This bibliography attempts to help the reader select from the rich body of sorting literature that which is in accord with his interests, needs, and prior training.

Historical trends within the field are briefly outlined, and subspecialties are identified. Critical comments and classification of the cited works are intended to help the reader to avoid wasted effort.

A guided bibliography to sorting

by H. Lorin

In this paper, we restrict the word sorting to mean the ordering of data by a digital computer. Given a collection of data entries and an ordering key, various processes can be invoked to arrange the entries into a desired order. The order is commonly an ascending or descending numerical or alphabetic sequence, but other orders are possible. At one time, the meaning of the word sorting was limited to the process of pigeon-hole classification, whereas the words "re-arranging" or "ordering" carried the broader meaning. More recently, however, sorting has come to include all techniques for the ordered arrangement of data.

A large body of literature about sorting has developed as the result of continuous and intensive work in the area since the invention of the general-purpose digital computer. At different times during the history of sorting, workers in the field were preoccupied with different problems. In the late 1950's, concern was with improved techniques using tape drives; in the early 1960's, with efficient methods using minimum storage space; in the mid 1960's, with disk-oriented methods; and currently the industry is becoming concerned with sorting on parallel processors and in virtual memory environments. Many of the techniques currently discussed in the literature go back to the very beginnings of the art; others are truly new.

Categories of sorting activity

directions in sorting effort

Work in sorting is progressing along several lines. Some effort is aimed at developing greater insight into known techniques and at discovering more details about their behavior in different situations.

A second line is the development of improved techniques. For example, the search for algorithms combining efficient use of storage space with a small number of comparisons has resulted in significantly different techniques from those that appear as "standard" in the early literature.

Other activity is concerned not so much with the fundamental techniques of achieving order but with the environment in which an ordering process occurs. Investigations of new kinds of devices, new data-handling techniques for new devices, new processor or channel architectures, etc., are constantly underway.

The field of sorting can be roughly categorized into a number of subspecialties:

internal sorting

Internal sorting is the process of ordering a list of elements residing in primary storage. The list may represent all of the data to be ordered, or it may be a portion of a larger list all of which cannot fit into primary storage at one time. There are two types of internal ordering algorithms: the *comparative* and the *distributive*. The comparative algorithms order the list by making a series of comparisons of the relative magnitude of the ordering keys of the elements. The distributive algorithms order the list by testing a key or a digit of a key against a standard and collecting all members of a group together. Group definitions are then modified so that all elements and groups are ordered during a last pass. The performance of comparative algorithms varies with the number of elements to be sorted and the permutation of the elements. The performance of distributive algorithm varies with the range of the keys and their distribution.

The criteria for measuring the performance of an ordering algorithm are: the number of comparisons that must be performed before the list is ordered, the number of movements of data on the list before the list is ordered, the amount of space required beyond that needed to hold the list, and the sensitivity to certain kinds of order of the data. The number of comparisons among algorithms varies considerably. The best of the minimum storage comparative algorithms achieve order with roughly $N \log_2 N$ comparisons, the worst with roughly N^2, where N is the number of elements to be sorted. A minimum storage algorithm is one that requires little or no additional storage to perform the ordering. The selection of a sorting method and achieving its most

efficient representation in a computer program is a complex process. In addition to the machine characteristics and the characteristics of the algorithms, the characteristics of the data to be sorted must be well understood. Size of the records, size of the key, physical placement of the key in records, and the distribution and permutation of key values all affect the selection and performance of a sort and the particular variation of a general algorithm that the user represents in code.

external sorting

External sorting is the process of ordering data lists that are too large to be represented in primary storage. The ordering process consists of cyclically reading portions of the data into primary storage and then distributing the elements to devices by some algorithm. There are two major external sorting methods. The *comparative method* is called the sort/merge and is the most commonly used. The *distributive* method places elements across output devices according to the values of individual digits assigned to each output device. (A number of values may be assigned to a device to reduce the number of required devices.) From pass to pass, different digit portions are scanned (left to right or right to left) until the data is ordered. It is common for there to be internal sorting of groups that fit into primary storage during the last pass. A variation of the technique substitutes ranges of key values for digit position values.

The sort/merge is a two-step process. For the first step, sorting, some internal sorting method is used to produce long strings (runs of ascending values) and to disperse these strings to output devices. Strings are produced by ordering parts of the data in primary storage. Sublists are read sequentially and ordered into strings. For the second phase, the strings are read and combined (merged) into larger strings, so that the number of strings is reduced at each pass. The process ends when the number of strings is reduced to one long ordered string.

There are a variety of merge types and string dispersion algorithms for tape and random-access storage devices. The design of a merge depends upon I/O subsystem characteristics, device type, size of primary storage, data characteristics, and many other factors. The design goal is the balance of as few merge passes as possible with having each pass as fast as possible. Attention must be paid to proper buffering and blocking, as well as to string dispersion. Tape merges differ from each other in the specific distribution of strings across tape devices. Disk and drum merge designs differ from each other in the specific assignment of addresses to strings as they are produced.

sort systems

Sort systems design includes a number of considerations. A sort system attempts to provide users with an efficient generalized sorting capability. The system must be capable of sorting a wide

range of data types, and the sort must be usable over a wide range of system configurations. The design of a sort system involves three major decision areas and a number of associated secondary determinations. The major decisions are: the type of sorting techniques that will be made available; the techniques that will be used to generate a specific sort for specific data and configuration; and the user language (the information the user is asked to provide and the extent of his options).

There are two major approaches to designing a package. One is to develop the package as a closed "parameterizable black box." In such a system, the user has parameter statements of a form unique to the sort system and some fixed entry points if he wishes to insert some code into the running form of the sort provided by the package. An alternative approach is to view the sort package as a family of routines called from the user program using the language CALL or macro facilities. In such a system, the user has considerably more flexibility in what he may do in the sort environment. The nature of the user sort "language" is naturally reflected in the differing approaches. The sort package for the IBM 709 is a classic form of the "parameterized" black box; the sort package for the UNIVAC III (SODA) is a classic form of the callable macro.

Sort package design requires many other decisions, such as the point and methods of calling the sort program. Other determinations include the interfaces between the sort program and higher-level languages of an operating system, system I/O support (and whether to use it or specially developed I/O routines), and operating system primary and auxiliary storage allocation mechanisms.

Developers of sort systems must be thoroughly familiar with the characteristics of the hardware of the sorting machine, they must be knowledgeable in program generation techniques, competent in program optimization and balancing methods, familiar with design technology and the operating system and I/O areas, as well as being competent sort specialists. In addition, since the performance of a sort is critical, a sort development team must be competent to undertake extensive prediction, analysis, and test of their product's performance characteristics.

related areas

Several areas in data processing are closely related to sorting:

Searching is the process of efficiently locating a particular element in a data set. Search techniques attempt to minimize the number of comparisons required before an element is found. Various data structures and their statistical properties are considered in terms of their effect on the length of search. The patterns of searching are useful for investigating comparison se-

quences in ordering algorithms, since the number of comparisons required to achieve order is a critical parameter of a sorting technique.

File organization includes the structure of data files, the methods other than physical contiguity that can be used to represent an ordering of a file, and the variability and complexity of records and record groups in a file. For example, the use of chaining techniques to represent a desired order can relieve a sort program of the need to move large amounts of data. The use of key-transformation hashing techniques as a means of re-ordering elements by developing addresses that represent their order is of interest. Similarly, additional problems exist in sort situations in which it is necessary to order records of variable size.

Hardware characteristics and organizations affect sort algorithms. Workers in the field have been fascinated in the past with the development of architectures that would be ideal and practical for sorting. Multiway comparison instructions are an example of what sorting people have been asking for and not getting. However, they have gotten various forms of indirect addressing, search instructions, associative memories, intelligent channels, caches, and a host of auxiliary storage devices. It is essential that sort development activities take cognizance of the subtleties of hardware and hardware performance.

Surely as important as CPU characteristics and I/O subsystem path capabilities is the nature of the devices that will hold data during the sorting process. The tape unit is the simplest device to support, because of its sequential, noncyclic, positional addressability. The read backward capability, rewind characteristics, and start times have some effect on sort design, but the major factor in tape sorting is a proper distribution across different tapes. Disks, drums, data cells, and other random- or mass-access devices present a different set of problems, because of their latency times and the great variation in performance due to specific positioning of data. Techniques to minimize seek and search time dominate sort design for these devices.

Important related considerations for sorting are the accessibility, and the organizational and path interface characteristics of new storage devices. In addition, there has been a recent rebirth of interest in dedicated sorting machines with hardware-implemented sorting algorithms.

Sorting literature

prerequisite knowledge

Those who wish only to find and implement a reasonable sort need no associated specialized knowledge. Working descriptions

of sorting methods with usable guides to relative performance exist in the extensive literature of the field. Many of the articles tend to be oriented toward statisticians or mathematicians, but there exists sufficient narrative material so that a programmer or analyst without this background can familiarize himself with techniques and alternatives.

A reading knowledge of ALGOL is important. One of the quickest ways to become familiar with a number of sorting techniques is to read the algorithms published in the *Communications of the ACM*. In addition to these, some authors have chosen ALGOL as a language to demonstrate an algorithm in their articles. Some caution must be exercised here because many of the algorithms are carelessly prepared or erroneously printed. Some are in a nonexecutable "pseudo-ALGOL" and do not represent a workable form.

The ALGOL algorithms noted in this bibliography are at least procedurally correct from a sorting point of view. They have been coded into workable PL/I equivalents and extensively tested.

Those who desire to specialize in the development or analysis of sort algorithms or to be very careful in their choice of a sort procedure must have a statistical and mathematical background. An appreciation of the derivation, applicability, and generality of formulas used to project performance requires concepts of permutation, distribution, randomness, nonparameteric tests for randomness, autocorrelation, etc. Developing performance analysis methods for new techniques or new combinations of techniques requires facility in algebra and calculus. Algebra is often used to describe sorting processes. Bounds or limits on performance are often expressed in the calculus. A very thorough understanding of sorting is based on a usable knowledge of these disciplines.

But the development of sorting programs is an activity far more extensive than the development of sorting algorithms. The worker with little mathematics or statistics can make important contributions to the field once he has understood an algorithm theoretically developed.

The following guide through the rich literature of sorting attempts to enable the reader to develop the degree of knowledge he desires with a minimum number of false starts and duplicated efforts, and without reading at the wrong mathematical level.

BIBLIOGRAPHY

1. *Sorting Techniques* C20-1639, International Business Machines, Data Processing Division, White Plains, New York (1965). A nice overview of internal and external sorting, with some general performance characteristics **introductory articles**

given. The manual, although excellent in general, contains some errors carried over from its sources. It is also a little dated and does not include recently developed techniques. Use it only as an introductory guide.

2. C. C. Gotlieb, "Sorting on computers," *Communications of the ACM* **6,** No. 5, 184–201 (May 1963). A very compact introduction to the field. An alternative to Reference 1 for the very busy reader.
3. E. H. Friend, "Sorting on electronic computer systems," *Journal of the Association for Computing Machinery* **3,** No. 3, 134–168 (July 1956). A true jewel and better every time it is read, despite its age. No better introduction to considerations in designing an external sort exists. The appendixes are worthwhile. Little mathematics is required except in the appendixes.
4. P. F. Windley, "Trees, forests, and rearranging," *British Computer Journal* **3,** No. 2, 84–88 (1960). This article introduces the concept of binary tree structure and its use in ordering data. The style is narrative except for the derivation of expected number of compares and standard deviation. The reader with limited mathematical background will come away with a good idea of the nature of trees. The reader with more mathematics will see a rare instance of the published development of deviation of number of compares in the general literature. The reader of this article might also profit from the following paper.
5. W. Burge, "Sorting, trees, and measures of order," *Information and Control* **1,** No. 3, 181–197 (September 1958).

internal sorting

The following collection of references is intended to extend the introductory material to acquaint the reader with some of the more widely known algorithms and to solidify his appreciation for the area. These articles may be read in any order. Together with the introductory articles, this material should bring the reader to an intermediate point where he understands and is familiar with the underlying concepts, problems, and procedures of internal sorting.

6. J. Boothroyd, "Stringsort, Algorithm 207," *Communications of the ACM* **6,** No. 10, 165 (October 1963). A clever merge that is worthwhile to understand as an example of a merge not dependent upon a premerge sorting process.
7. R. W. Floyd, "Treesort 3, Algorithm 243," *Communications of the ACM* **7,** No. 12, 701 (December 1964). This algorithm is the last in a series of algorithms by the author that impose a tree structure on a list of items to be sorted. It is useful as an example of a technique closely associated with the "replacement selection" so widely used in sorting packages.
8. M. H. Hall, "Method of comparing time requirements of sorting methods," *Communications of the ACM* **6,** No. 5, 259–263 (May 1963). Techniques for very fine calibration of sort performance. Nothing really new here, but it is worthwhile to tread through such an article. The formulas quoted are not reliable. The reader begins to develop a feeling for what is involved in comparing sorts.
9. C. A. R. Hoare, "QUICKSORT," *British Computer Journal* **5,** No. 1, 10–15 (1962). The introductory article for the technique discusses all the later refinements. A gem for all readers.
10. T. N. Hibbard, "Empirical study of minimum storage sorting," *Communications of the ACM* **6,** No. 5, 206–213 (May 1963). A presentation of radix exchange, a nonrecursively encoded form of QUICKSORT, a modified SHELLSORT, and a combined technique derived from SHELLSORT and merging. (Unhappily, "D" in ALGOL contains errors.) Methods are analyzed and compared in different data situations of random and nonrandom distribution and order. The nonmathematical reader will miss some details, but the algorithms and conclusions are worthwhile.
11. C. A. R. Hoare, "QUICKSORT, PARTITION, FIND, Algorithms 63, 64, 65," *Communications of the ACM* **4,** No. 7, 321–322 (July 1961). The first ALGOL presentation of the recursive partitioning algorithm. The algorithm in Hibbard is essentially the same but does not formally recurse.

A review of this form is useful to see the underlying concepts of the technique as originally presented. Either Hibbard or Hoare should be studied intensively.

12. D. L. Shell, "A highspeed sorting procedure," *Communications of the ACM* **2,** No. 7, 30–32 (July 1959). The famous SHELLSORT initial presentation. Easy narrative style, with example.
13. E. J. Isaac and R. C. Singleton, "Sorting by address calculation," *Journal of the Association for Computing Machinery* **3,** No. 3, 169–174 (1956). A narrative introduction of the method. Tradeoffs in space and time are discussed. Experimental results are obsolete.
14. W. Fuerzeig, "Mathsort, Algorithm 23," *Communications of the ACM* **3,** No. 11, 601 (November 1960). An encoding of a digit sort using frequency counts to reduce space requirements.

advanced internal sorting

The following articles can deepen the reader's familiarity with the area of internal sorting.

15. R. C. Bose and R. J. Nelson, "A sorting problem," *Journal of the Association for Computing Machinery* **9,** No. 2, 282–296 (April 1962). Presents an algorithm for developing a series of comparisons on a list to be sorted such that the sequence of compares is unchanged regardless of compare results. This sequence must be shown to be shortest (that is, no other fixed sequence can be shorter) and to successfully achieve order. The authors show both weight (number of compares) and ordering property. After this, the reader should read the following paper.
16. T. N. Hibbard, "A simple sorting algorithm," *Journal of the Association for Computing Machinery* **10,** No. 2, 142–150 (April 1963). In this article, a method of computer generation of Bose-Nelson sequence is incorporated into an ALGOL sorting algorithm. This and the preceding article have rather formidable notation.
17. T. N. Hibbard, "Some combinatorial properties of certain trees with application to searching and sorting," *Journal of the Association for Computing Machinery* **9,** No. 1, 13–28 (January 1962). A formal discussion of tree structure and the presentation of a QUICKSORT variant in this context. The QUICKSORT variant is identical to that given in the Hibbard article of 1963 listed above.
18. B. S. Brawn, F. G. Gustavson, and E. S. Mankin, *Sorting Performance in a Paged Virtual Memory*, IBM Thomas J. Watson Research Report RC2435, Yorktown Heights, New York (April 1969). A study of five variations of sorting on a paging machine. Each method is a variant of QUICKSORT with or without merging. The authors also comment on other techniques (insertion, radix-exchange) and other design choices (key or record sort) operating in a paging environment.
19. R. B. Lazarus and R. M. Frank, "A high-speed sorting procedure," *Communications of the ACM* **3,** No. 1, 20–22 (January 1960). A suggested modification of Shell's technique to avoid the development of disjoint sorted strings on the last pass. Basically, one attempts to force distance between comparands to an odd number. Issue is taken up by Hibbard and reflected in the following paper.
20. J. Boothroyd, "SHELLSORT Algorithm 201," *Communications of the ACM* **6,** No. 8, 445 (August 1963).
21. R. S. Scowen, "QUICKSORT, Algorithm 271," *Communications of the ACM* **8,** No. 11, 669–670 (November 1965). An algorithm, elegantly and carefully presented, that is very like Hibbard's version of QUICKSORT.
22. R. C. Singleton, "An efficient algorithm for sorting with minimal storage, Algorithm 347," *Communications of the ACM* **12,** No. 3, 185 (March 1969). A further honing of the QUICKSORT approach. The algorithm provides for median sampling and an alternative for short partitions.
23. G. S. Shedler, *An Example of Indeterminacy in a Parallel Algorithm*, IBM Thomas J. Watson Research Report RC2084, Yorktown Heights, New York (May 8, 1968). Also,

24. G. S. Shedler, *A Parallel Method for Sorting*, RC1823, (March 18, 1967) and *Illustrations of Decomposition in Parallel Algorithms*, RC2047 (April 3, 1968) IBM Thomas J. Watson Research Reports, Yorktown Heights, New York. Investigation and presentation of a QUICKSORT-like sort on a parallel processor. The general problem of operating on parallel processors is the development of noninterfering subtasks such that, although more comparisons may be made in sum by all processors, so many of them are made in parallel that elapsed time is reduced.
25. M. H. Van Emden, "Increasing the efficiency of QUICKSORT," *Communications of the ACM* **13,** No. 9, 563–566 (September 1970). A rather formal article discussing a technique for reducing the comparisons in QUICKSORT to closer to $N \log_2 N$. The method is an example of the growing interest in "heuristic" sorting, which attempts to take maximum advantage of what is dynamically discovered about data during the process of sorting. An important if formidable article in this area follows.
26. W. D. Frazer and A. C. McKeller, "Samplesort: a sampling approach to minimal storage time sorting," *Journal of the Association for Computing Machinery* **17,** No. 3, 496–507 (July 1970). See also the following.
27. M. H. Van Emden, "Increasing the efficiency of QUICKSORT, Algorithm 402," *Communications of the ACM* 13, No. 11, 693 (November 1970). This ALGOL algorithm is based upon the technique described in the other cited Van Emden article. The nonmathematical reader may observe the method from the algorithm. Comparison with another version of QUICKSORT is provided.
28. L. J. Woodrum, "Internal sorting with minimal comparing" *IBM Systems Journal* **8,** No. 3, 189–203 (1969). A nice discussion of the properties of comparative sorts, and the presentation (in APL) of a merge with minimum compare properties. A knowledge of APL and some statistics is required for a full appreciation of the article.
29. M. D. Maclaren, "Radix exchange plus sifting," *Journal of the Association of Computing Machinery* **13,** No. 3, 404–411 (July 1966). A narrative description of a combined method with a formalized discussion of its efficiency.

external sorting

This collection should provide an awareness of some details of the basic techniques on tape and drum/disk and of considerations in merge design.

30. B. K. Betz and W. C. Carter, "New merge sort techniques," *ACM National Proceedings*, p. 14 (1959). Introductory descriptions of cascade tape merges. Somewhat formal. But see also Reference 48.
31. R. L. Gilstad, "Polyphase merge sorting—an advanced technique," Proceedings Eastern Joint Computer Conference, 143–148 (1960). An introductory description of the polyphase merge.
32. N. A. Black, "Optimum merging from mass storage," *Communications of the ACM* **13,** No. 12, 745–749 (December 1970). A full discussion of the parameters involved in optimizing a mass storage merge. The article begins with a narrative of a simple "complete pass" merge scheme and an alternative to it. Notation is introduced but the nonmathematical reader who sticks to it will be rewarded with a real understanding of the complexities involved in merge design.
33. A. Bayes, "A generalized partial pass block sort," *Communications of the ACM* **11,** No. 7, 491–493 (July 1968). One of the rare recent discussions of a distribution-type sorting technique. Comparisons with known sorts of this type are provided and their general nature described.
34. W. H. Burge, *Analysis of Compromise Merge Sort Techniques*, IBM Thomas J. Watson Research Report RC2987, Yorktown Heights, New York (July 1970). This article describes the nature of the "k-1" merge family in terms of tree structures. "Compromise" techniques, which are between two standard k-1 merges, polyphase and cascade, are shown to be superior. For the sophisticated, mathematical reader. A very important article, since it is (perhaps) the first to investigate the "compromise" merge in detail.

35. B. J. Gassner, "Sorting by replacement selection," *Communications of the ACM* **10,** No. 2, 89–93 (February 1967). Formal discussion of the now standard method of producing long strings for a merge. Although this is an article about an internal method, its aim is to establish the length of strings that will be produced for a merge.
36. R. L. Gilstad, "Read backward polyphase sorting," *Communications of the ACM* **6,** No. 5, 220–223 (May 1963). A description of the distribution requirements for accomplishing backward read polyphase.
37. M. A. Goetz and G. S. Toth, "Comparison of oscillating and polyphase," *Communications of the ACM* **6,** No. 5, 223–225 (May 1963). The authors compare various claims and clarify some terms involved in comparing these techniques. An interesting example of a controversial article.
38. M. A. Goetz, "Some improvements in technology of string merging and internal sorting," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **26** (1964). Interaction between replacement-selection internal sort and polyphase merging.
39. M. A. Goetz "Organization and storage of data for efficient disk sorting," *Communications of the ACM* **6,** No. 5, 245–247 (May 1963).
40. G. Hubbard, "Some characteristics of sorting in computing systems using random access storage," *Communications of the ACM* **6,** No. 5 (May 1963). Introductory description of disk sorting in an IBM disk context.
41. D. Knuth, "Length of strings for a merge set," *Communications of the ACM* **6,** No. 11, 685–688 (November 1963). Despite his many comments, books, and articles, this author has not published frequently in the general periodical literature. This article discusses, rather formally, the length of strings that can be expected from replacement selection. The interesting result is the impact of reversing the direction of strings for a backward polyphase. The expected string size is shown to be 1.5 times the number of elements in the tournament.
42. W. D. Malcolm, "String distribution for polyphase merge," *Communications of the ACM* **6,** No. 5, 217–219 (May 1963). The "dummy" distribution technique now adopted as standard in most implementations of polyphase sort/merge programs.
43. A. G. Mendoza, "A dispersion pass algorithm for polyphase merge," *Communications of the ACM* **5,** No. 10, 502–504 (October 1962). This article undertakes a proof that a particular algorithm for adjusting to the proper distribution for a polyphase merge is efficient. The adjustment technique has been obsoleted by Reference 42. However, the distribution is of interest since it differs from that described in Reference 42. A combination of the ideas of both is a popular implementation. The general reader will avoid, with no loss, the proofs of efficiency.
44. S. Sobel, "Oscillating sort—a new sort merging technique," *Journal of the Association for Computing Machinery* **9,** No. 3, 372–374 (1962). First description of oscillating merge. A nice narrative, but be wary of the comparisons in Table 1. Even if accurate, these "power of merge" figures are a misleading index to how hard a merge works to order data.
45. C. E. Radke, "Merge-sort analysis by matrix techniques," *IBM Systems Journal* **5,** No. 4, 226–247 (1966). Various merges seen as operations on matrices. Conditions under which merges "converge" and do not "converge," where convergence is defined to be the reduction to one ordered string without a change in matrix operator.

sorting systems

This collection is intended to introduce basic considerations in organizing a sort package of some generality and contending with a sort system in an operating system environment. Advanced reading is not listed here since it will obviously be derived from the collection of user and internal manuals associated with vendor-provided sort programs.

46. J. B. Glore, "Sorting nonredundant files with FACT compiler," *Communications of the ACM* **6,** No. 5, 231–239 (May 1963). An interesting introduction to the interaction of language, its data facilities, and sorting.

47. J. B. Paterson, "COBOL sort verb," *Communications of the ACM* **6,** No. 5, 255–258 (May 1963). The capabilities and implications of the COBOL sort call are described.
48. R. Pratt, "UNIVAC III sort," ACM Sort Symposium, Princeton (November 1962). A nice general description of the techniques, organization, and structure of a vendor package. A narrative description of a cascade merge is included.
49. D. J. Waks, "Conversion, reconversion, and comparison techniques in variable-length sorting," *Communications of the ACM* **6,** No. 5, 267–271 (May 1963).

The bibliography is by no means exhaustive. The bibliographies of the articles listed here will lead the reader further. An extensive bibliography exists in the May, 1963 *Communications of the ACM* and in the *IBM Sorting Techniques* manual. The reader will have noticed a great proportion of the references come from ACM publications. This is intentional since the author feels these are the publications most accessible to general workers in our field. The large number of articles from the May 1963 issue of *Communications of ACM* is due to the fact that this issue published articles from the November 1962 ACM Sort Symposium in Princeton, New Jersey.