Martin Gardner (New York: Chas. Scribner's Sons, 1968), Puzzles 190 and 194; Ross Eckler, *Making the Alphabet Dance* (N.Y.: St. Martin's Griffin, 1997), Fig. 46c.]

The first and last sets of three or more five-letter English anagrams are {ALBAS, BALAS, BALSA, BASAL} and {STRUT, STURT, TRUST}, if proper names are not allowed. However, the proper names Alban, Balan, Laban, and Nabal lead to an earlier set {ALBAN, BALAN, BANAL, LABAN, NABAL, NABLA} if that restriction is dropped. The most striking example of longer anagram words in common English is perhaps the amazingly mathematical set {ALERTING, ALTERING, INTEGRAL, RELATING, TRIANGLE}.

A faster way to proceed is to compute $f(\alpha) = (h(a_1)+h(a_2)+\cdots+h(a_5)) \bmod m$, where $a_1, \ldots, a_5$ are numerical codes for the individual letters in α, and $(h(1), h(2), \ldots)$ are 26 randomly selected constants; here m is the computer word size. Sorting the file $(f(\alpha), \alpha)$ will bring anagrams together; afterwards when $f(\alpha) = f(\beta)$ we must make sure that we have a true anagram with $\alpha' = \beta'$. The value $f(\alpha)$ can be calculated more rapidly than α', and this method avoids the determination of α' for most of the words α in the file.

Note: A similar technique can be used when we want to bring together all sets of records that have equal multiword keys $(a_1, \ldots, a_n)$. Suppose that we don't care about the order of the file, except that records with equal keys are to be brought together; it is sometimes faster to sort on the one-word key $(a_1 x^{n-1} + a_2 x^{n-2} + \cdots + a_n) \bmod m$, where x is any fixed value, instead of sorting on the original multiword key.

22. Find isomorphic invariants of the graphs (functions that take equal values on isomorphic directed graphs) and sort on these invariants, to separate "obviously nonisomorphic" graphs from each other. Examples of isomorphic invariants: (a) Represent vertex v_i by (a_i, b_i), where a_i is its in-degree and b_i is its out-degree; then sort the pairs (a_i, b_i) into lexicographic order. The resulting file is an isomorphic invariant. (b) Represent an arc from v_i to v_j by (a_i, b_i, a_j, b_j), and sort these quadruples into lexicographic order. (c) Separate the directed graph into connected components (see Algorithm 2.3.3E), determine invariants of each component, and sort the components into order of their invariants in some way. See also the discussion in exercise 21.

After sorting the directed graphs on their invariants, it will still be necessary to make secondary tests to see whether directed graphs with identical invariants are in fact isomorphic. The invariants are helpful for these tests too. In the case of free trees it is possible to find "characteristic" or "canonical" invariants that completely characterize the tree, so that secondary testing is unnecessary [see J. Hopcroft and R. E. Tarjan, in *Complexity of Computer Computations* (New York: Plenum, 1972), 140–142.]

23. One way is to form a file containing all three-person cliques, then transform it into a file containing all four-person cliques, etc.; if there are no large cliques, this method will be quite satisfactory. (On the other hand, if there is a clique of size n, there are at least $\binom{n}{k}$ cliques of size k; so this method can blow up even when n is only 25 or so.)

Given a file that lists all $(k-1)$-person cliques, in the form $(a_1, \ldots, a_{k-1})$ where $a_1 < \cdots < a_{k-1}$, we can find the k-person cliques by (i) creating a new file containing the entries $(b, c, a_1, \ldots, a_{k-2})$ for each pair of $(k-1)$-person cliques of the respective forms $(a_1, \ldots, a_{k-2}, b)$, $(a_1, \ldots, a_{k-2}, c)$ with $b < c$; (ii) sorting this file on its first two components; (iii) for each entry $(b, c, a_1, \ldots, a_{k-2})$ in this new file that matches a pair (b, c) of acquaintances in the originally given file, output the k-person clique $(a_1, \ldots, a_{k-2}, b, c)$.

24. (Solution by Norman Hardy, c. 1967.) Make another copy of the input file; sort one copy on the first components and the other on the second. Passing over these

files in sequence now allows us to create a new file containing all pairs (x_i, x_{i+2}) for $1 \le i \le N-2$, and to identify (x_{N-1}, x_N). The pairs $(N-1, x_{N-1})$ and (N, x_N) should be written on still another file.

The process continues inductively. Assume that file F contains all pairs (x_i, x_{i+t}) for $1 \le i \le N-t$, in random order, and that file G contains all pairs (i, x_i) for $N-t < i \le N$ in order of the second components. Let H be a copy of file F, and sort H by first components, F by second. Now go through F, G, and H, creating two new files F' and G', as follows. If the current records of files F, G, H are, respectively (x, x'), (y, y'), (z, z'), then:

i) If $x' = z$, output (x, z') to F' and advance files F and H.

ii) If $x' = y'$, output $(y-t, x)$ to G' and advance files F and G.

iii) If $x' > y'$, advance file G.

iv) If $x' > z$, advance file H.

When file F is exhausted, sort G' by second components and merge G with it; then replace t by $2t$, F by F', G by G'.

Thus t takes the values $2, 4, 8, \ldots$; and for fixed t we do $O(\log N)$ passes over the data to sort it. Hence the total number of passes is $O((\log N)^2)$. Eventually $t \ge N$, so F is empty; then we simply sort G on its *first* components.

25. (An idea due to D. Shanks.) Prepare two files, one containing $a^{mn} \bmod p$ and the other containing $ba^{-n} \bmod p$ for $0 \le n < m$. Sort these files and find a common entry.

Note: This reduces the worst-case running time from $\Theta(p)$ to $\Theta(\sqrt{p}\log p)$. Significant further improvements are often possible; for example, we can easily determine if n is even or odd, in $\log p$ steps, by testing whether $b^{(p-1)/2} \bmod p = 1$ or $(p-1)$. In general if f is any divisor of $p-1$ and d is any divisor of $\gcd(f, n)$, we can similarly determine $(n/d) \bmod f$ by looking up the value of $b^{(p-1)/f}$ in a table of length f/d. If $p-1$ has the prime factors $q_1 \le q_2 \le \cdots \le q_t$ and if q_t is small, we can therefore compute n rapidly by finding the digits from right to left in its mixed-radix representation, for radices $q_1, \ldots, q_t$. (This idea is due to R. L. Silver, 1964; see also S. C. Pohlig and M. Hellman, *IEEE Transactions* **IT-24** (1978), 106–110.)

John M. Pollard discovered an elegant way to compute discrete logs with about $O(\sqrt{p})$ operations mod p, requiring very little memory, based on the theory of random mappings. See *Math. Comp.* **32** (1978), 918–924, where he also suggests another method based on numbers $n_j = r^j \bmod p$ that have only small prime factors.

Asymptotically faster methods are discussed in exercise 4.5.4–46.

SECTION 5.1.1

1. 205223000; 27354186.

2. $b_1 = (m-1) \bmod n$; $b_{j+1} = (b_j + m - 1) \bmod (n-j)$.

3. $\overline{a}_j = a_{n+1-j}$ (the "reflected" permutation). This idea was used by O. Terquem [*Journ. de Math.* **3** (1838), 559–560] to prove that the average number of inversions in a random permutation is $\frac{1}{2}\binom{n}{2}$.

4. C1. Set $x_0 \leftarrow 0$. (It is possible to let x_j share memory with b_j in what follows, for $1 \le j \le n$.)

C2. For $k = n, n-1, \ldots, 1$ (in this order) do the following: Set $j \leftarrow 0$; then set $j \leftarrow x_j$ exactly b_k times; then set $x_k \leftarrow x_j$ and $x_j \leftarrow k$.

C3. Set $j \leftarrow 0$.