SCIENTIFIC AND BUSINESS APPLICATIONS

A High-Speed Sorting Procedure

D. L. SHELL, *General Electric Company, Cincinnati, Ohio*

There are a number of methods that have been used for sorting purposes in various machine programs from time to time. Most of these methods are reviewed by Harold Seward [1] in his thesis. One tacit assumption runs through his entire discussion of internal sorting procedures, namely, that the internal memory is relatively small. In other words, the number of items to be sorted is so large that they cannot possibly all fit into the memory at one time.

The methods of internal sorting which he discusses are sorting by:

1) Finding the smallest.
2) Interchanging pairs.
3) Sifting.
4) Partial sort.
5) Merging pairs.
6) Floating decimal sort.

The first four methods all require a time proportional to n^2, where n is the number of items being sorted. The time for the fifth method is proportional to $n(\ln n)$. The time for the sixth method is proportional to $n(\ln r)$, where r is the largest number to be used in a key.

As pointed out in Seward's paper, one would normally choose either method five or six for a rapid internal sort, especially if n is to be very large. The chief drawback of these two methods, however, is the fact that they require twice as much storage as the other four methods.

The advent of very large high-speed random access memories changes the picture relative to sorting somewhat. It is now possible to have a very large number of items to be sorted in memory all at one time. It is highly desirable, therefore, to have a method with the speed characteristics of the merging by pairs and the space characteristics of sifting. If such a method were available it would be possible to sort twice as many items at one time in the machine and still do it at a reasonably high speed.

Such a method is outlined in this paper. The idea is, in fact, to combine some of the properties of merging with some of the properties of sifting. The method is most easily described by reference to the block diagram, figure 1. Suppose we are given a sequence of elements f_i to be sorted with keys r_i, $i = 1, 2, \cdots, n$. One begins by dividing the set of elements into $n/2$ subsets. As can be seen, there will be two elements in each subset, with the possible exception of one which may have three. Each of these subsets is then sorted. It should be noted that each subset of two elements is so placed in the total list that they are separated by approximately $n/2$ places. Thus,

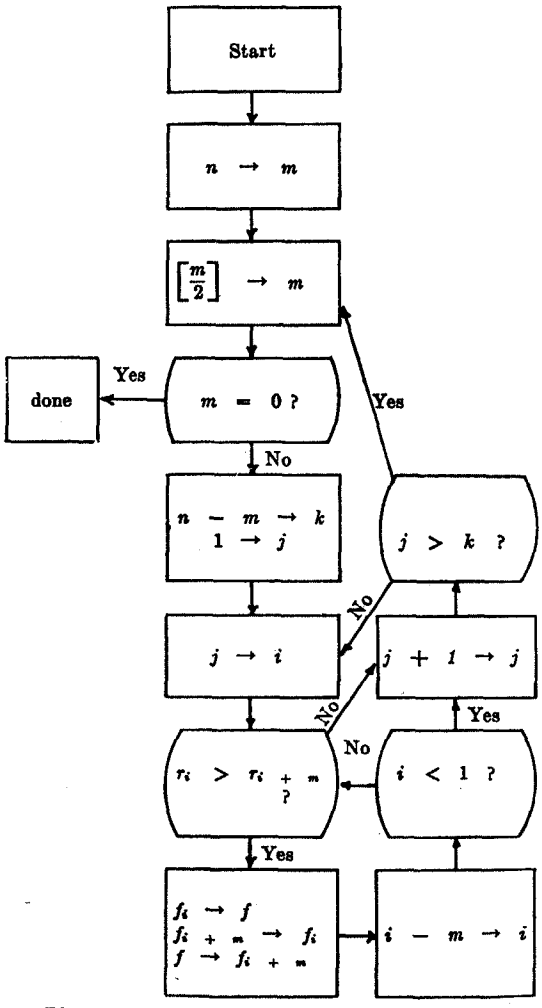

FIG. 1. Block diagram to sort a set of elements f_i with keys r_i, $i = 1, 2, \cdots, n$.

TABLE 1

1	2	3	4	5	6	7	8
3		1				1	1
11	7	7	4		3	3	2
6		6	2			2	3
4		4	7	3	4	4	4
9	2	2	6			5	5
5	1	3		7		7	6
7	11	11	10		5	6	7
8		8				8	8
10		10	11	5	10	10	9
2	9	9				9	10
1	5	5		11		11	11

TABLE 2

Summary of running times

Number of Items	Number of Runs	Average Time (Seconds)
500	100	1.03
1000	50	2.36
2500	12	7.85
5000	12	18.2
10000	13	42.2
20000	6	96.8

if the two elements are out of sort and must be interchanged they will move a distance of $n/2$ places in the total list. Once this pass is completed the total set is now divided into approximately $n/4$ subsets of elements. This, in effect, merges two of the original subsets into one of the new subsets. The merging is not done according to the key, but rather is done by first taking one element from the first subset and one from the second, then another from the first and another from the second, etc. Each of these subsets is then sorted by sifting.

The logic of this whole procedure is outlined in the block diagram, figure 1. The notation of the block diagram is as follows:

$\rightarrow$ means "replaces".

$[m/2]$ means the largest integer less than or equal to

Table 1 illustrates an example sorted by this method. Column 1 is the original arrangement of 11 items. Column 3 is the arrangement after the first complete pass, i.e., with $m = 5$. On the next pass $m = 2$, and column 7 is the result. On the final pass $m = 1$, and the elements are in complete sort.

This particular method requires a negligible amount of storage space in addition to that occupied by the list of items. In addition, it operates at fairly high speed. Thus

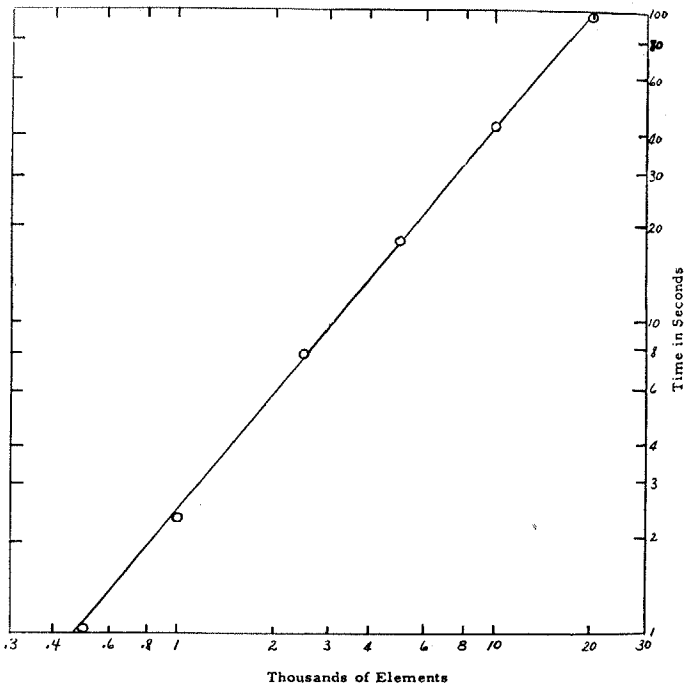

FIG. 2. Sort times for sequences of one-word elements. (Each element is its own key.)

far, an analytical determination of the expected speed has eluded the writer. However, experimental use has established its speed characteristics. It appears that the time required to sort n elements is proportional to $n^{1.226}$ At least this is true for one-word items which are their own key.

Table 2 shows a summary of average times for sorting numbers of one-word elements which were originally in random order. In this case, $r_i = f_i$. The listing in appendix I is a 704 program for this particular case. The graph in figure 2 is a plot of the time required to sort a random sequence of n elements with the above program.

This sorting method is useful in machines with a large high-speed random access memory. The time requirement is only slightly longer than that for merging pairs—but this method can be used to sort twice as many elements in the same memory space. In many cases, the little extra time for internal sorting will be more than compensated by the diminished use of slow memory, such as tapes.

REFERENCES

1. SEWARD, HAROLD H., Information sorting in the application of electronic digital computers to business application (ASTIA 35462).
2. FRIEND, E. H. Sorting on electronic computer systems, *J. Assoc. Comp. Mach.* 3 (1956), 134.

APPENDIX I. SAMPLE PROGRAM

This is a program for sorting n one-word elements using the block diagram on page 30. Each one-word element is its own key.

```
*       DL SHELL
*  SORT PROGRAM, ALGEBRAIC
```

```
*  CALLING SEQUENCE
*       SXJ  SORT,4
*       ZER  IA,0,N[a]
*         RETURN
*
SORT  CAL  1,4
      SXD  IA,4
      SXD  B1,1
      STA  S
      STA  A
      STA  Y-1
      STA  IA
      STD  M
      CPL[b]
      ADD  DEC1
      STD  B
C     CLA   M
      ARS   1
      STD  M
      CLA  M
      ZEJ  E
      STD  W
      STD  X
      ADD  B
      STD  K
      CLA  M
      ARS  18
      ADD  IA
      STA  U
      STA  Y
      LXA  M,5
U     CLA  --,4
S     SGA  --,4
      UNJ  V
      UNJ  V
      LDQ  --,4
Y     STQ  --,4
A     STO  --,4
W     RXJ  *+1,4,--
X     HXJ  S,4,--
      LXJ  S,4,0
V     LXA  M,4
      RXJ  *+1,5,-1
K     HXJ  U,4,--
      UNJ  C
E     LXD  IA,4
      LXD  B1,1
      UNJ  2,4
DEC1  ZER  0,0,1
IA    ZER  --,0,--
M     ZER  0,0,--
B     ZER  --,0,--
B1    ZER  0,0,--
*       LAST CARD OF SORT PROGRAM
```

[a] IA is address of first element; N is the number of elements to sort.

[b] Twos complement of N + 1.