in this tree always means the *original* keys K_i and K_j, not the keys that might currently occupy the ith and jth positions of the file after the records have been shuffled around.

It is possible to make redundant comparisons; for example, in Fig. 35 there is no reason to compare 3:1, since $K_1 < K_2$ and $K_2 < K_3$ implies that $K_1 < K_3$. No permutation can possibly correspond to the left subtree of node 3:1 in Fig. 35; consequently that part of the algorithm will never be performed! Since we are interested in minimizing the number of comparisons, we may assume that no redundant comparisons are made. Hence we have an extended binary tree structure in which every external node corresponds to a permutation. All permutations of the input keys are possible, and every permutation defines a unique path from the root to an external node; it follows that *there are exactly $n!$ external nodes in a comparison tree that sorts n elements with no redundant comparisons.*

The best worst case. The first problem that arises naturally is to find comparison trees that minimize the *maximum* number of comparisons made. (Later we shall consider the *average* number of comparisons.)

Let $S(n)$ be the minimum number of comparisons that will suffice to sort n elements. If all the internal nodes of a comparison tree are at levels $< k$, it is obvious that there can be at most 2^k external nodes in the tree. Hence, letting $k = S(n)$, we have

$$n! \le 2^{S(n)}.$$

Since $S(n)$ is an integer, we can rewrite this formula to obtain the lower bound

$$S(n) \ge \lceil \lg n! \rceil. \qquad (1)$$

Stirling's approximation tells us that

$$\lceil \lg n! \rceil = n \lg n - n/\ln 2 + \tfrac{1}{2} \lg n + O(1), \qquad (2)$$

hence roughly $n \lg n$ comparisons are needed.

Relation (1) is often called the *information-theoretic lower bound*, since cognoscenti of information theory would say that $\lg n!$ "bits of information" are being acquired during a sorting process; each comparison yields at most one bit of information. Trees such as Fig. 34 have also been called "questionnaires"; their mathematical properties were first explored systematically in Claude Picard's book *Théorie des Questionnaires* (Paris: Gauthier-Villars, 1965).

Of all the sorting methods we have seen, the three that require fewest comparisons are binary insertion (see Section 5.2.1), tree selection (see Section 5.2.3), and straight two-way merging (see Algorithm 5.2.4L). The maximum number of comparisons for binary insertion is readily seen to be

$$B(n) = \sum_{k=1}^{n} \lceil \lg k \rceil = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1, \tag{3}$$

by exercise 1.2.4–42, and the maximum number of comparisons in two-way merging is given in exercise 5.2.4–14. We will see in Section 5.3.3 that tree selection has the same bound on its comparisons as either binary insertion or two-way merging, depending on how the tree is set up. In all three cases we achieve an asymptotic value of $n \lg n$; combining these lower and upper bounds for $S(n)$ proves that

$$\lim_{n\to\infty} \frac{S(n)}{n \lg n} = 1. \tag{4}$$

Thus we have an approximate formula for $S(n)$, but it is desirable to obtain more precise information. The following table gives exact values of the lower and upper bounds discussed above, for small n:

$n =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\lceil \ln n! \rceil =$	0	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49
$B(n) =$	0	1	3	5	8	11	14	17	21	25	29	33	37	41	45	49	54
$L(n) =$	0	1	3	5	9	11	14	17	25	27	30	33	38	41	45	49	65

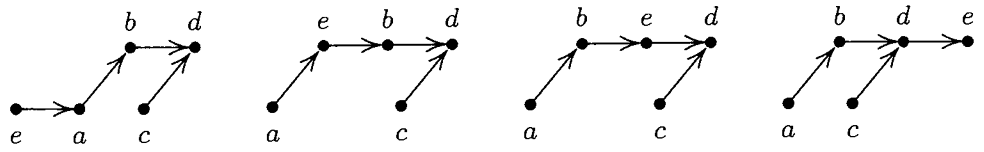
Here $B(n)$ and $L(n)$ refer respectively to binary insertion and two-way list merging. It can be shown that $B(n) \le L(n)$ for all n (see exercise 2).

From the table above, we can see that $S(4) = 5$, but $S(5)$ might be either 7 or 8. This brings us back to a problem stated at the beginning of Section 5.2: What is the best way to sort five elements? Can five elements be sorted using only seven comparisons?

The answer is yes, but a seven-step procedure is not especially easy to discover. We begin as if we were sorting four elements by merging, first comparing $K_1\!:\!K_2$, then $K_3\!:\!K_4$, then the larger elements of these pairs. This produces a configuration that may be diagrammed as

$$\begin{tikzcd} b \arrow[r] & d & \\ a \arrow[u] & c \arrow[u] & e \end{tikzcd} \tag{5}$$

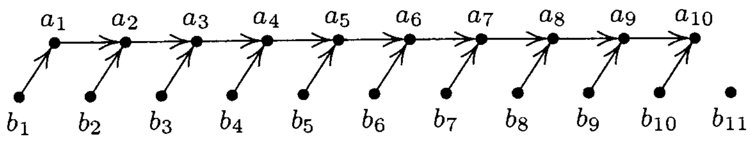
indicating that $a < b < d$ and $c < d$. (It is convenient to represent known ordering relations between elements by drawing directed graphs such as this, where x is known to be less than y if and only if there is a path from x to y in the graph.) At this point we insert the fifth element $K_5 = e$ into its proper place among $\{a, b, d\}$; only two comparisons are needed, since we may compare it first with b and then with a or d. This leaves one of four possibilities,



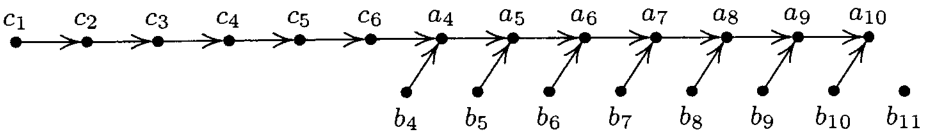
(6)

and in each case we can insert c among the remaining elements less than d in two more comparisons. This method for sorting five elements was first found by H. B. Demuth [Ph.D. thesis, Stanford University (1956), 41–43].

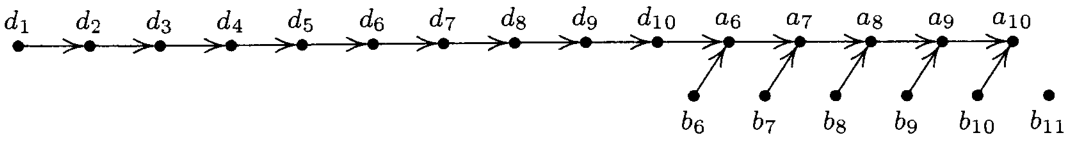
Merge insertion. A pleasant generalization of the method above has been discovered by Lester Ford, Jr. and Selmer Johnson. Since it involves some aspects of merging and some aspects of insertion, we shall call it *merge insertion.* For example, consider the problem of sorting 21 elements. We start by comparing the ten pairs $K_1 : K_2, K_3 : K_4, \ldots, K_{19} : K_{20}$; then we sort the ten larger elements of the pairs, using merge insertion. As a result we obtain the configuration


a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10}
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11}
(7)

analogous to (5). The next step is to insert b_3 among $\{b_1, a_1, a_2\}$, then b_2 among the other elements less than a_2; we arrive at the configuration


c_1 c_2 c_3 c_4 c_5 c_6 a_4 a_5 a_6 a_7 a_8 a_9 a_{10}
b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11}
(8)

Let us call the upper-line elements the *main chain.* We can insert b_5 into its proper place in the main chain, using three comparisons (first comparing it to c_4, then c_2 or c_6, etc.); then b_4 can be moved into the main chain in three more steps, leading to


d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9 d_{10} a_6 a_7 a_8 a_9 a_{10}
b_6 b_7 b_8 b_9 b_{10} b_{11}
(9)

The next step is crucial; is it clear what to do? We insert b_{11} (*not* b_7) into the main chain, using only four comparisons. Then b_{10}, b_9, b_8, b_7, b_6 (in this order) can also be inserted into their proper places in the main chain, using at most four comparisons each.

A careful count of the comparisons involved here shows that the 21 elements have been sorted in at most $10 + S(10) + 2 + 2 + 3 + 3 + 4 + 4 + 4 + 4 + 4 + 4 = 66$

steps. Since

$$2^{65} < 21! < 2^{66},$$

we also know that no fewer than 66 would be possible in any event; hence

$$S(21) = 66. \qquad (10)$$

(Binary insertion would have required 74 comparisons.)

In general, merge insertion proceeds as follows for n elements:

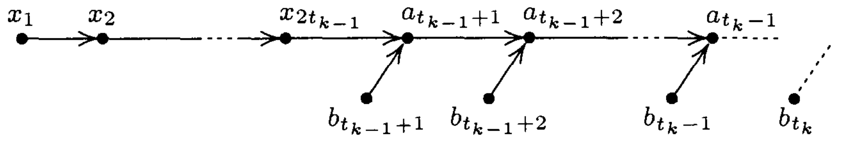
i) Make pairwise comparisons of $\lfloor n/2 \rfloor$ disjoint pairs of elements. (If n is odd, leave one element out.)

ii) Sort the $\lfloor n/2 \rfloor$ larger numbers, found in step (i), by merge insertion.

iii) Name the elements $a_1, a_2, \ldots, a_{\lfloor n/2 \rfloor}, b_1, b_2, \ldots, b_{\lceil n/2 \rceil}$ as in (7), where $a_1 \le a_2 \le \cdots \le a_{\lfloor n/2 \rfloor}$ and $b_i \le a_i$ for $1 \le i \le \lfloor n/2 \rfloor$; call b_1 and the a's the "main chain." Insert the remaining b's into the main chain, using binary insertion, in the following order, leaving out all b_j for $j > \lceil n/2 \rceil$:

$$b_3, b_2;\ b_5, b_4;\ b_{11}, b_{10}, \ldots, b_6;\ \ldots;\ b_{t_k}, b_{t_k - 1}, \ldots, b_{t_{k-1}+1};\ \ldots. \qquad (11)$$

We wish to define the sequence $(t_1, t_2, t_3, t_4, \ldots) = (1, 3, 5, 11, \ldots)$, which appears in (11), in such a way that each of $b_{t_k}, b_{t_k - 1}, \ldots, b_{t_{k-1}+1}$ can be inserted into the main chain with at most k comparisons. Generalizing (7), (8), and (9), we obtain the diagram



where the main chain up to and including $a_{t_k - 1}$ contains $2t_{k-1} + (t_k - t_{k-1} - 1)$ elements. This number must be less than 2^k; our best bet is to set it equal to $2^k - 1$, so that

$$t_{k-1} + t_k = 2^k. \qquad (12)$$

Since $t_1 = 1$, we may set $t_0 = 1$ for convenience, and we find that

$$\begin{aligned} t_k = 2^k - t_{k-1} = 2^k - 2^{k-1} + t_{k-2} = \cdots &= 2^k - 2^{k-1} + \cdots + (-1)^k 2^0 \\ &= \bigl(2^{k+1} + (-1)^k\bigr)/3 \qquad (13) \end{aligned}$$

by summing a geometric series. (Curiously, this same sequence arose in our study of an algorithm for calculating the greatest common divisor of two integers; see exercise 4.5.2–36.)

Let $F(n)$ be the number of comparisons required to sort n elements by merge insertion. Clearly

$$F(n) = \lfloor n/2 \rfloor + F(\lfloor n/2 \rfloor) + G(\lceil n/2 \rceil), \qquad (14)$$

where G represents the amount of work involved in step (iii). If $t_{k-1} \le m \le t_k$, we have

$$G(m) = \sum_{j=1}^{k-1} j(t_j - t_{j-1}) + k(m - t_{k-1}) = km - (t_0 + t_1 + \cdots + t_{k-1}), \qquad (15)$$

summing by parts. Let us set

$$w_k = t_0 + t_1 + \cdots + t_{k-1} = \lfloor 2^{k+1}/3 \rfloor, \tag{16}$$

so that $(w_0, w_1, w_2, w_3, w_4, \ldots) = (0, 1, 2, 5, 10, 21, \ldots)$. Exercise 13 shows that

$$F(n) - F(n-1) = k \qquad \text{if and only if} \qquad w_k < n \le w_{k+1}, \tag{17}$$

and the latter condition is equivalent to

$$\frac{2^{k+1}}{3} < n \le \frac{2^{k+2}}{3},$$

or $k+1 < \lg 3n \le k+2$; hence

$$F(n) - F(n-1) = \lceil \lg \tfrac{3}{4}n \rceil. \tag{18}$$

(This formula is due to A. Hadian [Ph.D. thesis, Univ. of Minnesota (1969), 38–42].) It follows that $F(n)$ has a remarkably simple expression,

$$F(n) = \sum_{k=1}^{n} \lceil \lg \tfrac{3}{4}k \rceil, \tag{19}$$

quite similar to the corresponding formula (3) for binary insertion. A closed form for this sum appears in exercise 14.

Equation (19) makes it easy to construct a table of $F(n)$; we have

$n =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\lceil \lg n! \rceil =$	0	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49
$F(n) =$	0	1	3	5	7	10	13	16	19	22	26	30	34	38	42	46	50

$n =$	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
$\lceil \lg n! \rceil =$	53	57	62	66	70	75	80	84	89	94	98	103	108	113	118	123
$F(n) =$	54	58	62	66	71	76	81	86	91	96	101	106	111	116	121	126

Notice that $F(n) = \lceil \lg n! \rceil$ for $1 \le n \le 11$ and for $20 \le n \le 21$, so we know that merge insertion is optimum for those n:

$$S(n) = \lceil \lg n! \rceil = F(n) \qquad \text{for } n = 1, \ldots, 11, 20, \text{ and } 21. \tag{20}$$

Hugo Steinhaus posed the problem of finding $S(n)$ in the second edition of his classic book *Mathematical Snapshots* (Oxford University Press, 1950), 38–39. He described the method of binary insertion, which is the best possible way to sort n objects if we start by sorting $n-1$ of them first before the nth is considered; and he conjectured that binary insertion would be optimum in general. Several years later [*Calcutta Math. Soc. Golden Jubilee Commemoration* **2** (1959), 323–327], he reported that two of his colleagues, S. Trybuła and P. Czen, had "recently" disproved his conjecture, and that they had determined $S(n)$ for $n \le 11$. Trybuła and Czen may have independently discovered the method of merge insertion, which was published soon afterwards by Ford and Johnson [*AMM* **66** (1959), 387–389].

After the discovery of merge insertion, the first unknown value of $S(n)$ was $S(12)$. Table 1 shows that 12! is quite close to 2^{29}, hence the existence of a

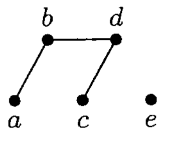
Table 1

VALUES OF FACTORIALS IN BINARY NOTATION

$(1)_2$	$= 1!$
$(10)_2$	$= 2!$
$(110)_2$	$= 3!$
$(11000)_2$	$= 4!$
$(1111000)_2$	$= 5!$
$(1011010000)_2$	$= 6!$
$(1001110110000)_2$	$= 7!$
$(1001110110000000)_2$	$= 8!$
$(1011000100110000000)_2$	$= 9!$
$(1101110101111100000000)_2$	$= 10!$
$(10011000010001010100000000)_2$	$= 11!$
$(11100100011001111110000000000)_2$	$= 12!$
$(10111001100101000110011000000000)_2$	$= 13!$
$(1010001001100001110110010100000000000)_2$	$= 14!$
$(10011000001110111011101110101100000000000)_2$	$= 15!$
$(100110000011101110111011101011000000000000000)_2$	$= 16!$
$(1010000110111111011101110110011011000000000000000)_2$	$= 17!$
$(101101011111011101100110010100111001100000000000000000)_2$	$= 18!$
$(110110000001010111001001100000110100010010000000000000000)_2$	$= 19!$
$(10000111000011011001110111110010000010101101000000000000000000)_2$	$= 20!$

29-step sorting procedure for 12 elements is somewhat unlikely. An exhaustive search (about 60 hours on a Maniac II computer) was therefore carried out by Mark Wells, who discovered that $S(12) = 30$ [*Proc. IFIP Congress 65* **2** (1965), 497–498; *Elements of Combinatorial Computing* (Pergamon, 1971), 213–215]. Thus the merge insertion procedure turns out to be optimum for $n = 12$ as well.

***A slightly deeper analysis.** In order to study $S(n)$ more carefully, let us look more closely at partial ordering diagrams such as (5). After several comparisons have been made, we can represent the knowledge we have gained in terms of a directed graph. This directed graph contains no cycles, in view of the transitivity of the $<$ relation, so we can draw it in such a way that all arcs go from left to right; it is therefore convenient to leave arrows off the diagram. In this way (5) becomes



(21)

If G is such a directed graph, let $T(G)$ be the number of permutations consistent with G, that is, the number of ways to assign the integers $\{1, 2, \ldots, n\}$ to the vertices of G so that the number on vertex x is less than the number on vertex y whenever $x \to y$ in G. For example, one of the permutations consistent with (21) has $a = 1$, $b = 4$, $c = 2$, $d = 5$, $e = 3$. We have studied $T(G)$ for various G in Section 5.1.4, where we observed that $T(G)$ is the number of ways in which G can be sorted topologically.

If G is a graph on n elements that can be obtained after k comparisons, we define the *efficiency* of G to be

$$E(G) = \frac{n!}{2^k T(G)}. \tag{22}$$

(This idea is due to Frank Hwang and Shen Lin.) Strictly speaking, the efficiency is not a function of the graph G alone, it depends on the way we arrived at G during a sorting process, but it is convenient to be a little careless in our language. After making one more comparison, between elements i and j, we obtain two graphs G_1 and G_2, one for the case $K_i < K_j$ and one for the case $K_i > K_j$. Clearly

$$T(G) = T(G_1) + T(G_2).$$

If $T(G_1) \geq T(G_2)$, we have

$$T(G) \leq 2T(G_1),$$

$$E(G_1) = \frac{n!}{2^{k+1}T(G_1)} = \frac{E(G)T(G)}{2T(G_1)} \leq E(G). \tag{23}$$

Therefore each comparison leads to at least one graph of less or equal efficiency; we can't improve the efficiency by making further comparisons.

When G has no arcs at all, we have $k = 0$ and $T(G) = n!$, so the initial efficiency is 1. At the other extreme, when G is a graph representing the final result of sorting, G looks like a straight line and $T(G) = 1$. Thus, for example, if we want to find a sorting procedure that sorts five elements in at most seven steps, we must obtain the linear graph •—•—•—•—•, whose efficiency is $5!/(2^7 \times 1) = 120/128 = 15/16$. It follows that all of the graphs arising in the sorting procedure must have efficiency $\geq \frac{15}{16}$; if any less efficient graph were to appear, at least one of its descendants would also be less efficient, and we would ultimately reach a linear graph whose efficiency is $< \frac{15}{16}$. In general, this argument proves that all graphs corresponding to the tree nodes of a sorting procedure for n elements must have efficiency $\geq n!/2^l$, where l is the number of levels of the tree (not counting external nodes). This is another way to prove that $S(n) \geq \lceil \lg n! \rceil$, although the argument is not really much different from what we said before.

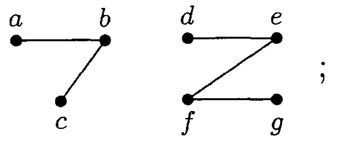
The graph (21) has efficiency 1, since $T(G) = 15$ and since G has been obtained in three comparisons. In order to see what vertices should be compared next, we can form the *comparison matrix*

$$C(G) = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \!\!\begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 15 & 10 & 15 & 11 \\ 0 & 0 & 5 & 15 & 9 \\ 5 & 10 & 0 & 15 & 9 \\ 0 & 0 & 0 & 0 & 3 \\ 4 & 8 & 6 & 12 & 0 \end{pmatrix} \end{array}, \tag{24}$$

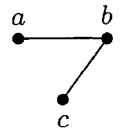
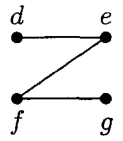
where C_{ij} is $T(G_1)$ for the graph G_1 obtained by adding the arc $i \to j$ to G. For example, if we compare K_c with K_e, the 15 permutations consistent with G

split up into $C_{ec} = 6$ having $K_e < K_c$ and $C_{ce} = 9$ having $K_c < K_e$. The latter graph would have efficiency $15/(2 \times 9) = \frac{5}{6} < \frac{15}{16}$, so it could not lead to a seven-step sorting procedure. The next comparison *must* be $K_b : K_e$ in order to keep the efficiency $\geq \frac{15}{16}$.

The concept of efficiency is especially useful when we consider the connected components of graphs. Consider for example the graph

$G =$  ;

it has two components

$G' =$  and $G'' =$ 

with no arcs connecting G' to G'', so it has been formed by making some comparisons entirely within G' and others entirely within G''. In general, assume that $G = G' \oplus G''$ has no arcs between G' and G'', where G' and G'' have respectively n' and n'' vertices; it is easy to see that

$$T(G) = \binom{n' + n''}{n'} T(G')T(G''), \tag{25}$$

since each consistent permutation of G is obtained by choosing n' elements to assign to G' and then making consistent permutations within G' and G'' independently. If k' comparisons have been made within G' and k'' within G'', we have the basic result

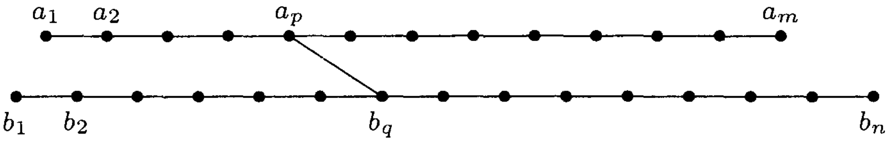
$$E(G) = \frac{(n' + n'')!}{2^{k'+k''}T(G)} = \frac{n'!}{2^{k'}T(G')}\frac{n''!}{2^{k''}T(G')} = E(G')E(G''), \tag{26}$$

showing that the efficiency of a graph is related in a simple way to the efficiency of its components. Therefore we may restrict consideration to graphs having only one component.

Now suppose that G' and G'' are one-component graphs, and suppose that we want to hook them together by comparing a vertex x of G' with a vertex y of G''. We want to know how efficient this will be. For this purpose we need a function that can be denoted by

$$\begin{pmatrix} p & < & q \\ m & & n \end{pmatrix}, \tag{27}$$

defined to be the number of permutations consistent with the graph

 (28)

Thus $\binom{p}{m} < \binom{q}{n}$ is $\binom{m+n}{m}$ times the probability that the pth smallest of a set of m numbers is less than the qth smallest of an independently chosen set of n numbers. Exercise 17 shows that we can express $\left(\begin{smallmatrix}p\\m\end{smallmatrix} < \begin{smallmatrix}q\\n\end{smallmatrix}\right)$ in two ways in terms of binomial coefficients,

$$\left(\begin{matrix}p\\m\end{matrix} < \begin{matrix}q\\n\end{matrix}\right) = \sum_{0\le k<q} \binom{m-p+n-k}{m-p}\binom{p-1+k}{p-1}$$

$$= \sum_{p\le j\le m} \binom{n-q+m-j}{n-q}\binom{q-1+j}{q-1}. \qquad (29)$$

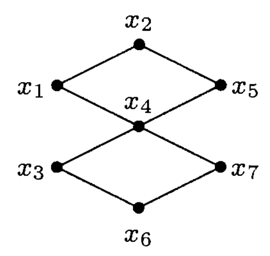
(Incidentally, it is by no means obvious on algebraic grounds that these two sums of products of binomial coefficients should come out to be equal.) We also have the formulas

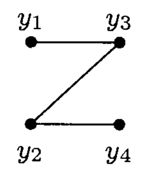
$$\left(\begin{matrix}p\\m\end{matrix} < \begin{matrix}q\\n\end{matrix}\right) + \left(\begin{matrix}q\\n\end{matrix} < \begin{matrix}p\\m\end{matrix}\right) = \binom{m+n}{m}; \qquad (30)$$

$$\left(\begin{matrix}q\\n\end{matrix} < \begin{matrix}p\\m\end{matrix}\right) = \left(\begin{matrix}m+1-p\\m\end{matrix} < \begin{matrix}n+1-q\\n\end{matrix}\right); \qquad (31)$$

$$\left(\begin{matrix}p\\m\end{matrix} < \begin{matrix}q\\n\end{matrix}\right) = \left(\begin{matrix}p\\m-1\end{matrix} < \begin{matrix}q\\n\end{matrix}\right) + \left(\begin{matrix}p\\m\end{matrix} < \begin{matrix}q\\n-1\end{matrix}\right) + [p\le m][q=n]\binom{m+n-1}{m}. \qquad (32)$$

For definiteness, let us now consider the two graphs

$G' =$ 

, $G'' =$ 

. (33)

It is not hard to show by direct enumeration that $T(G') = 42$ and $T(G'') = 5$; so if G is the 11-vertex graph having G' and G'' as components, we have $T(G) = \binom{11}{4} \cdot 42 \cdot 5 = 69300$ by Eq. (25). This is a formidable number of permutations to list, if we want to know how many of them have $x_i < y_j$ for each i and j. But the calculation can be done by hand, in less than an hour, as follows. We form the matrices $A(G')$ and $A(G'')$, where A_{ik} is the number of consistent permutations of G' (or G'') in which x_i (or y_i) is equal to k. Thus the number of permutations of G in which x_i is less than y_j is the (i,p) element of $A(G')$ times $\left(\begin{smallmatrix}p\\7\end{smallmatrix} < \begin{smallmatrix}q\\4\end{smallmatrix}\right)$ times the (j,q) element of $A(G'')$, summed over $1 \le p \le 7$ and $1 \le q \le 4$. In other words, we want to form the matrix product $A(G') \cdot L \cdot A(G'')^T$, where $L_{pq} = \left(\begin{smallmatrix}p\\7\end{smallmatrix} < \begin{smallmatrix}q\\4\end{smallmatrix}\right)$. This comes to

$$\begin{pmatrix} 21 & 16 & 5 & 0 & 0 & 0 & 0\\ 0 & 5 & 10 & 12 & 10 & 5 & 0\\ 21 & 16 & 5 & 0 & 0 & 0 & 0\\ 0 & 0 & 12 & 18 & 12 & 0 & 0\\ 0 & 0 & 0 & 0 & 5 & 16 & 21\\ 0 & 5 & 10 & 12 & 10 & 5 & 0\\ 0 & 0 & 0 & 0 & 5 & 16 & 21 \end{pmatrix} \begin{pmatrix} 210 & 294 & 322 & 329\\ 126 & 238 & 301 & 325\\ 70 & 175 & 265 & 315\\ 35 & 115 & 215 & 295\\ 15 & 65 & 155 & 260\\ 5 & 29 & 92 & 204\\ 1 & 8 & 36 & 120 \end{pmatrix} \begin{pmatrix} 2 & 3 & 0 & 0\\ 2 & 2 & 0 & 1\\ 1 & 0 & 2 & 2\\ 0 & 0 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 48169 & 42042 & 66858 & 64031\\ 22825 & 16005 & 53295 & 46475\\ 48169 & 42042 & 66858 & 64031\\ 22110 & 14850 & 54450 & 47190\\ 5269 & 2442 & 27258 & 21131\\ 22825 & 16005 & 53295 & 46475\\ 5269 & 2442 & 27258 & 21131 \end{pmatrix}.$$

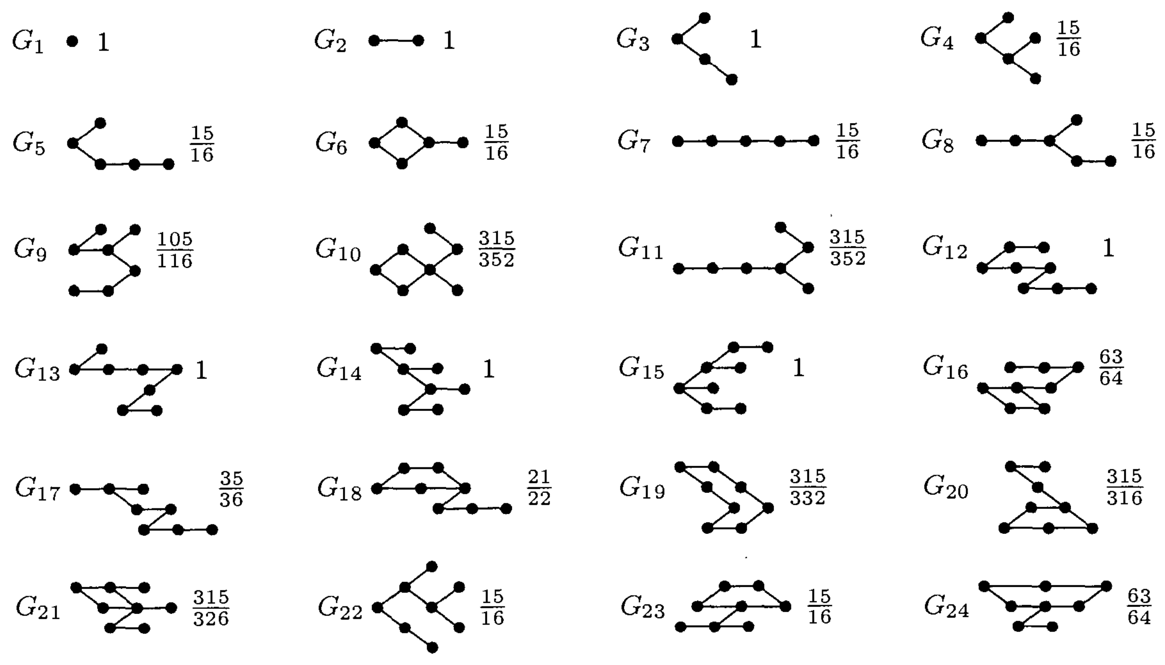


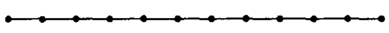
Fig. 36. Some graphs and their efficiencies, obtained at the beginning of a long proof that $S(12) > 29$.

Thus the "best" way to hook up G' and G'' is to compare x_1 with y_2; this gives 42042 cases with $x_1 < y_2$ and $69300 - 42042 = 27258$ cases with $x_1 > y_2$. (By symmetry, we could also compare x_3 with y_2, x_5 with y_3, or x_7 with y_3, leading to essentially the same results.) The efficiency of the resulting graph for $x_1 < y_2$ is

$$\frac{69300}{84084} E(G')E(G''),$$

which is none too good; hence it is probably a bad idea to hook G' up with G'' in any sorting method! The point of this example is that we are able to make such a decision without excessive calculation.

These ideas can be used to provide independent confirmation of Mark Wells's proof that $S(12) = 30$. Starting with a graph containing one vertex, we can repeatedly try to add a comparison to one of our graphs G or to $G' \oplus G''$ (a pair of graph components G' and G'') in such a way that the two resulting graphs have 12 or fewer vertices and efficiency $\geq 12!/2^{29} \approx 0.89221$. Whenever this is possible, we take the resulting graph of least efficiency and add it to our set, unless one of the two graphs is isomorphic to a graph we already have included. If both of the resulting graphs have the same efficiency, we arbitrarily choose one of them. A graph can be identified with its dual (obtained by reversing the order), so long as we consider adding comparisons to $G' \oplus \text{dual}(G'')$ as well as to $G' \oplus G''$. A few of the smallest graphs obtained in this way are displayed in Fig. 36 together with their efficiencies.

Exactly 1649 graphs were generated, by computer, before this process terminated. Since the graph  was not obtained, we may conclude that $S(12) > 29$. It is plausible that a similar experiment could be performed to deduce that $S(22) > 70$ in a fairly reasonable amount of time, since

$22!/2^{70} \approx 0.952$ requires extremely high efficiency to sort in 70 steps. (Only 91 of the 1649 graphs found on 12 or fewer vertices had such high efficiency.)

The intermediate results suggest strongly that $S(13) = 33$, so that merge insertion would not be optimum when $n = 13$. It should certainly be possible to prove that $S(16) < F(16)$, since $F(16)$ takes no fewer comparisons than sorting ten elements with $S(10)$ comparisons and then inserting six others by binary insertion, one at a time. There must be a way to improve upon this! But at present, the smallest case where $F(n)$ is definitely known to be nonoptimum is $n = 47$: After sorting 5 and 42 elements with $F(5) + F(42) = 178$ comparisons, we can merge the results with 22 further comparisons, using a method due to J. Schulte Mönting, *Theoretical Comp. Sci.* **14** (1981), 19–37; this beats $F(47) = 201$. (Glenn K. Manacher [*JACM* **26** (1979), 441–456] had previously proved that infinitely many n exist with $S(n) < F(n)$, starting with $n = 189$.)

The average number of comparisons. So far we have been considering procedures that are best possible in the sense that their worst case isn't bad; in other words, we have looked for "minimax" procedures that minimize the *maximum* number of comparisons. Now let us look for a "minimean" procedure that minimizes the *average* number of comparisons, assuming that the input is random so that each permutation is equally likely.

Consider once again the tree representation of a sorting procedure, as shown in Fig. 34. The average number of comparisons in that tree is

$$\frac{2+3+3+3+3+2}{6} = 2\tfrac{2}{3},$$

averaging over all permutations. In general, the average number of comparisons in a sorting method is the *external path length* of the tree divided by $n!$. (Recall that the external path length is the sum of the distances from the root to each of the external nodes; see Section 2.3.4.5.) It is easy to see from the considerations of Section 2.3.4.5 that the minimum external path length occurs in a binary tree with N external nodes if there are $2^q - N$ external nodes at level $q-1$ and $2N - 2^q$ at level q, where $q = \lceil \lg N \rceil$. (The root is at level zero.) The minimum external path length is therefore

$$(q-1)(2^q - N) + q(2N - 2^q) = (q+1)N - 2^q. \tag{34}$$

The minimum path length can also be characterized in another interesting way: *An extended binary tree has minimum external path length for a given number of external nodes if and only if there is a number l such that all external nodes appear on levels l and $l+1$.* (See exercise 20.)

If we set $q = \lg N + \theta$, where $0 \le \theta < 1$, the formula for minimum external path length becomes

$$N(\lg N + 1 + \theta - 2^\theta). \tag{35}$$

The function $1 + \theta - 2^\theta$ is shown in Fig. 37; for $0 < \theta < 1$ it is positive but very small, never exceeding

$$1 - (1 + \ln\ln 2)/\ln 2 = 0.08607\ 13320\ 55934+. \tag{36}$$

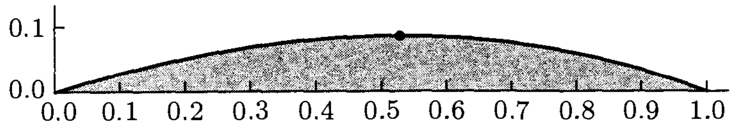


Fig. 37. The function $1+\theta-2^\theta$.

Thus the minimum possible average number of comparisons, obtained by dividing (35) by N, is never less than $\lg N$ and never more than $\lg N+0.0861$. (This result was first obtained by A. Gleason in 1956.)

Now if we set $N = n!$, we get a lower bound for the average number of comparisons in any sorting scheme. Asymptotically speaking, this lower bound is

$$\lg n! + O(1) = n\lg n - n/\ln 2 + O(\log n). \qquad (37)$$

Let $\bar{F}(n)$ be the average number of comparisons performed by the merge insertion algorithm; we have

$n =$	1	2	3	4	5	6	7	8
lower bound (34) $=$	0	2	16	112	832	6896	62368	619904
$n!\,\bar{F}(n) =$	0	2	16	112	832	6912	62784	623232

Thus merge insertion is optimum in both senses for $n \le 5$, but for $n = 6$ it averages $6912/720 = 9.6$ comparisons while our lower bound says that an average of $6896/720 = 9.577777\ldots$ comparisons might be possible. A moment's reflection shows why this is true: Some "fortunate" permutations of six elements are sorted by merge insertion after only eight comparisons, so the comparison tree has external nodes appearing on three levels instead of two. This forces the overall path length to be higher. Exercise 24 shows that it is possible to construct a six-element sorting procedure that requires nine or ten comparisons in each case; it follows that this method is superior to merge insertion, on the average, and no worse than merge insertion in its worst case.

When $n = 7$, Y. Césari [Thesis (Univ. of Paris, 1968), page 37] has shown that no sorting method can attain the lower bound 62368 on external path length. (It is possible to prove this fact without a computer, using the results of exercise 22.) On the other hand, he has constructed procedures that do achieve the lower bound (34) when $n = 9$ or 10. In general, the problem of minimizing the average number of comparisons turns out to be substantially more difficult than the problem of determining $S(n)$. It may even be true that, for some n, all methods that minimize the *average* number of comparisons require *more* than $S(n)$ comparisons in their worst case.

EXERCISES

1. [*20*] Draw the comparison trees for sorting four elements using the method of (a) binary insertion; (b) straight two-way merging. What are the external path lengths of these trees?

2. [*M24*] Prove that $B(n) \le L(n)$, and find all n for which equality holds.