Z77-7257

TECHNICAL INFORMATION EXCHANGE

SYSTEM/360 SORTING TECHNIQUES

Mr. R. L. Shores
IBM Corporation
112 East Post Road
White Plains, New York

IBM Corporation, Technical Publications Dept., 112 E. Post Road, White Plains, N.Y. 10601

IBM®

October 3, 1967

This paper is a general introduction to the internal sorting and intermediate merge techniques implemented in IBM S/360 sort programs. Its main intent is to provide simplified, concise presentation of S/360 sort fundamentals. Additionally, the manual indicates comparative advantages of techniques in the tape and in the disk sorts.

The techniques discussed pertain to the BPS, BOS, TOS, DOS and OS sort programs for S/360 models 30, 40, 50, 65 and 75, and for the Model 20 tape system.

For IBM Internal Use Only

Z77-7257

CONTENTS

Generalized Sort Programs 1

System/360 Sort Technique Table 4

Sort Phase - Internal Sorting Techniques 5
- Straight Two-Way Merge 6
- Binary Insertion 7
- Replacement Selection 10
- Quadratic Selection 14

Intermediate Merge Phase - External Sorting Techniques 16
- Tape Sorting Methods 16
  - Balanced Merging 16
  - Polyphase Merge Sort 18
  - Oscillating Merge 22
- Direct Access Sorting Methods 24
  - Balanced Direct Access Technique 25
  - String Interleaving 26

Tag Mode vs. Record Mode 27

Comparison of Tape Sorting Techniques 29

General Considerations 33

S/360 Sort Publications 37

FIGURES

1. Phases of a generalized sort/merge program 38
2. Comparison of tape sorting techniques for T=4 and T= 10 47

TABLES

1. Initial string distribution for four-tape polyphase merge 43
2. Initial string distribution for five-tape polyphase merge 43
3. Initial string distribution for six-tape polyphase merge 43
4. Optimum total number of sequences for polyphase merging 43
5. Effective power of the merge (reduction factor) for tape sorting techniques 45
6. Number of data passes for tape sorting techniques 46

EXAMPLES

1. Straight two-way merge 38
2. Binary insertion 39
3. Replacement selection (G=4) 40
4. Replacement selection, possible organization for G=12 41
5. Replacement selection, possible organization for G=64 41
6. Linear selection 41
7. Quadratic selection 42
8. Read-forward polyphase merge with four tapes 43
9. Read-backward polyphase merge of 17 strings 44
10. Read-backward polyphase merge of 25 strings 44
11. Oscillating sort of 27 strings with four tapes 44
12. Balanced direct access sequence - distribution technique 45
13. Two-way merge on disks with string interleaving 45
14. Sorting time (in seconds) for record and tag sorts with an IBM 1301 disk file 45
15. Polyphase data pass determination 45

GENERALIZED SORT PROGRAMS

The objective of a generalized sort program is to permit sorting of a variety of files with reasonably high overall efficiency and without intervention by the user. A generalized sort accepts the introduction of file and system parameters, entered on control cards, and modifies the sort at execution time in accordance with those specific characteristics. It does not actually generate the object code to be executed, though it may assemble the sort program for a particular job from a series of relocatable subroutines.

A typical IBM generalized sorting or merging program accomplishes its objectives in approximately the following manner:

The user first specifies on control cards the relevant file, machine, and peripheral information that defines the sort. The file information, for example, must include a general description of the records to be processed, the control fields upon which the records are to be sorted, and the modifications to be made to the program. The range of specifications permissible is set by the sort designer.

The control cards bearing the parameter specifications are sometimes presented to a separate routine, known by such names as sort definition program, edit program, and analysis program, whose function is to select the subprograms needed for the type of sort or merge program required by the user. The selected subprograms are then combined into an absolute program through a linkage loader.

At object time, the selected sort/merge program adjusts itself to meet the requirements of the specific application, in accordance with the control card information supplied by the user.

The selected sort/merge program is divided into four phases:

1. The General Assignment Phase
2. The Sort Phase
3. The Intermediate Merge Phase
4. The Final Merge Phase

Where a complete sort is to be done, the program normally consists of all four phases, as shown in the flowchart (Figure 1). Where only a merge is to be performed, the general assignment and final merge phases are executed, while the sort and intermediate merge phases are omitted. A more detailed description of the four phases is presented in the following paragraphs.

GENERAL ASSIGNMENT PHASE

This phase is loaded into storage when the selected sort/merge program is called for execution, and it contains the initial entry point for the program itself. Unless control is assumed by an external monitor, part of the general assignment phase remains in storage throughout the execution of the program, to control the loading of each program phase.

The main functions of the general assignment phase are to perform the initial housekeeping for the program and to reserve and set up the common or phase-to-phase information area used by all phases of the sort or merge program. The assignment phase calls in the control cards and analyzes the control statements that describe the sort or merge to be performed. It checks the control information for validity and consistency. Based on these specifications, and on other information supplied by the system, the assignment phase sets up the common information area for all phases. This information is then used to select the most efficient technique (among those built in) for the specified sort and to calculate its basic parameters. Among specific tasks carried out by the general assignment phase, the following are typical:

1. Setting up of appropriate input-output device addresses
2. Establishing the initial storage addresses, in accordance with record length, blocking factors, and other parameters
3. Determining the size of G (the number of logical records accommodated in core storage during the sort phase), the maximum volume of records, the order of merge, etc., and setting up the appropriate counters, constants, and switches
4. Activating routines for interruption procedures, editing, controls, labels, etc.
5. Typing appropriate messages to the operators

In addition, if a separate sort definition program has not been set up, the general assignment phase also takes over the function of defining the other program phases by creating a list of the subroutines used for execution. It may call upon a linkage editor to link together the subprograms to be executed.

SORT PHASE

This phase performs the initial internal sorting of the records from the input file. It is divided into two parts: the assignment program and the running program. The assignment program performs the final initialization and modification of the running program. As part of the initialization, a record storage area is set up to contain the records being sorted. The running program performs the internal sort required to produce a set of ordered sequences, for eventual merging by later phases.

INTERMEDIATE MERGE PHASE

This phase performs one or more merging passes of the ordered sequences from the sort phase, until the number of sequences is fewer than or equal to the order of merge possible in the final merge phase. Control then passes to the final merge phase. As does the sort phase, this phase also contains a separate assignment portion, which initializes the running program.

FINAL MERGE PHASE

A final merge of the sort is performed during this phase, and the output data are arranged in the file format specified by the user. The assignment and running program portions are similar to those used in the intermediate merge phase. The assignment portion performs the final initialization and modifies the running program for either a sort or a merge.

SYSTEM/360 SORT TECHNIQUE TABLE

SYSTEM NUMBER	M20 360U-SM-150	BPS 360P-SM-043	BPS 360P-SM-044	BOS 360B-SM- 308	TOS 360M-SM-400	DOS 360N-SM-400	DOS 360N-SM-450	OS 360-SM-023
WHAT SORTED	Records	Records	Records	Record or Tag	Records	Records	Record or Tag	Records
I/O:		1 Selr Chan	1 or 2 Selr Chans	Mpx or Selr	Mpx or 1 or 2 Selr	Mpx or 1 or 2 Selr	Mpx or Selr	Selr or Mpx
INPUT DEVICE	Tape	Tape	Tape	Tape/ Disk	Tape	Tape	Tape/ Disk	Devices Supported by QSAM
INTERMEDIATE WORK STORAGE DEVICE	Tape- 3 to 6	Tape- 3 to 6	Tape- 3 to 6	Disk- 1 to 4	Tape- 3 to 7	Tape- 3 to 7	Disk- 1 to 6	Tape- 3 to 32, 2311, 2301
OUTPUT DEVICE	Tape	Tape	Tape	Tape/ Disk	Tape	Tape	Tape/ Disk	Devices Supported by QSAM
MAXIMUM CORE USED: 8K	X	X	X	X				
16K	X	X	X	X	X	X	X	
32K		X	X	X	X	X	X	X
64K		X	X	X	X	X	X	X
128K					X	X	X	X
256K						X	X	X
512K						X	X	X
1024K								X
TECHNIQUES: SORT PHASE-INTERNAL	Quadratic Selection	Binary Insertion	Binary Insertion	Straight Two way Merge	Binary Insertion	Binary Insertion	Straight Two way Merge	Replace-ment Selection
INTERMEDIATE and FINAL MERGE PHASE-EXTERNAL	Polyphase	Polyphase	Polyphase	Interleave	Polyphase	Polyphase	Interleave	Balanced Tape
								Polyphase Tape
								Oscillating Tape
								Balanced Direct Access

SORT PHASE - INTERNAL SORTING TECHNIQUES

Internal sorting is defined as the sequencing of a group of logical records contained in the internal high-speed (core) storage of the computer. It generally involves reading successive records from auxiliary storage into the record storage area (RSA) of core, sorting the group in storage by one of the methods to be described and then writing the sequenced group onto an external storage device.

In most sort applications, the file of records to be sorted is too large to be contained, at one time, within internal core storage. Thus, the internal sort passes serve only as a prelude to the subsequent external (intermediate) merge phase of the sort. The purpose of the internal sort, then, is to form a number of sequences, or strings which are placed on an external storage device and subsequently merged. The more efficient the internal sort technique, the longer the strings it generates, and hence, the fewer external merge passes required.

Since logical records can be many bytes in length, internal sorts do not usually manipulate the records themselves, but rather, the control words and/or address tags, thus conserving time and space. Generally, records can be sorted by (1) physically moving them about until they are in order; (2) forming tables of record addresses (tags) in storage, which are then sorted; or (3) combining the control word and the record key and sorting the resulting short key record. Either tag or key sorting is the preferred method and one or the other is assumed in the specific examples in this section.

STRAIGHT TWO-WAY MERGE

Merging is the process of combining several sequences of records to form a single specified sequence. The same rules by which sequences are combined may also be used to form sequences (of two or more items). Thus, the merging process has, essentially, a dual nature: it can be used for creating sequences (usually in an internal sort), and it is also capable of reducing previously created sequences to one (usually in an external sort). The versatility, speed, and simplicity of merging make it one of the most widely used sorting techniques.

The straight two-way merge with fixed-length strings is the simplest internal sort technique. The programming is straightforward, and the required number of comparisons is not affected by adverse sequences within the file. The group of records (G) to be merged is stored initially in two areas of core. The keys of pairs of records, one from each area, are then examined in turn and placed in sequence by exchanging the elements of a pair whenever necessary. The first merge pass combines single items (records, keys, or tags) to create strings of two items. At the end of this first pass the initial group of G items has been reduced to G/2 strings, each of length 2. In the second pass, pairs of these two-item strings are merged to produce G/4 strings, each of length 4. This is accomplished by successive comparisons of the keys in each of the strings (see example 1).

During the third pass, pairs of four-item strings are again merged in proper sequence to create G/8 strings, each of length 8. Each successive merge pass, therefore, cuts the number of strings in half, while doubling the length of each string until, finally, a single string of length G results. If the number of items, G, is equal to a power of two (2^n), a total of n passes is required to complete a two-way merge sort. Example 1 illustrates the principles of a standard two-way merge.

The first pass in example 1 compares single items, alternately drawn from storage areas A and B, and, by means of simple exchange, merges them into ascending sequences of two items each. At the end of this pass, eight strings of two items each are stored in core areas C and D. On the second pass, pairs of these two-item strings are merged into four strings of four items each. Consider the first pair of strings (13-69 and 02-56), stored in areas C and D, respectively. The following comparisons are required to arrive at the first output string (02, 13, 56, 69) in area A:

Comparisons	Output String (Area A)
1st: item 13 (area C) to item 02 (area D)	02
2nd: item 56 (area D) to item 13 (area C)	13
3rd: item 69 (area C) to item 56 (area D)	56
4th: copy item 69 (area C)	69

All comparisons are done in this manner. During the third pass, the four strings of four items are merged into two strings of eight items each, and on the final (fourth) pass a single ordered string of all 16 items results. Thus, for $G = 16 = 2^4$, a total of four 2-way merge passes were required. In general, any size G between 2^{n-1} and 2^n will require n merge passes. Equivalently, the number of passes for a two-way merge is the smallest integer that is equal to, or greater than, the log of G to base 2 (that is, $\log_2 G$).

It is evident, from example 1, that the number of comparisons during each pass equals the number of items to be sorted (G), so that the total number of comparisons (C) for a two-way merge becomes

$$C = G \times \text{No. of Passes} = G \log_2 G$$

Four storage areas (A, B, C, D), capable of holding eight items each, were required to sort the 16 items in the example. In general, for a file of G items, the required total storage is 2G items. However, this is true only if complete records are moved during the sort. The required storage is much less if record addresses or tags, alone, are moved about and sorted. In the latter case, the records are held during the sort in a record storage area with a capacity equal to G x Record Length, or G · L. The tags are assembled and ordered in two working areas, each with a capacity of G words or addresses. The required total storage (S) for a two-way tag sort, therefore, is

$$S = \underbrace{G \cdot L}_{\text{Records}} + \underbrace{2G \cdot \text{Address size}}_{\text{Tags}}$$

Summing up, the straight two-way merge is a rapid, efficient technique for sorting large files. It is relatively easy to program and is unaffected by adverse sequences, or even by reverse ordering.

BINARY INSERTION

A fairly effective method for sorting a small number of items, the insertion technique, places each item in sequence as soon as it is encountered. The records (or tags) are brought into the record storage area one at a time, the key of each is examined in turn, and the item is then inserted in the correct

place of an increasing file. Earlier members of the partial file are moved aside, when necessary, to make room for new items. The method is straightforward but is relatively slow, compared to other techniques.

Sorting by simple insertion has two inherent drawbacks: (1) excessive shifting of the sorted records is necessary for each new insertion; (2) the partial file must be searched each time to locate the correct place for inserting the new item.

The first drawback -- the large amount of record movement -- can be avoided by sorting record addresses (tags), rather than the records themselves.

The second limitation can be overcome, to some extent, by subdividing the area that must be searched to locate the correct position of each new item. A binary search may be used for this purpose.

The binary search technique eliminates from consideration, with each comparison, one-half of the remaining items in the partial file (hence, the name "binary search"). This greatly reduces the required number of comparisons, though at the expense of more extensive programming.

The method starts by an examination of the middle item of the partial file. If the key of the new item is smaller than that of the center item, it belongs in the upper half of the file, and the middle item of the top half (that is, the quarter point of the partial file) is examined next. If the key of the new item is larger than that of the center item, however, it belongs in the bottom half of the partial file, and the middle item of this lower half (that is, the three-quarter point of the partial file) is examined next. The examinations, thus, reveal whether the item belongs in the first, second, third, or fourth quarter of the partial file. The middle item of the selected quarter file is examined next to see whether the new item belongs in the top or in the bottom half of the quarter file, and so on. The division stops when there is but one item in the segment of the file under examination. The new item belongs next to this item and is inserted above or below it, depending upon whether it compares low or high, respectively. In the insertion itself old items are shifted up or down, as required.

Example 2 illustrates the technique of binary insertion, using the same sequence of 16 keys as in the previous example. After nine insertion steps (not shown) a partial file of nine items has been built up. The tenth item, 45, is now to be inserted. The 45 is compared first with the center, or G'/2, item of the partial file, which is the 34. Since it compares high, the 45 belongs in the bottom half of the partial file. The next comparison is with the (3/4) G' item, the 60, to which the 45 compares low. It, therefore, belongs in the third quarter of the file. The 45 is next compared to the (5/8) G' item,

the 56, and still compares low. Since this segment of the file contains only one item (the 56), the 45 belongs next to the 56 and is inserted immediately above it. Insertion step 10 shows the 45 inserted in this slot, the old items (56, 60, 69, and 83) having been shifted down one position.

The next item to be inserted is the 37. The first compare at the center (G'/2) item, 34, is high; hence, the 37 belongs in the bottom half of G' (see insertion step 10). The second compare, at the (3/4)G' item (60) is low; this eliminates the bottom (fourth) quarter file. The third compare, at the approximate (5/8) G' item (45), shows the 37 still low; it is, therefore, inserted immediately above the 45, the old items (45, 56, 60, 69, 83) being shifted down one place (see insertion step 11). This procedure is continued until the file is in order after 16 steps of insertion. Note that from step 13 on, four comparisons are necessary to locate the insertion point for each new item in the lengthening file. (The number of comparisons increases logarithmically with G'.) Note also, in step 13, that no shifts are necessary to insert the 96, since it compares high to the remainder of the file and is placed at the bottom.

It can be shown that for a file of G items, the approximate total number of comparisons (C) required in the binary insertion method is

$$C = G \log_2 (G/e)$$

where
$e = 2.7183$

Thus, in example 2, the approximate number of comparisons is

$$C = 16 \log_2 (16/2.72) = 16 \log_2 (5.9) = 16 \times 2.56 = 41$$

The total number of shifts required in binary insertion is $(G^2 - G)/8$. A record storage area for G items must be provided. The major disadvantage of the binary insertion technique is the need for a large and complicated program. This offsets considerably the benefits derived from the relatively efficient search technique.

The binary insertion technique is affected, to a major extent, by the natural ordering of the original file. Although the number of insertion steps and comparisons does not depend upon the original order, the amount of record movement required is greatly increased with adverse, or reverse sequencing.

The method is too slow for larger G's (G>50) primarily because the number of shifts and comparisons goes up with the square of the number of items to be sorted. Insertion, however, can be used to advantage when the

input is buffered and consists of single records, which may be read in one at a time while a previous record is being inserted.

REPLACEMENT SELECTION

The internal sorting methods described thus far are all capable of sorting a group (G) of records that can be contained at one time in the record storage area. The maximum string length is, therefore, limited to G items. The replacement (sometimes, replenishment) technique, endeavors to keep the record storage area filled with G items by replacing records that have been withdrawn during the sort. As a result, for a file in random order, an average string length of approximately 2G items is developed in an area with a capacity of only G records. For a given amount of available core storage, the replacement technique produces the maximum possible sequence length. This characteristic makes the technique eminently suitable as a premerge sort and permits a significant reduction in the number of merge passes required for a subsequent external sort. The price paid for this advantage is increased complexity of programming, relatively long processing time per record, and a slight increase in the required working storage. One must also keep in mind the fact that the number and length of the sequences is variable and, hence, not predictable. Most replacement sorts, however, will generate string length approximating 2G.

Essentially, the replacement-selection method determines the lowest record in the record storage area, moves it to the output area, and then replaces it with a new record from the input file. If the new record is lower than the one just moved to the output, it cannot be part of the current sequence and, therefore, is flagged or held for the next sequence. The process then continues with the selection of the next-lowest record, and so on, until there are no more replacement records in the record storage area that fit into the current sequence. A new sequence then is started, and the procedure continues until the entire input file is processed. Since the sequences are usually formed by a binary tree procedure (see example 3), the method has also become known as a "Christmas Tree Sort". However, not all replacement sorts are of the tree variety.

Basic Tournament (Christmas Tree) Sort

In a basic, simplified version of replacement selection, illustrated by example 3, the method is implemented by playing an elimination "tournament" of matches (compares) between pairs of records to select a "winner" (low) record. The number of records to be sorted internally (G) is a power of 2 (G = 4 in example 3), though this is not necessary in the more sophisticated practical versions to be described later.

The tournament is initialized by selection of the first winner (that is, the record with the lowest key). The keys of successive pairs of records in the record storage area are compared, and the winner of each match (that is, the lowest key or its address) is placed in a first-level table. Pairs of level 1 winners are then compared, and the winners are placed in the second-level table of winners. The procedure is continued, in standard tournament style, until the winner of the final match is determined, and the winner record is moved to the output. This completes the initialization round.

During the second round, or pass, the next record from the input file replaces the vacancy created by the winner, and the tournament continues. If the key of the new record is lower than that of the previous winner, it is flagged (shown by asterisks in example 3) and held for the next string. When all records in the storage area have been flagged, the current sequence is complete. The tournament then is initialized again to process the next string. This procedure is continued until the input file is exhausted.

In example 3, four items (13, 69, 56, 02) from the input file of 16 are read initially into storage. (To keep the number of comparisons to a minimum, a G of only 4 has been chosen; this permits the entire procedure to be illustrated on one page.) The first pair of items, 13 and 69, are compared, and the winner of this match, the 13, is placed on the first level. The second pair of items, 56 and 02, are compared, and the winner (02) is also placed on the first level. The winners of the first two matches, 13 and 02, are now compared in the "finals" match. The winner (02) is moved to the second level and thence to the output area. This completes the initialization.

On the second pass, a new item (08) has been moved into the location of the first winner. The 08 is compared with the 56, and the winner (08) is moved to the first level. The finals are played between the 08 and the 13, and the winner (08) is moved to the output. Note that on this pass (and on every pass after the first) only two compares are needed to determine the final winner. In general, the number of comparisons to find a winner -- after the first -- equals the number of levels in the tournament.

The procedure continues on subsequent passes with the determination of additional winners, but starting with pass 7, each replacement item is lower than the previous winner and is, therefore, flagged for the next sequence (as indicated by the asterisk). First-level winners, between flagged items, are determined, but no flagged item can be moved to the output. In pass 9, all items except the 83 are flagged; thus, the 83 completes the first sequence. In pass 10, all items are flagged, and the tournament must be initialized again to begin the second string. Again, three matches are necessary to determine the first winner, the 17, in this case. The second string is completed at the end of pass 16 with the transfer of the winner (96) to the output area.

(An additional pass would be required to determine that all locations have been flagged and that no more input items are left.) Note that the first string is nine items long, or more than twice the number of items (G) in storage, while the second string of seven items is a little less than 2G in length. When the input file has good natural sequencing, the strings are generally longer than 2G in length, and in the worst case of reverse sequencing, string length is either G or G + 1.

It is easily shown that G - 1 comparisons are required to select the first winner record. The number of comparisons (c) required to select each additional record, after the first, equals the number of testing levels. When G is a power of 2,

$2^c = G$, and hence, the number of testing levels or comparisons per record,

$$c = \log_2 G$$

Accordingly, the total number of compares (C) for G records in storage is:

$$C = (G - 1) + G \log_2 G$$

When G is not an exact power of 2, the formula above is still approximately correct. The number of testing levels in this case is the smallest integer greater than $\log_2 G$, but since some records are not tested on every level, the average number of compares per record is a fraction, which approximately equals $\log_2 G$. (See discussion of example 4.)

Variations and Refinements

It is not necessary, in practice, that the number of records sorted internally (G) be equal to a power of 2. This would waste a great deal of storage space, especially if G were nearly, but not quite, equal to a power of 2. (For example, if 127 records could be contained in the record storage area, the next-lower power of 2 would be 64 records only, thus wasting almost half the available space.) If the branches of the binary-tree structure are properly organized, practically any number of items that can be contained in the record storage area can be sorted by replacement selection. However, departures from a power-of-2 structure frequently involve minor inefficiencies, and care must be taken to keep the average number of comparisons to a minimum. In general, the more symmetrical the tree structure, the lower will be the average number of comparisons per record.

Example 4 illustrates the possible organization of the binary tree when the number of records in storage (G) equals 12. Only the initialization of the

tournament through selection of the first winner record (key 02) is shown. Four testing levels are required. (Note that 4 is the smallest integer greater than $\log_2(12) = 3.59$, approximately.) The keys of pairs of records are compared to select the first-level (quarter-finals) winners -- 13, 02, 08, 34, 45, and 22. Pairs of level-1 winners are then compared to select three level-2 (semi-finals) winners: 02, 08, and 22. Only two of these (02 and 08) need be compared to select the level-3 winners, or finalists (02 and 22). The winner of the finals, 02, is placed on level 4, and the corresponding record (R 4) is moved to the output area.

In example 4, the total number of comparisons required to select the first winner is 11, or one fewer than the number of records in storage (that is, G-1). Note that records 1 through 8 go through four testing levels, while records 9 through 12 go through only three testing levels. Thus, the average number of tests per record (after the first) is a fraction, given approximately by $\log_2(12)$, or 3.59. (The actual average number of compares is 3.66, rather than 3.59.)

Further flexibility for sorting any group (G) of records in storage can be attained by abandoning the binary tree structure. A binary tree results when the comparison point, or node, is between two records to be compared at one time. However, a node may be established between any given number of records, such as three, four, or even eight records. The nodes are tied together in a treelike structure, similar to the binary tree. Larger-size nodes tend to be more efficient for a given group (G) of records. A node of four items to be compared is commonly used.

Example 5 illustrates the possible organization for an internal sort of 64 records (G = 64), using nodes of four, each. The number of testing levels here is $\log_4(64) = 3$, as shown. The first winner to initialize the tournament is selected by (G-1), or 63, comparisons, as for the binary tree. However, the number of compares required for the selection of subsequent winners tends to be proportional to $\log_4 G$, rather than to $\log_2 G$, and, therefore, is somewhat more efficient than for a binary tree.

Tag Sorting

If replacement selection required the continual movement of records through every testing level, it would be a relatively cumbersome and slow sorting technique. Actually, records are not moved at all during the comparison procedure, but only the record addresses or tags. Records are moved initially from the input to the record storage area for participation in the tournament. While the tournament is progressing, the records hold their positions in storage, and only the record addresses (tags) are used. Comparisons between records are made through an index register or other coding

methods. After a record has been selected as the smallest, it is written out or moved to an output area, and a new input record replaces the winner in the identical storage location. Tag sorting not only saves an undue amount of record movement, but becomes essential when dealing with records of variable length.

QUADRATIC SELECTION

Sorting by selection -- perhaps the simplest of the internal sorting methods -- consists essentially of an examination of the record storage area to find the record with the smallest key (for an ascending sort) and placing this record or its key in the output area as the first item of the new file. The RSA is then scanned for the smallest key of the remaining records, which becomes the second item of the new file, and so on, until all items have b een placed in the output file.

When the selection process is carried through the entire RSA in one stage, it is called linear selection; when the original file is broken up into groups, and the smallest key of each group is chosen, and then the smallest of these smallest keys, the process is termed quadratic selection. By breaking the groups into smaller subgroups and then selecting the smallest key of such a group of groups of groups, cubic selection may be accomplished, and so on, up to nth degree selection of $(\text{groups})^n$.

Selection requires a relatively small working storage area, equal to the number of items being sorted (G).

In linear selection, the entire RSA is searched for the record with the smallest key (control word) during each pass. When it is found, the key (or the record containing it) is placed in the output area. To make sure that the same record is not selected again, the key that has been removed is replaced by a key of all nines (or all Z's) in the original file. The process is continued until all keys in the file have been moved to the output and been replaced with nines. This is sometimes called sifting with nines. Example 6 shows the results of sorting a file of six numeric keys with six selection passes. Keys to be selected during the next pass are underlined.

Note that at the end of pass 6, all keys have been replaced with nines, and the output file is in sequential order.

With quadratic selection, the original RSA G is divided into $\sqrt{G}$ groups of $\sqrt{G}$ records each. The sort is most efficient if G is a perfect square, such as 4, 9, 16, 25, 36, etc.; hence, the term quadratic. (If G is not a perfect square, the file is broken up into $\sqrt{G'}$ groups, where G' is the next-largest integer that is a perfect square. In this case, not all groups will have the same number of items.)

After the RSA has been divided into groups, the record with the smallest key in each group is determined by a linear selection pass. The selected least keys (and addresses) form a new subgroup, which is moved to a separate storage area, or control register, capable of containing $\sqrt{G}$ items. The smallest key of this subgroup, which is the smallest of all, is now selected. In order not to select this key again, the key is replaced in the original group by a number larger than any possible key in the file; for example, all nines or Z's. The entire procedure of initially selecting the smallest key in the RSA is called initialization or, sometimes, priming.

The program now goes back to the group from which the smallest key was chosen and selects a new minimum from this group. This is moved again to the subgroup of smallest keys (in the control register), and a smallest-key-of-all is chosen. In this way, successive smallest keys are located, and a sorted output file is built up. The major advantage of the method is that once the file has been initialized (that is, the first minimum key has been found), a linear selection pass need be made only over the group of items that contributed the last minimum. This saves many comparisons, especially for a large file. Despite the need for a small amount of additional working storage (for the subgroup of smallest keys), quadratic selection can be a highly efficient internal sorting method.

An example of quadratic selection with a RSA of 16 items, divided into four groups of four items each, is shown in example 7. For clarity, the keys that have been selected are crossed out, rather than replaced by all nines. The first, or initialization pass, results in the selection of keys 02, 08, 22, and 17 for the control register and of 02 as the smallest-key-of-all. During the second pass, a selection is made in group 1, from which the previous key was chosen. This results in key 13 being moved to the subgroup of smallest keys, or control register. Key 08 "wins" during this pass as the smallest of all. The remaining passes, 3 through 16, shown in the example for completeness, follow the same procedure and are self-explanatory. The complete list of "winners" -- the sorted file -- appears at the output of pass 16.

It is apparent from example 7 that the required number of comparisons decreases drastically as more and more keys are eliminated from the file. The total number of comparisons in quadratic selection, for a file of G items, is

$$C = (G - 1) \times (2\sqrt{G} - 1)$$

When G is relatively large (G>>1), this, approximately, becomes

$$C = 2\,G\,\sqrt{G}$$

Thus, the number of comparisons required in quadratic selection increases with the three-halves power of the number of items to be sorted.

The initial sequencing of the file has no effect on the total number of comparisons. The number of passes in quadratic selection equals G and the total required core storage is equal to $G + \sqrt{G}$.

INTERMEDIATE MERGE PHASE - EXTERNAL SORTING TECHNIQUES

TAPE SORTING METHODS

The object of tape sorting is to bring together sequences of records, developed during the internal sort, into a single, sequenced tape. This is usually accomplished by some kind of merging. The merge process may be of the balanced type, using an equal number of input and output work units, or it may be of the unbalanced type, using an unequal number of input/output work devices. Among unbalanced merge methods, the following types are described:

1. Polyphase Sort
2. Oscillating Sort.

Balanced Merging

In any balanced merge the records are moved back and forth between an equal number of input and output tapes, so as to avoid tape changing. The output sequences from the internal sort of the previous phase are written on one-half of the tapes available for merging. About the same number of sequences are placed on each of these input tapes. The remaining half of the available tapes is reserved for the output of the merge. Initially, one sequence from each input tape is merged into one longer sequence, which is placed on the first output tape. The second sequence from each of the input tapes is then merged onto the second output tape, and so on, until all input sequences have been merged onto the output tapes in a cyclic fashion. The output tapes now become the input tapes for the second pass, and the input tapes from the previous pass become the output tapes. Each record is processed on each pass. During each pass the total number of sequences is divided by the order of merge (m), while each sequence is lengthened by a factor equal to the order of merge. The process continues until the last pass results in a single sequence on one of the output tapes. Where 2m tape units are available for an m-way merge, the total number of passes (P) required is the smallest integer that is equal to or greater than $\log_m S$, i.e.

$$P = \lfloor \log_m S \rfloor$$

where S is the total number of sequences to be merged.

While it is generally true that the higher the order of merge, the fewer passes will be required, this is not necessarily correct for a particular generalized sort. The most efficient order of merge is not always the highest, but is determined by a combination of the following factors:

1. Internal storage capacity
2. Block length
3. Size of control field
4. Length of records
5. Ratio of input/output (tape) speed to processing speed.

The last factor is particularly relevant in choosing the order of merge. For a low order of merge, the time required for comparing the keys is less than that needed for reading in the records, so that the overall speed is determined by the tape-reading speed, that is, the sorting speed is tape-limited. Thus, it is important to keep the tapes moving at all times and to have the block that is needed next readily accessible. On the other hand, when both the tape speed and the order of merge are high, the sort may become process-limited; that is, the sorting speed is determined by the machine times for basic arithmetical and transfer operations. Most present day machines have several channels available that permit overlapping input/output operations with processing, so that both are performed at the same time. This results in the most rapid merge possible for a given machine and method.

Balanced merging has been adapted for generalized sorting programs because it has a number of significant advantages over unbalanced (asymmetrical) merging. The balanced merge requires no tape changing and a minimum amount of programming. A generalized, unbalanced program is not only larger, but it reduces the maximum file size to that which can be sorted on the side (input or output) with the smaller number of tapes. (As will be seen later, the maximum file size for the specialized, unbalanced merges is one to m - 2 reels of tape.) It is, moreover, desirable to attain a complete overlap of reading and writing, so that reading is performed on one or more channels at the same time that records are written out on the other channels. This is accomplished most easily with a balanced number of input and output channels. Additional economies in rewind time are possible if the tapes can be read backward as well as forward.

In general, the balanced merges used in generalized sorting programs are more adaptable to machine schedules and the usual organization of tapes than are unbalanced merges. On the other hand, where maximum sorting effieiency and speed are desired, the polyphase or oscillating unbalanced merges described below should be considered.

Polyphase Merge Sort

The polyphase technique is capable of performing (m - 1)-way merging from m tape units; the output sequences from the internal sort are, however, distributed in a prescribed ratio. When the number of available tape drives is fewer than eight ($m<8$), polyphase merging generally is faster and requires fewer passes than balanced merging. For a larger number of tapes, polyphase techniques begin to lose their advantage. Also, when the total number of strings to be sorted cannot be fitted easily into the prescribed distribution ratio, an elaborate presort, or adjustment phase, may be required that will offset some of the timing advantage gained from the technique.

In polyphase merging, the sequences are distributed initially onto m - 1 of the available tapes, as is described later. An (m - 1)-way merge pass from the (m - 1) input tapes onto the mth output tape is then performed, until the tape with the least number of sequences is depleted. At this point, however, the previous output is made an input tape, and the (m - 1)-way merge is continued by merging additional strings from the tapes not yet depleted with strings from the tape just created. Note that the original output tape, which is now an input, contains records that are entering the merge for the second time, while sequences of records from the longer tapes have not yet been merged. As each successive input tape reaches its end-of-file, the previous output tape replaces it, and the depleted input tape becomes the new output tape. Since this is a continuous process, at no point can it be said that a complete pass over the file has been made, but instead, there are a series of partial passes, or phases, wherein sequences from several previous phases are merged together. The sort ends when all strings have been merged into one sequence.

Example 8 illustrates a read-forward polyphase merge of 57 strings with four tape units. The strings are distributed initially in the ratio 13, 20, 24 onto input tapes A, B, C, respectively, with tape D serving as output. After a three-way merge of 13 strings onto tape D, input tape A is exhausted, with seven strings left on tape B and eleven on tape C. Tape A now becomes the output tape, and tape D, with 13 newly created sequences, serves as input tape. After another three-way merge of seven strings, from tapes B, C, and D onto tape A, tape B is exhausted. Four strings remain on tape C, six on tape D, and seven new strings have been created on tape A. The next three-way merge, from tapes A, C, and D, onto tape B, depletes tape C, leaves two strings on tape D, three on tape A, and creates four new strings on tape B. With tape C serving as output, the next three-way merge of two strings exhausts tape D, leaves two strings on tape B, and one on tape A. Another three-way merge of one string onto tape D depletes tape A and leaves one string, each, on tapes B and C. Tapes B, C, and D now each have one string. A final three-way merge onto tape A merges these remaining strings into a single sequence.

Assuming equal string lengths originally, the total number of records read and written in this example amounts to the equivalent of about four full passes over the file. This compares to six passes required for a balanced two-way merge with four tapes.

String Distribution for Polyphase Merge Sort

As mentioned earlier, polyphase merging depends for its operation upon a special initial distribution of strings, which is designed to permit merging a single string from each tape onto the output tape during the final merge pass. If the required initial distribution is absent, an adjustment and starting process is necessary before the main merge can be entered.

For a three-tape (two-way) polyphase merge sort, the initial string distribution follows a sequence of numbers developed by Fibonacci in the 13th century. In the Fibonacci sequence each term is formed by taking the sum of its two predecessors, thus: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, etc.

In a three-tape polyphase merge, two tapes are available for the initial string distribution, while the third tape is used for the output. To find the distribution from the Fibonacci sequence, look for the term in the sequence that represents the total number of strings to be merged. (If not exact, it may require an initial adjustment or presort.) Then, allocate the strings in accordance with the two successive terms preceding the sum term. For example, if 13 strings are to be merged, eight strings should be allocated to one of the input tapes, and five to the other. Similarly, if 89 strings are to be merged, the distribution of strings on the two input tapes would be 55 and 34, respectively.

The Fibonacci numbers can be generalized to serve for four-tape, five-tape, and higher merges, but to avoid mathematical intricacies, it is best to assemble a string distribution table based upon the following formation rules:

1. The optimum distribution for one partial polyphase merge pass is one sequence on each of the m - 1 tapes available for distribution (the mth tape serves as initial output). Hence, for the first partial pass (line 1 of table) write a 1 under each of the input tapes.

2. For the second, and each subsequent, partial pass, the tape with the largest number of strings (first column of table) receives the sum of strings allocated in the previous pass to the largest and second largest tapes (first and second columns of previous line). The next-smaller tape (second column) receives the sum of strings of the largest and third-largest tapes of the previous pass (first and third columns of previous line in table). The next-smaller tape (next column to the right) receives the sum of strings of the largest and fourth-largest tapes of the previous pass (first and fourth columns of previous line),

and so on, until the tape with the least number of strings (last column at right) is reached; this tape receives the same number of strings as the tape with the greatest number of strings (largest tape) of the previous pass (first column of previous line).

3. Continue building the table until the total number of strings distributed during a partial pass equals the number of strings to be merged. The optimum total number of strings is given in table 4. If one of the optimum numbers cannot be obtained from the internal sort, a string adjustment to the next-lower optimum distribution becomes necessary. Tables 1-3 for four-tape, five-tape, and six-tape polyphase merge sorts illustrate the procedure.

Read-Backward Considerations

To eliminate rewinding after merge passes, the input tapes always are read backward and the output tape always is written forward. However, if the sequences from the internal sort are written in one order (either all ascending or all descending) and then read backward continued merging soon becomes impossible because alternately ascending and descending strings are encountered.

To understand the nature of the problem consider, for example, a simple four-tape (three-way) polyphase merge of 17 ascending strings, distributed in the ratio of 7, 6, 4 onto three input tapes. (This is the required string distribution shown in Table 1 for 17 strings.) A three-way merge is now performed by reading tapes A, B, and C backward and writing the merged output strings forward on tape D until tape C is depleted. It is evident that when the ascending strings on the input tapes (A, B, C) are read backward and are then written out on tape D, they become descending output strings. Thus, at the completion of this first merge pass, four descending strings have been placed on tape D, three ascending strings remain on tape A, two ascending strings are left on tape B, and tape C is empty. The next merge pass, with tape C as output, cannot be done, however, since no way is known to merge the ascending strings on tapes A and B with the descending strings on tape D. The attempted read-backward polyphase sort, therefore, must be abandoned at this point.

Attempted read-backward polyphase merge
of 17 ascending strings

	Tape A	Tape B	Tape C	Tape D
Initial Distribution	7 Asc.	6 Asc.	4 Asc.	0
3-way merge of 4 strings (read-backward)	3 Asc.	2 Asc.	0	4 Desc.
3-way merge of 2 strings	Impossible			

To solve the difficulty, it becomes necessary to create both ascending and descending sequences during the internal sort phase, in such a manner that each read-backward merge can be performed with either all ascending or all descending strings.

Certain peculiarities of polyphase sorting provide the clues for the pattern of alternately ascending and descending strings that must be generated during the internal sort:

1. It has been found that -- with a few exceptions -- every optimum distribution provides one tape with an odd number of strings, and all other tapes with an even number of strings. The tape chosen as output for the final merge pass is always the one with an odd number of strings.

2. Therefore, if the final output is to be in ascending order, the final output tape must have an ascending string at its beginning and end, while all other tapes must begin with a descending string and end with an ascending string. (This order would be reversed if the final output is desired in descending order.) Intermediate strings on each tape are alternately ascending and descending. Generally, it is necessary only to make sure that the first string (from the presort) placed on each tape is of the proper ascending or descending type, since for every optimum distribution shown in the distribution tables (with a few exceptions) the last string will automatically be of the required type. Thus, if the first string on one of the tapes (the final output tape) is ascending, every odd string thereafter will again be ascending, as required; if the first string on each of the other tapes is descending, every even string thereafter on these tapes will be ascending. If the internal sort cannot attain one of the optimum distributions shown in the tables, or if the total number of strings falls between table entries, marked dummy strings must be used to fill out the proper distribution.

3. With the output strings from the internal sort being of the kind described above and being distributed as described, every partial merge pass of the read-backward sort will automatically contain either all ascending or all descending strings. The final pass will merge one descending string from each of the input tapes, which is written out as an ascending single string on the final output tape.

Example 9 shows how the earlier unsuccessful attempt to sort 17 strings by a read-backward polyphase merge can be properly carried through by the formation of alternately ascending and descending strings during the internal sort phase.

It is evident that the output from the internal sort in example 9 satisfies both the optimum distribution for 17 strings and the conditions laid down earlier for string alternation. Tape A, the final output tape, receives seven (odd) strings, starting and ending with an ascending sequence. Tape B receives six

(even) strings, starting with a descending string and ending with an ascending one. Tape C receives four (even) strings, again starting with a descending sequence and ending with an ascending string. Tape D is empty.

Each merge pass is performed by reading the input tapes backward and writing the merged strings forward on the output tape. Note that all ascending (A) input strings to the merge are written on the output tape in descending (D) sequence, and all descending (D) inputs to the merge are written out in ascending (A) sequence. (This takes full advantage of the read-backward feature and avoids all rewinding and copying until after the final merge.) Thus, the first three-way merge of four strings starts with the three A-strings (with asterisks) on tapes A, B, and C, which are written out in descending sequence on output tape D. Three additional strings from each tape, in alternating D-A-D sequence, are merged onto tape D in A-D-A sequence. At the end of this first merge, three strings (A-D-A) remain on tape A, two strings (D-A) remain on tape B, tape C is empty, and four strings (in D-A-D-A sequence) have been newly created on tape D.

Merging continues in the same fashion, and by the end of the third merge, tape A is empty, and one D string remains on each of tapes B, C, and D. The final three-way merge consists of merging these three descending strings into a single string, which is written out in ascending (A) sequence on tape A. Example 10, of a five-tape (four-way) read-backward polyphase merge of 25 strings, further illustrates the procedure.

Example 10 has the same general features and follows the same merge procedure that has been explained for example 9.

In summing up, it appears that the polyphase merge techniques are generally superior to balanced merging, if fewer than eight tape units are used. When a greater number of tapes are used, either the balanced merge or the oscillating sort (described in the next section) seem to have the advantage. Exact generalizations cannot be made, since much depends on tape and central processing speeds.

Finally, for full efficiency, reading and writing must be simultaneous; that is, it should be possible to read from one tape and to write on another in any combination of tapes. To maintain this complete read-write overlap, cross-channel switching is required.

Oscillating Merge

When more than eight tape units are available for sorting, the oscillating merge sort technique generally requires fewer complete passes over the data than do polyphase or balanced merge methods. The oscillating sort attains (m - 1)-way merging with m tapes, makes use of the tape read-backward feature, and for

optimum efficiency, can employ cross-channel switching; that is, the ability to read any tape while another is being written. In contrast to other sort techniques, however, the oscillating sort method integrates the internal sort phase with the tape-merging phase.

The oscillating sort begins with a conventional internal sort, but enters a merging phase immediately after the internal sort has written one string of sequenced records on each of m - 1 tapes. The m - 1 strings are read backward from the tapes and are merged onto the available mth tape. The other (m - 1) tapes are back at the beginning (load point) after being read backward. Control is now returned to the internal sort phase of the program. Another (m - 1) strings are then written by the internal sort, including one on the previous output tape. (A tape mark separates this string from the previously merged output.) Another (m - 1)-way merge pass onto the remaining available tape takes place.

This oscillating process is continued until each of the m - 1 tapes has had m - 1 sequences merged onto it. At this point, therefore, a total of $(m - 1) \times (m - 1)$ or $(m - 1)^2$ strings, generated by the internal sort, have been merged into m - 1 tape sequences. These m - 1 sequences are now again merged onto the available (mth) tape, resulting in a single sequence. This completes the first stage of the sort.

The iterative process now starts over again (if more internal strings are to be sorted) and continues in the fashion described until another tape will contain a sequence formed from $(m - 1)^2$ internally sorted strings. Similar stages follow until, eventually, each of (m - 1) tapes contains a sequence from $(m - 1)^2$ original strings. At this point, therefore, (m - 1) tape sequences have been created from $(m - 1) \times (m - 1)^2 = (m - 1)^3$ original strings. These (m - 1) sequences are now again merged as a single sequence onto the remaining tape. If necessary, additional stages of the sort take place until all input records have gone through the internal sort. A single tape sequence occurs whenever the number of strings processed equals successive powers of the (m - 1) available merge tapes. A final merge onto the output tape concludes the sort. Example 11 illustrates the technique graphically for 27 strings sorted with four merge tapes.

The example is largely self-explanatory, except for a few peculiarities that should be noted. After the initial three strings have been merged onto tape D (steps 1 and 2), the internal sort writes another string on each of tapes B, C, and D (step 3). Since tape D was the previous output tape, a tape mark (/) must be placed between the new string and the previously merged three strings (shown as 3/1). Note that all numbers in the example refer to the number of originally created strings contained in each tape sequence. Thus, in step 3, with a distribution of 0, 1, 1, 3/1 on tapes A through D, there are

only four tape sequences (two on tape D), which consist of six original strings (that is, 1 + 1 + 3 + 1) created by the internal sort. By the end of step 6, each of three merge tapes has three original strings, or a total of nine strings. These are merged onto tape C as a single sequence (step 7). This concludes the first stage.

The second stage of the sort (steps 8-14) repeats the previous process, except that in step 13 it becomes necessary to merge three strings onto the already existing sequence of nine strings, no other merge tape being available. Again a tape mark separates the two string sequences (shown as 9/3). In step 14, another nine strings are merged onto tape A, thus concluding the second stage.

During the third stage (steps 15-22), nine additional strings are merged onto tape D (step 21), and all original 27 strings are finally merged as a single sequence onto output tape B (step 22). If more strings had been generated by the internal sort, the next single sequence (for three-way merging) would have resulted for $(3)^4$ or 81 original strings. If the number of strings generated falls between powers of (m - 1), it is possible to perform a partial (m - 2)-way merge, or lower orders of merge, whenever the number of strings on two or more of the tapes are equal.

Note that the order (power) of merge remains constant at (m - 1) for the oscillating sort. Thus, the average reduction factor, i.e., for each pass is the same as the merge order, or (m - 1). The reduction factor begins to exceed those for balanced and polyphase merging for as few as four tapes, with a merge order of 3. However, in an oscillating sort, the input device (even if tape) is never available as merging tape, since it is used for the internal sort. Inasmuch as other sorting methods can make use of the input tape for merging, the order of merge for the oscillating sort may be considered to be (m - 2) of all m tapes in use. In this case, the reduction factor of the oscillating sort begins to exceed that of polyphase for a total of six tapes in use (m - 6). (Detailed comparisons are given in a later section.) Note also that the oscillating sort has the advantage of facilitating interruptions and restarting at those points where all records are in a single sequence.

DIRECT ACCESS SORTING METHODS

As in the tape sort techniques, the object of the direct access sorting is to bring together sequences of records, developed during the internal sort phase, into a single sequenced file. A basic characteristic of a direct access, namely, that every record in the input area is equally accessible, allows two modes of sorting. In record mode, the entire record is always read into the core, both in the sort phase and the intermediate merge phases of the sort. In tag mode, only the control fields of the record plus the address of the record are used. The output of the tag mode is a list of

direct access addresses. The records, if retrieved according to this list, will be fetched in the desired sequence. The advantages and disadvantages of the tag mode of sorting is discussed later.

Balanced Direct Access Technique

The sort phase distributes sequences onto all but the largest area used for intermediate storage. The order (i.e., ascending or descending) of the control fields of all sequences is the same as the order desired for the output.

The locations of individual sequences in each area are maintained in a directory for each area. The directory for each area is kept in that area and is pointed to by a parameter in a constant area.

For example, if 50 tracks are reserved for a work space, data is written starting at the first track and the directory is written beginning on the last track. The number of tracks used for the directory (a minimum of one track is always used) depends upon the number of sequences to be written in the work area.

The intermediate merge phase combines sequences from a filled area into longer sequences and distributes these sequences onto the empty area. When all sequences from one area are distributed onto another, the first area is considered empty. It can then receive merged sequences from some other area. The merging process continues until the number of sequences is less than or equal to the merge order (in example 12 a merge order of 5 is used). At that time, the final merge phase combines the remaining sequences into a single sequence and places it onto the output device.

Example 12 shows an example of the balanced direct access technique. Area A is the same size as or smaller than area B, which is the same size as or smaller than area C. The sort phase distributes 18 sequences onto both A and B. The intermediate merge phase merges the 18 sequences from B into four longer sequences, which are placed onto C. (Since C is the same size as or larger than B, all the sequences from B fit on C.)

B is now considered empty and can receive sequences from A. The 18 sequences on A are merged into four longer sequences and placed onto B. A is now considered empty.

Since the total number of sequences (eight) on B and C is greater than the merge order (five), another intermediate merge phase pass is required. The four sequences from B are merged into one long sequence, which is placed onto A. (Since B consists of sequences from A, the sequences from B fit on A.) B is now considered empty. The four sequences on C are merged into one long sequence, which is placed onto B.

Since the total number of sequences on A and B is now less than the merge order, the final merge phase is executed next. This phase combines the remaining two sequences into a single sequence and places it onto the output device.

In example 12, the merge order is 5 until the final pass, where it becomes 2. Performance of the sort can be increased by optimizing the merge order to the smallest m which will not cause an additional pass. Secondly, the sequential use of the work areas causes a savings in the number of writes but pays a penalty by increasing the number of reads over the string interleave merge external sort. Finally, the balanced merge allows for variable length sequences, whereas the string interleave method requires fixed length records. The capability of handling variable length strings in the intermediate and final merge phases allows the sort phase to use the replacement selection internal sort technique.

String Interleaving

String interleaving takes advantage of both the sequential and randum characteristics of a disk file to minimize the seek time and, thus, the total merge time. The technique assumes that the usual phase 1 internal sort has developed strings of records of length G, which are blocked in the output. The blocks of each sequence are then interleaved within alternate blocks of disk storage, so as to make the strings to be merged accessible with a minimum number of head movements.

Example 13 illustrates the interleaving technique for a simple two-way merge of four strings contained on four cylinders of a type 2311 disk drive. Two work areas are necessary to perform the sort. As shown in (A) of example 13 the output blocks of each sequence from the internal sort have been written on alternate blocks of the disk file, so that a block of one sequence is always followed by a block of the other sequence. (A sequence or string is represented by the symbol S, and a block is represented by B in the example. Thus, S_1B_1 stands for block 1 of sequence or string 1; S_2B_1 represents block 1 of string 2; S_4B_3 represents block 3 of string 4, and so on.) Note that all the blocks of strings 1 and 2 are contained on cylinders 1 and 2, while the blocks of strings 3 and 4 are located on cylinders 3 and 4 of each area.

To initiate the first merge pass, the first blocks of strings 1 and 2 (S_1B_1 and S_2B_1) are retrieved. Since the access arm must move to cylinder 1, this requires one positioning move and one rotational delay for both blocks to be located. Upon the depletion of one of these blocks, the next block of the same sequence must be retrieved. Assume that S_2B_1 is depleted, and S_2B_2 must be sought. This requires only one rotational delay and no positioning move.

The merge process continues, with only rotational delays involved, until the last blocks of cylinder 1 are to be accessed. Here additional positioning moves of the access mechanism to cylinder 2 become likely. An analysis reveals that the merging of strings 1 and 2 (on cylinders 1 and 2) requires at least two movements of the access mechanism and may require as many as ten, with the probable number being fewer than four moves. Similarly, the merging of strings 3 and 4 (on cylinders 3 and 4) requires from two to ten movements of the access mechanism (fewer than four being probable), the remainder of the seeks consisting only of rotational delays. Thus, the probable number of positional seeks for the entire first merge pass is fewer than eight. This should be compared with from 4 to 40 movements (38 probable) of the access mechanism, in the same general setup, with conventional non-interleaved (i. e., balanced merge) file organization.

At the end of the first two-way merge pass, strings 1-4 have been merged into two new sequences (S_1 and S_2) on cylinders 1 through 4 of the alternate disk area. With the blocks of these two sequences again being interleaved throughout the area of the four cylinders, the input to the second (final) merge pass is as shown in (B) of example 13. The merging of the two remaining sequences proceeds in the same fashion as before, and the probable number of positioning moves is again fewer than eight. It is seen that the interleaving technique reduces the total seek time considerably, not only because there are fewer positioning moves, but also because most seeks involve adjacent cylinders only. It should be noted that the string interleave method requires that each string is of constant length G. So, while seek time is less, the number of passes may be greater than that of the balanced method.

TAG MODE VS. RECORD MODE

Usually, tag sorting is of no advantage, even in large disk files, when most or all of the original records are to be retrieved. Modifying the sort and reading modes to minimize the total seek time can have a considerable effect, but the advantage, generally, still lies with record sorting. The choice of the reading mode -- whether full-track or record-by-record -- depends on several factors, among them the file size, the record length, and the hardware configuration. The original records may be retrieved by reading the file sequentially, in a full-track mode, and then selecting the desired records as each track is brought into core. Records may also be obtained by reading the file, one record at a time, and bringing into core only the records actually desired for that block. In general, the full-track mode appears to be better for relatively small files, while the record mode is better for larger files. The actual dividing line is determined by the record length and the characteristics of the machine used.

To bring the various factors into focus, example 14 provides a specific illustration of the breakdown of overall sort time for record and key sorts

performed on an IBM 7090, with a 1301 disk file, on files of 5000 and 10, 000 records, respectively. The file has been generally organized to minimize seek time. Assume that the records are 100 characters in length and that the key records consist of 20 characters each. Assume further that a computer core working area of 100, 000 characters is available. Note that the example shows only read and write times for the 1301 disk file; the CPU processing time, which represents only a small fraction of overall sort time, has been ignored for simplicity.

Part A of example 14 shows the breakdown of sorting time, in accordance with the three sorting phases. As expected, the tag sorts are considerably faster than record sorting during the internal sort and merge phases. The tremendous impact of seek times on the total sort time becomes apparent during phase 3 of the key sorts, when the original records must be retrieved. Solely because of the retrieval operation, the record sort is seven to ten times faster than the tag sort.

Part B of the example, which gives the breakdown according to file operations, points to the seek time as the cause of the disproportionate retrieval time. The seek time comprises both the time required to position the read-write heads and the rotational delay time. In the 1301 disk file, the time required to position the movable heads is relatively high. Rotational delay is required for each record retrieved during tag sorting, regardless of the number of head movements. Finally, the larger (10, 000-record) file demonstrates even more clearly the advantage of record sorting over tag sorting, in this particular case.

In summary, whether a record sort or a tag sort should be used to sort a direct access file depends largely on the ultimate disposition of the sorted records. If only an index of sorted records is necessary, and few of the sorted records are actually used, tag sorting would appear to have the edge. Reports by exception, which are extracted from the sorted file, is an example of this type of situation. On the other hand, if most or all of the original records are to be retrieved, record sorting is preferable to tag sorting. Moreover, the advantage increases with the size of the file. There are, however, circumstances where the time for record retrieval may be of secondary importance. If, for example, after completion of phases 1 and 2 of the sort, the computer can be put to work on other tasks, the slow retrieval operation on one or more channels may be of no particular consequence.

COMPARISON OF TAPE SORTING TECHNIQUES

Because of its speed, versatility, and ease of programming, balanced merging is still frequently used for merging medium-size to large files in generalized sort programs. A balanced number of input and output channels provides the most efficient overlap of reading and writing, and where simultaneous tape operation is possible only on two channels, the balanced merge is the standard method. With T tape units available, the balanced method attains T/2-way merging of the internally sorted strings (S) of records in

$$\left[\log_{0.5T}(S)\right] \quad \text{passes.}$$

For N items in the file to be sorted, the number of strings (S) created by the internal sort is generally N/G, except in the case of replacement sorting, where the number of strings, S = N/2G. (Here G is the number of items held at one time in internal storage.)

Since the power (order) of merge of the balanced method is only half of the number of tapes available (T/2), efficient merging of many strings requires a relatively large number of tape drives. Unbalanced methods have a considerably higher effective merging power than balanced sorting and, therefore, provide faster, more efficient sorting, especially when the number of available tape drives is limited. As is shown later, this is true even if the number of available tapes is large, for example, more than eight.

The major available unbalanced merge methods have been described previously, but their relative effectiveness under various circumstances remains to be determined. This cannot be done exactly, since the sort efficiency depends not only on the sort parameters, but also on the relative tape and central processing speeds (whether tape-or process-limited), the file characteristics, and other factors. Moreover, the techniques are not directly comparable in some aspects. For example, the partial passes of the polyphase merge, which combine previously processed strings, cannot be compared to the complete file passes of the balanced merge method.

Nevertheless, one traditional way of comparing merge techniques has been through the effective power of merge, or reduction factor, which may be defined as the average reduction in the number of strings occurring during each pass (or partial pass) of the merge. In a balanced merge of order (power) m, the number of strings is divided by m during each pass, while--at the same time--the length of each string is multiplied by m. With T tapes available, the reduction factor, or effective power of the balanced merge, is, therefore, m = T/2. (For example, for a two-way merge with four tapes, the number of strings is halved during each pass.)

Similarly, for an oscillating sort with T tapes, the effective power of the merge is either T-2 or T-1, depending upon whether the input tape is considered to be a merge tape. Since the input tape in an oscillating sort is used for internal sorting and, hence, is never available for merging, the merging proceeds effectively with one fewer merging tape than is available in other sorts. For this reason, the effective power of merge for the oscillating sort should be taken as T-2.

Finally, for the polyphase merge sort, the effective power of the merge varies approximately between T-1 and T/2, depending on the amount of input data and the number of strings. (The degree of variation is described later.)

Table 5 compares the effective merging power of the balanced, polyphase, and oscillating sorts as a function of the available tape drives. The merge powers listed for the polyphase technique apply for a very large (theoretically infinite) number of strings, which correspond to the optimum initial string distributions from the internal sort.

Note in the table that for the balanced and oscillating sorts, the effective power of merge goes up in approximately equal steps, as the number of tapes increases. In the polyphase merge, however, the steps become successively smaller with an increasing number of tapes, and the merging power approaches a maximum (equal to four for an infinite number of strings). Thus, as the number of merging tapes goes up, the polyphase technique becomes progressively less effective, compared to other merge methods. On the basis of the table, a balance between polyphase and other techniques is reached for a total of six tapes. When fewer than six tapes are available, the polyphase merge would appear to be superior to all other merge techniques. For more than six tape units, the oscillating sort would appear to be preferable to polyphase merging, and for more than eight available tapes, even balanced merging appears more effective. Thus, judging from the table alone, it could be concluded that polyphase merging is superior to other methods for fewer than six available tape units, while the oscillating technique is superior for six or more tape units.

Unfortunately, the table of effective powers of merge is an oversimplification, since it is based on somewhat unrealistic restrictions (perfect string distributions, very large number of sequences, etc.) and does not take into account many applicable sort and peripheral circumstances.

For example, for ten available tapes, and ignoring the number of read reversals, the oscillating sort would seem to be far superior to the polyphase merge (with a merge power of 8 compared to 3.95). However, for ten tapes and, for example, 100 strings to be sorted, the oscillating sort requires 13 read reversals, the polyphase merge only 4. Thus, if there is any substantial

delay in effecting the read reversals (read-reverse interlock time), the polyphase merge would be more effective, despite its lower average merging power.

More important, the table of effective powers of merge (Table 5) is based on a very large, impractical number of strings and does not gauge the power of each method for varying numbers of strings. In most large scale machines (for example, a 128K S/360), the OS sort for 100 character records, generates on the average string sizes of 1170 logical records. Therefore, the initial number of strings to be merged in a 200, 000 record file is about 170. A better way of comparing sorting techniques would, therefore, consist of determining for each the number of complete passes over all data for varying numbers of strings. Here the old difficulty is encountered of being unable to compare the partial passes (phases) of the polyphase technique with the complete file passes of other methods. The balanced and oscillating sorts consist of complete passes over the file, though in the oscillating technique these passes are not contiguous. The polyphase merge, however, consists of a number of partial passes, or phases, which combine the strings of several prior phases. Some of the original strings are processed many times, others only a few times, and one only once.

The way out of this difficulty is to count for each method the actual number of times each original string must be processed during the entire merge (a measure sometimes referred to as string passes). By dividing this total number of strings processed (string passes) by the initial number of strings (from the internal sort), an equivalent number of complete data passes over the file is obtained for each method. For the balanced and oscillating sorts, the number of data passes is simply the conventionally calculated number of passes, since, by definition, each pass is over the entire file, with each sequence being processed only once. Thus, with T tapes available, the number of data passes for a balanced merge,

$$P = \lfloor \log_{T/2} (S) \rfloor$$

while for the oscillating sort (power of merge T-2), the number of data passes,

$$P = \lfloor \log_{T-2} (S) \rfloor$$

where S is the number of strings generated by the internal sort.

For the polyphase merge, the number of data passes for each optimum initial string distribution (in accordance with Tables 1, 2, 3 and 4) is:

$$P = \frac{\text{Total Number of Strings Read (String Passes)}}{\text{Initial Number of Strings}}$$

The total number of strings processed must be determined by an actual count of the merge process, as is shown in example 15 for 65 initial strings and six tapes. (For an imperfect initial string distribution, a sufficient number of dummy strings must be added to attain the next higher optimum distribution.) Thus, the number of data

$$\text{passes} = \frac{\text{String Passes}}{\text{Number of Initial Strings}} = \frac{208}{65} = 3.2$$

Table 6 shows, for each of the major merge methods, the required number of data passes for roughly comparable numbers of initial strings and for various numbers of available tapes (T = 4, 5, 6, 8, 10). The table is based on perfect string distributions; the string lengths produced by the internal sort, as well as the capabilities of the tape drives, are assumed to be identical for all methods.

The table is interpreted as follows:

For the balanced and oscillating methods, each range of strings listed corresponds to the number of integral data passes shown in the adjoining column. For the polyphase technique, the high end of each string range corresponds to a perfect distribution, while the low end represents an imperfect distribution. The number of fractional data passes have been computed for both ends of each range and are shown in the adjoining column. Within each polyphase string range, the increase in number of data passes is reasonably smooth, so that intermediate string numbers can be obtained by interpolation. (For all other methods, however, even one additional string at the end of a range causes an extra pass.) All comparisons must be made on the basis of the same number of initial strings produced by the internal sort. (There is an exception to this, which is explained later.)

To compare the relative effectiveness of the merge methods, the data in the table have been plotted as smooth graphs (for T = 4 and T = 10), ignoring the stepwise increments in the number of passes (see Figure 2). This discriminates somewhat against the polyphase technique, which actually has a fairly smooth curve. Nevertheless, the superiority of the polyphase merge over the other methods for a four-tape merge is clearly evident. As the number of strings increases to 100 and beyond, the polyphase technique outperforms the oscillating and balanced merges by about two passes. (If the step increments are taken into account, the difference is even greater.) On the other hand, with ten tapes available, the lineup is almost reversed. For 100 strings or more, the oscillating sort is clearly best, and the polyphase and balanced techniques almost a pass behind.

An examination of the table of data passes (Table 6) reveals that the cutover point in the effectiveness of the techniques occurs somewhere between T = 6 and T = 8. With six tapes available, the polyphase merge still appears

superior to all other techniques. With eight tapes available, however, the polyphase merge appears evenly matched with the oscillating technique for fewer than 100 strings, and it is slightly superior for more than 100 strings.

The following consideration gives an additional edge of effectiveness to the polyphase merge over its runner-up, the oscillating sort. When the replacement method is used for internal sorting, the average string length is twice the number of records stored internally (2G). For N input records, the polyphase sort is thus required to process N/2G initial strings. The oscillating internal sort, on the other hand, produces somewhat smaller strings, varying between G and 2G in length. (The first string is 1.7 G in length, the last 1.0G, and the remaining strings are 2 G in length.) As a result, for a given file, the oscillating sort must process a greater number of initial strings than the polyphase method (instead of an equal number of strings, as assumed previously). The additional number of strings depends on the number of tapes available. For T = 4, the oscillating sort must process 48% more strings than polyphase; for T = 6, 20% more strings; for T = 8, 12% more strings; and for T = 10, 8% more strings.

When the table of the number of data passes (Table 6) is adjusted for this additional percentage of strings in the oscillating sort, the polyphase technique emerges clearly superior to all other methods for fewer than eight tapes available. For eight tape units, the polyphase technique is about equal in effectiveness to the oscillating sort. For more than eight tape drives, the polyphase merge usually requires more data passes than does the oscillating sort. For fewer than eight tape drives available, the implementation of the polyphase technique is practically always of advantage.

GENERAL CONSIDERATIONS

As explained previously, the sort performance depends not only on the choice of techniques, but also on the characteristics of the file, the machine, and the application. A complete discussion of these characteristics and their interactions would require a large volume. Only a brief discussion of some relevant factors and considerations is possible here.

FILE SIZE

The number of records that must be sorted is obviously important, both with respect to the choice of technique and to the type of storage that can be used. The maximum number of records that can be handled by a sort program depends on the number of tapes used, on the method of writing the records, and on the equipment. Some machines permit writing larger blocks of records--thus, increasing the maximum file size--than do others. More records can be written on a single tape reel if they are blocked, for example, to 3000 characters,

instead of to 1000. The length of the records is a limiting factor in terms of their maximum number per reel for a given block length and character density. Finally, the number of reels, and, hence, the maximum file size, depends on the type and order of merge. For a balanced merge, this file size is one reel fewer than the order of merge; for example, a four-way merge can accommodate three reels of records.

RECORD LENGTH

The length of the record, or sometimes, the ratio of record length to key length, determines the amount of manipulation that is feasible and economical in sorting the record. This factor is less important when the control word (key) is separated from the record, and only the keys are sorted. Variable-length records pose a problem in generalized sort programs, since the blocking of such records requires considerable processing time.

LENGTH, LOCATION, AND RANGE OF CONTROL WORD (Key)

The characteristics of the control word, or key, are of over-riding importance in most sorting methods. The required time for processing is proportional to the length of the key (and the number of control fields contained in it). The length and type of key--whether numeric, alphabetic, or alphameric--greatly influences the choice of sorting method. In some methods, the difference between a two-bit numeric key and an 18-character alphameric key is crucial.

DEGREE OF ORIGINAL ORDERING

In many internal sorts the degree of original ordering affects the total amount of manipulation and the required number of passes. Several different concepts must be distinguished, however. The degree of ordering may refer to a scale, ranging from a file completely out of sequence (that is, in reverse order) to one almost ordered. Several internal sorts, such as selection and exchanging, are highly sensitive to this type of ordering. The term may also refer to the number of naturally occurring sequences within the file. A file of N items in random order will have, theoretically, N/2 such natural sequences, each having an average length of two items. Longer sequences, stemming from some previous ordering of the file, are used to advantage in the replacement selection internal sort method and may save one or more merging passes later.

BLOCKING

Records are blocked for sorting to reduce tape start/stop time per record and to increase the total number of items that can be written on one reel of tape. Separate factors are used for input blocking, for output blocking and for sort blocking. Although the efficiency of the sort does not depend entirely on the

relative size of the blocked records, optimum blocking and sort efficiency go hand in hand.

Several indirect factors must also be considered when discussing blocking. Increasing the block size reduces the tape time per record. This is of advantage only if the sort is not process-limited. In sorting programs that tend to take more processing time than tape time, however, increasing block size serves only to increase the number of records per tape reel. On the other hand, the maximum block size varies inversely with the order of merge. As a consequence, the highest possible order of merge does not always provide the fastest sort. Unless a merge pass can be saved, it may be preferable to use a lower order of merge and a larger block size.

In an unbuffered sort, the total running time depends directly on the tape time. The tape time is reduced most significantly by completely overlapping reading and writing. The block size must be chosen to accomplish this purpose.

The significance of user choosing an optimal blocking size for the input and output blocking factor cannot be overemphasized. It has a direct bearing on G and, thus, the efficiency of core utilization. Also, unless a sort is of hours duration, blocking is more critical than sorting technique used in overall run time. The optimum technique for a given file size and number of available intermediate work units will usually not overcome bad blocking.

RELATIVE TAPE AND PROCESS SPEEDS

In any given machine, sorting is most rapid when tape and process speeds are approximately in balance and when both operations are overlapped. When the order of merge during the merge phase and the tape speed are both low, the sort becomes tape-limited; that is, total sort time depends on tape-reading speed. When both order of merge and tape speed are high, however, the sort may become process-limited; that is, the sort time is determined by the speed of the basic arithmetic and transfer operations. Some of the factors that tend to make a sort process-limited are listed and explained briefly below:

1. Blocking -- tends to make a buffered operation process-limited by reducing tape time per record.

2. Buffer transfer time -- adds to process time. It may also add to tape time when using read-while-write mode.

3. Checking and restart procedures -- can add as much as 20% to process time.

4. Size of control word -- affects process time. This is true for both variable-and fixed-length word machines.

5. Number of control fields -- determines process time during internal sort and merge phases.

6. Order of merge -- process time per record increases with merge order.

7. Record length (at which a sort becomes process-limited) -- depends on size of control word, tape speed, and particular machine involved. Merge type sorts of short records are often process-limited.

8. Start/stop (acceleration) time (per record) -- varies with type of tape control used. With fairly short, unblocked records, start/stop time may be considerably longer than time required to read entire record.

9. High tape speeds -- tend to create more process-limited situations.

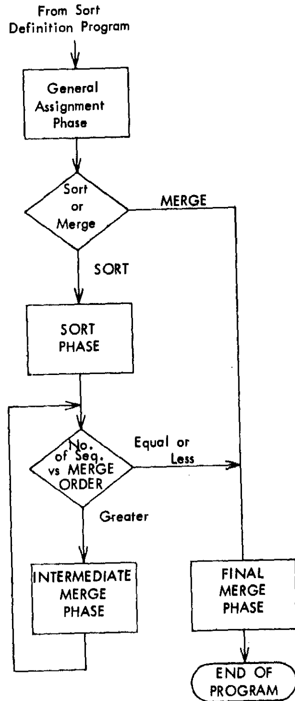
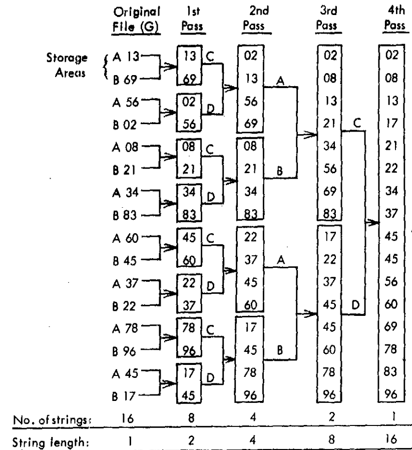


Figure 1. Phases of a generalized sort/merge program



Example 1. Straight two-way merge

S/360 SORT PUBLICATIONS

S/360 Sort/Merge Specifications

Mod 20 Tape Programming System C26-3804
BPS C24-3320
BOS C24-3321
TOS C24-3438
DOS Tape - C24-3438
Disk - C24-3444
OS C28-6543

S/360 Sort/Merge Program Logic Manuals

Mod 20 Tape Programming System Y33-9005
BPS Z24-5008
BOS Z24-5001
TOS Z24-5016
DOS Tape - Z24-5016
Disk - Y24-5021
OS Y28-6597

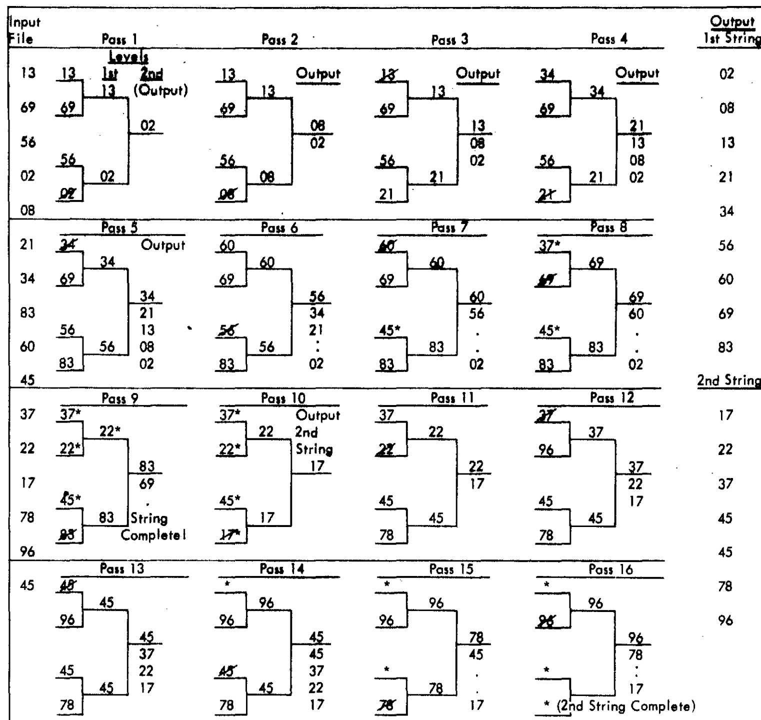
A general reference to sorting techniques may be found in the following IBM publication:

Sorting Techniques, C20-1639

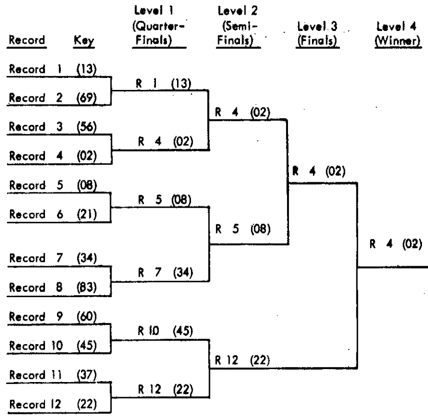
Item No.	Input File	Insertion Steps 1...9	10	11	12	13	14	15	16
1	13								02
2	69				02	02	02	02	08
3	56	02	02	02	08	08	08	(3) C/H 08	13
4	02	08	08	08	13	13	13	(4) C/H 13	17
5	08	13	13	(2) C/H 13	21	21	21	(2) C/L 21	21
6	21	21	21	(3) C/H 21	22	22	22	22	22
7	34	(1) C/H 34	(1) C/H 34	34	34	34	34	34	34
8	83	(3) C/L 56	(3) C/L 45	(1) C/L 37	(1) C/H 37	(1) C/H 37	(1) C/H 37	37	37
9	60	(2) C/L 60	56	45	45	45	(4) C/E 45	(1) C/L 45	45
10	45	69	(2) C/L 60	56	56	56	(3) C/L 56	45	45
11	37	83	69	60	(2) C/H 60	(2) C/H 60	60	56	56
12	22		83	69	(3) C/H 69	69	(2) C/L 69	60	60
13	78			83	83	(3) C/H 78	78	69	69
14	96					(4) C/H 83	83	78	78
15	45						96	83	83
16	17							96	96

(C/H = Compare High; C/L = Compare Low; C/E = Compare Equal. Arrows indicate direction of shift.)

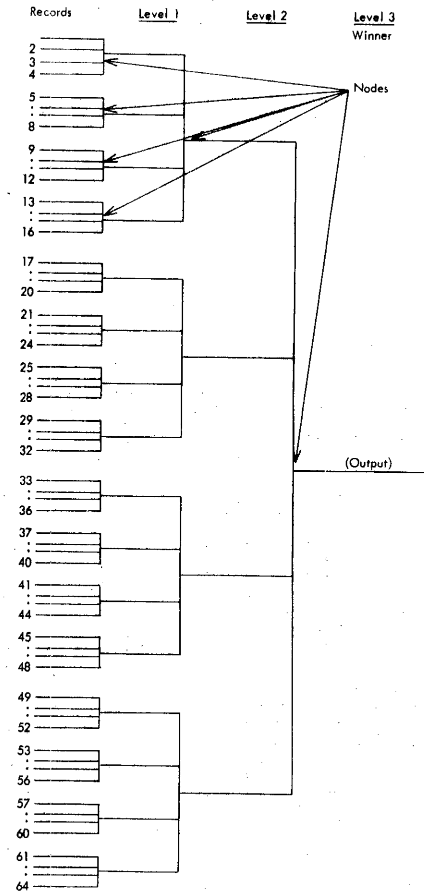
Example 2. Binary insertion



Example 3. Replacement selection (G=4)



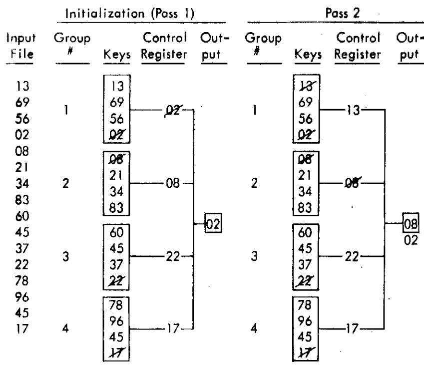
Example 4. Replacement selection, possible organization for G = 12



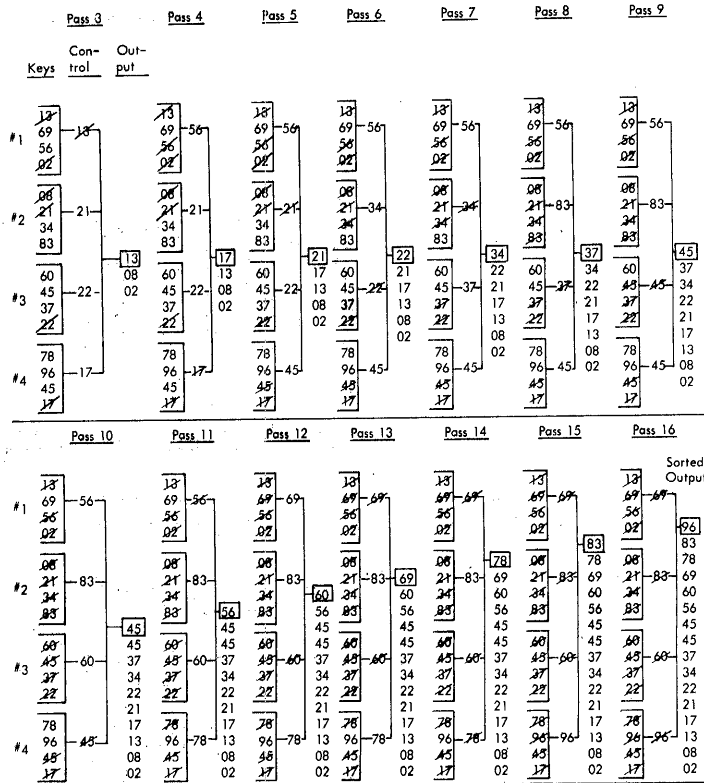
Example 5. Replacement selection, possible organization for G = 64

Input File	End Pass 1	End Pass 2	End Pass 3	End Pass 4	End Pass 5	End Pass 6
13	13	13	99	99	99	99
69	69	69	69	69	69	99
56	56	56	56	56	99	99
02	99	99	99	99	99	99
08	08	99	99	99	99	99
21	21	21	21	99	99	99
Output:	02	08	13	21	56	69

Example 6. Linear selection



Example 7. Quadratic selection, initialization and pass 2



Example 7. Quadratic selection (continued), passes 3 through 16

	Tape A	Tape B	Tape C	Tape D
Initial String Distribution:	13	20	24	0
After 3-way merge of 13 strings	0	7	11	13
After 3-way merge of 7 strings	7	0	4	6
After 3-way merge of 4 strings	3	4	0	2
After 3-way merge of 2 strings	1	2	2	0
After 3-way merge of 1 string	0	1	1	1
After final 3-way merge	1	0	0	0

Note: Underlined entries represent output tapes.

Example 8. Read-forward polyphase merge with four tapes

Number of Partial Passes	Merge Tapes I	II	III	IV	Total Number of Strings
1	1	1	1	1	4
2	2	2	2	1	7
3	4	4	3	2	13
4	8	7	6	4	25
5	15	14	12	8	49
6	29	27	23	15	94
7	56	52	44	29	181
8	108	100	85	56	349
9	208	193	164	108	673
10	401	372	316	208	1297
11	773	717	609	401	2500
12	1490	1382	1174	773	4819

Table 2. Initial string distribution for five-tape polyphase merge

Three Tapes	Four Tapes	Five Tapes	Six Tapes
1	1	1	1
2	3	4	5
3	5	7	9
5	9	13	17
8	17	25	33
13	31	49	65
21	57	94	129
34	105	181	253
55	193	349	497
89	355	673	977
144	653	1297	1921
233	1201	2500	3777
377	2209	4819	7425
.	.	.	.
.	.	.	.
.	.	.	.
etc.	etc.	etc.	etc.

Table 4. Optimum total number of sequences for polyphase merging

Number of Partial Passes	Merge Tapes I	II	III	Total Number of Strings
1	1	1	1	3
2	2	2	1	5
3	4	3	2	9
4	7	6	4	17
5	13	11	7	31 (see Ex. 30)
6	24	20	13	57 (see Ex. 29)
7	44	37	24	105
8	81	68	44	193
9	149	125	81	355
10	274	230	149	653
11	504	423	274	1201
12	927	778	504	2209

Note 1: Final output tape (0 strings) is not shown

Note 2: As an illustration, the distribution used in example 29 (13, 20, 24) for 57 strings is obtained from the previous line (partial pass 5), as follows:

Tape I = Tape I + Tape II of previous line, or 24 = 13 + 11

Tape II = Tape I + Tape III of previous line, or 20 = 13 + 7

Tape III = Tape I of previous line, or 13 = 13

Table 1. Initial string distribution for four-tape polyphase merge

Number of Partial Passes	Merge Tapes I	II	III	IV	V	Total Number of Strings
1	1	1	1	1	1	5
2	2	2	2	2	1	9
3	4	4	4	3	2	17
4	8	8	7	6	4	33
5	16	15	14	12	8	65
6	31	30	28	24	16	129
7	61	59	55	47	31	253
8	120	116	108	92	61	497
9	236	228	212	181	120	977
10	464	448	417	356	236	1921
11	912	881	820	700	464	3777
12	1793	1732	1612	1376	912	7425

Table 3. Initial string distribution for six-tape polyphase merge

	Tape A	Tape B	Tape C	Tape D
Output from internal sort (Initial distribution: 7, 6, 4, 0)	(7) Asc. D A D A D A*	(6) Desc. A D A D A*	(4) Desc. A D A*	0
After 3-way merge of 4 strings	(3) A D A	(2) D A	0	(4) D A D A (Output)
After 3-way merge of 2 strings	(1) A	0	(2) D A (Output)	(2) D A
After 3-way merge of 1 string	0	D (Output)	D	D
Final 3-way merge of 1 string	A (Output)	0	0	0

* First strings to be processed by read-backward merge.

Example 9. Read-backward polyphase merge of 17 strings

	Tape A	Tape B	Tape C	Tape D	Tape E
Output from internal sort	(7) A D A D A D A*	(8) D A D A D A D A*	(6) D A D A D A*	(4) D A D A*	0
After 4-way merge of 4 strings	(3) A D A	(4) D A D A	(2) D A	0	(4) D A D A
After 4-way merge of 2 strings	(1) A	(2) D A	0	(2) D A	(2) D A
After 4-way merge of 1 string	0	D	D	D	D
Final 4-way merge of 1 string	A	0	0	0	0

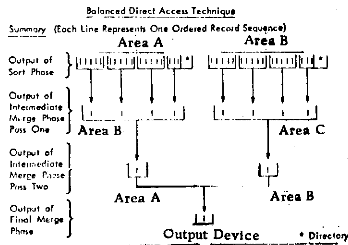
Note: Underlined entries represent output for each merge.

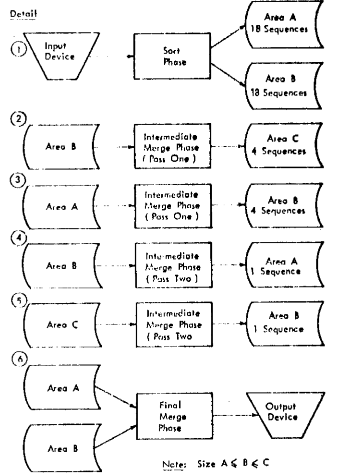
Example 10. Read-backward polyphase merge of 25 strings

Step	Tape A	Tape B	Tape C	Tape D	Operation Performed
1	1	1	1	0	After first internal sort (3 strings generated)
2	0	0	0	3	After first 3-way merge (3 internal strings merged into 1 tape sequence)
3	0	1	1	3/1	After next internal sort (6 internal strings contained in 4 tape sequences)
4	3	0	0	3	After next 3-way merge (6 internal sort strings contained in 2 tape sequences)
5	3/1	0	1	3/1	After next internal sort (9 internal sort strings contained in 5 tape sequences)
6	3	3	0	3	After next 3-way merge (9 internal sort strings contained in 3 tape sequences
7	0	0	9	0	After next 3-way merge (9 internal sort strings merged into 1 tape sequence)
8	1	1	9/1	0	After next internal sort (3 more strings generated)
9	0	0	9	3	After next merge (12 internal sort strings contained on 2 tape sequences)
10	1	0	9/1	3/1	After next internal sort
11	0	3	9	3	After next 3-way merge
12	1	3/1	9	3/1	After next internal sort
13	0	3	9/3	3	After next 3-way merge
14	9	0	9	0	After next 3-way merge (18 internal sort strings contained in 2 tape sequences)
15	9/1	0	9/1	1	After next internal sort (3 more strings generated)
16	9	3	9	0	After next 3-way merge
17	9/1	3/1	9	1	After next internal sort
18	9	3	9/3	0	After next 3-way merge
19	9	3/1	9/3/1	1	After next internal sort
20	9/3	3	9/3	0	After next 3-way merge
21	9	0	9	9	After next 3-way merge (27 internal sort strings contained in 3 tape sequences)
22	0	27	0	0	After final 3-way merge (all 27 internal sort strings merged into 1 tape sequence).

Note: Numbers refer to the number of internal sort strings contained in each tape sequence. Underlined entries are output tapes for that merge. The slash symbol (/) represents a tape mark that separates new sequences from previously merged output.

Example 11. Oscillating sort of 27 strings with four tapes





Example 12. Balanced direct acces sequence-distribution technique

CYLINDER 1	CYLINDER 2	CYLINDER 3	CYLINDER 4
A. Block Organization for First Merge Pass (Area 1)			
S_1B_1	S_1B_6	S_3B_1	S_3B_6
S_2B_1	S_2B_6	S_4B_1	S_4B_6
S_1B_2	S_1B_7	S_3B_2	S_3B_7
S_2B_2	S_2B_7	S_4B_2	S_4B_7
S_1B_3	S_1B_8	S_3B_3	S_3B_8
S_2B_3	S_2B_8	S_4B_3	S_4B_8
S_1B_4	S_1B_9	S_3B_4	S_3B_9
S_2B_4	S_2B_9	S_4B_4	S_4B_9
S_1B_5	S_1B_{10}	S_3B_5	S_3B_{10}
S_2B_5	S_2B_{10}	S_4B_5	S_4B_{10}
B. Input to Second (Final) Merge Pass (Area 2)			
S_1B_1	S_1B_6	S_1B_{11}	S_1B_{16}
S_2B_1	S_2B_6	S_2B_{11}	S_2B_{16}
S_1B_2	S_1B_7	S_1B_{12}	S_1B_{17}
S_2B_2	S_2B_7	S_2B_{12}	S_2B_{17}
S_1B_3	S_1B_8	S_1B_{13}	S_1B_{18}
S_2B_3	S_2B_8	S_2B_{13}	S_2B_{18}
S_1B_4	S_1B_9	S_1B_{14}	S_1B_{19}
S_2B_4	S_2B_9	S_2B_{14}	S_2B_{19}
S_1B_5	S_1B_{10}	S_1B_{15}	S_1B_{20}
S_2B_5	S_2B_{10}	S_2B_{15}	S_2B_{20}

Example 13. Two-way merge on disks with string interleaving

Operation Performed	5000-Record File		10,000-Record File	
	Record Sort	Tag Sort	Record Sort	Tag Sort
A. In Accordance with Sorting Phases				
Phase 1 (Internal Sort)	35	13	69	35
Phase 2 (Merge)	11	--	25	12
Phase 3 (Retrieve)	--	308	--	940
Total Sort Time	46	321	94	987
B. In Accordance with File Operations				
Position Heads	3	215	7	750
Rotational Delay	14	89	29	187
Data Transfer	29	17	48	50
Total Sort Time	46	321	94	987

Example 14. Sorting time (in seconds) for record and tag sorts with an IBM 1301 disk file

Total No. of Tapes	Effective Power of Merge (Reduction Factor)		
	Balanced T/2	Polyphase	Oscillating (T - 2)
3	1.5	1.95	---
4	2	2.68 (2.7)	2
5	2.5	3.19 (3.3)	3
6	3	3.51 (3.7)	4
7	3.5	3.72 (4.0)	5
8	4	3.84 (4.2)	6
9	4.5	3.91	7
10	5	3.95	8
11	5.5	3.97	9
..	..	...	.
..	..	...	.
20	10		
21	10.5		

*Note: Figures for polyphase merges assume perfect initial string distributions and an infinite number of strings. The figures in parentheses indicate the reduction factors in actual situations for a practical number of strings.

Table 5. Effective power of the merge (reduction factor) for tape sorting techniques

Example 15. Polyphase data pass determination (65 strings and 6 tapes)

Operation	Tape 1	Tape 2	Tape 3	Tape 4	Tape 5	Tape 6	String Passes
Initial String Distribution	8 (1)	12 (1)	14 (1)	15 (1)	16 (1)	0	(65 strings initially)
End of first merge	0	4 (1)	6 (1)	7 (1)	8 (1)	8 (5) *	40
End of second merge	4 (9) *	0	2 (1)	3 (1)	4 (1)	4 (5)	36
End of third merge	2 (9)	2 (17) *	0	1 (1)	2 (1)	2 (5)	34
End of fourth merge	1 (9)	1 (17)	1 (33) *	0	1 (1)	1 (5)	33
End of fifth merge	0	0	0	1 (65) *	0	0	65
					Total Strings		208

* Indicates output tape during each merge. Note: The number of original strings contained in each resulting string is shown in parentheses.

Balanced		Polyphase		Oscillating	
Number Strings	Number Passes	Number Strings	Number Passes	Number Strings	Number Passes
T = 4: 3 - way polyphase merge; 2 - way balanced & oscillating merges					
1-2	1	1-3	1	1-2	1
3-4	2	4-5	1.5-1.6	3-4	2
5-8	3	6-9	2 -2.2	5-8	3
9-16	4	10-17	2.4-2.8	Same as for	
17-32	5	18-31	3.0-3.5	Balanced Merge	
33-64	6	32-57	3.6-4	"	"
65-128	7	58-105	4.2-4.7	"	"
129-256	8	106-193	4.85-5.3	"	"
		194-355	5.4-5.9		
T = 5: 4 - way polyphase merge; 3 - way oscillating merge					
---	-	1-4	1	1-3	1
		5-7	1.4-1.6	4-9	2
		8-13	1.9-2.1		
		14-25	2.3-2.7	10-27	3
		26-49	2.8-3.3	28-81	4
		50-94	3.4-3.8		
		95-181	3.9-4.4	82-243	5
T = 6: 5 - way polyphase merge; 4 - way oscillating; 3 - way balanced merge					
1-3	1	1-5	1	1-4	1
4-9	2	6-9	1.3-1.6	5-16	2
10-27	3	10-17	1.8-2.1		
28-81	4	18-33	2.2-2.7	17-64	3
		34-65	2.7-3.2		
82-243	5	66-129	3.2-3.7	65-256	4
		130-253	3.8-4.3		
T = 8: 7 - way polyphase merge; 6 - way oscillating; 4 - way balanced merge					
1-4	1	1-7	1	1-6	1
5-16	2	8-13	1.3-1.6	7-36	2
		14-25	1.7-2.1		
17-64	3	26-49	2.2-2.6	37-216	3
		50-97	2.6-3.1		
65-256	4	98-193	3.1-3.7	217-1296	4
T = 10: 9 - way polyphase merge; 8 - way oscillating; 5 - way balanced merge					
1-5	1	1-9	1	1-8	1
6-25	2	10-17	1.2-1.5	9-64	2
		18-33	1.7-2.1		
		34-65	2.1-2.6		
26-125	3	66-129	2.6-3.1	65-512	3
126-625	4	130-257	3.1-3.6		
		258-513	3.6-4.1		

Table 6. Number of data passes for tape sorting techniques

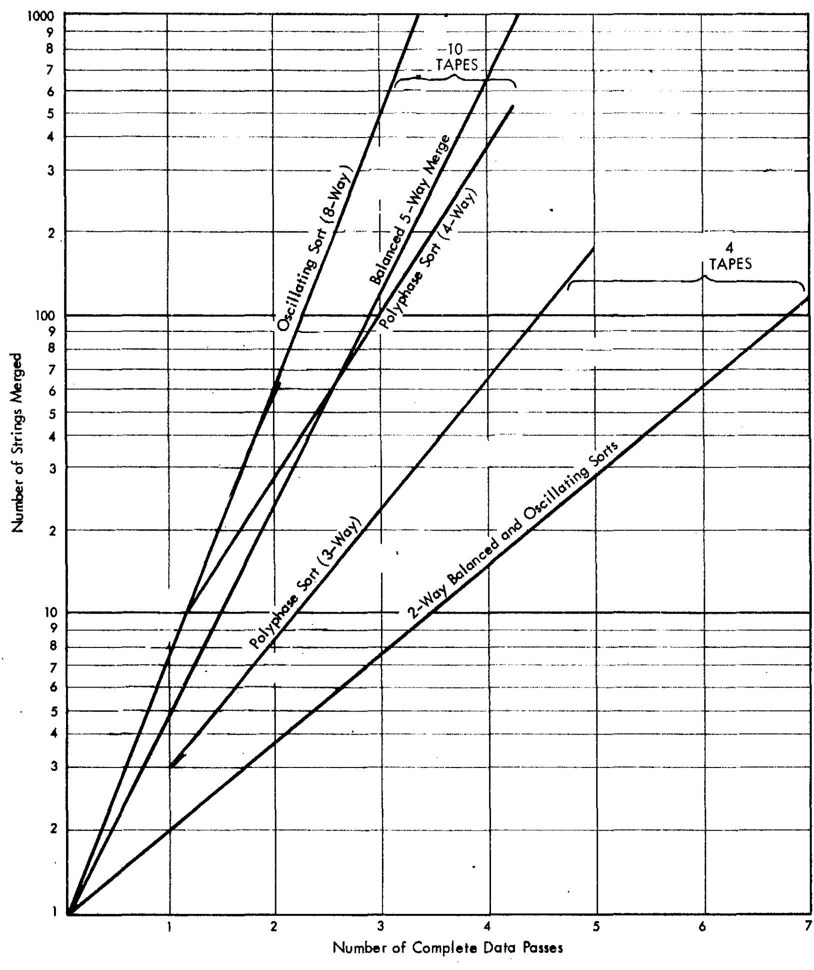


Figure 2. Comparison of tape sorting techniques for T = 4 and T = 10