QUAESTIONES INFORMATICAE

Vol. 1 No. 2

September, 1979

Quaestiones Informaticae

An official publication of the Computer Society of South Africa
'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika

Editors: Dr. D. S. Henderson,
Vice Chancellor, Rhodes University, Grahamstown, 6140, South Africa.
Prof. M. H. Williams,
Department of Computer Science and Applied Maths,
Rhodes University, Grahamstown, 6140, South Africa.

Editorial Advisory Board

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R2	$3	£1.50
Institutions	R4	$6	£3.00

Quaestiones Informaticae is prepared for publication by SYSTEMS PUBLISHERS (PTY) LTD

for the Computer Society of South Africa.

An Algorithm for Merging Disk Files in Place

P.P. Roets

National Research Institute for Mathematical Sciences, CSIR, PRETORIA, South Africa

Abstract

An algorithm is presented for sorting a random access file in place. The algorithm is unique in the sense that no auxiliary storage, apart from a number of buffers in the machine's high-speed memory, is required. In addition, the algorithm makes active use of the random access capabilities of rotating magnetic media and cannot be converted to operate on sequential files. The run time is known to be of O(n log n) under certain circumstances. An example also shows that this run time can deteriorate to O(n_*), which is probably also the worst-case run time.

1. Introduction

The algorithm to be described has the unique property that it requires no auxiliary files during execution. This could be useful on (small) computer systems with little or no free file space. Although, in general, the speed of sorting will be slower than that of the commonly used external sort methods, the algorithm will perform well in any environment where the user is forced to execute the sort using a single disk drive.

The underlying principle of the algorithm is to use space on the file, vacated by those records currently in the high-speed memory of the computer, as additional work space. Careful organization of the records temporarily stored in this area obviates unnecessary reading and writing of records to this area of the file. The algorithm merges a number of runs into a final sorted file. At any point in time two adjacent runs are processed to form an output run that replaces the original runs in the file.

To understand the operation of the merge algorithm it is necessary to describe (here in 'pseudo' ALGOL) the environment in which it operates. The following are assumed:

(1) There are C + 2 buffers available in memory. These buffers are used by both the MERGE and GETSTRING procedures. C of these buffers are grouped together as a structured global variable BUFARRAY. The remaining two are named BUF2 and TEMP. The detailed handling of these buffers will not be described. A convenient data structure for BUFARRAY would be a tree structure for use by GETSTRING and a circular list for use by MERGE. For the latter algorithm a number of functions that operate on BUFARRAY need to be defined. 'Set-empty' will initialize the structure, 'Full' will test for the availability of space, and 'Empty' will be TRUE when no records are left. Outputting a record will remove the record from the structure, and reading or moving a record into BUFARRAY will add it to the structure. Finally, the function Before (BUFARRAY ,b) will be TRUE when the first element of BUFARRAY should precede the record b in the output run.

(2) The function S( ) calculates the number of records in the file to be sorted, or at least estimates the maximum possible number of records.

(3) Several procedures will be left undefined. These are mostly concerned with the random input and output of records. The function of these procedures should be clear from the descriptive names that were chosen.

(4) In addition, the procedure GETSTRING will be left undefined. Any internal sorting algorithm could be employed, the only assumption being that GETSTRING will create initial runs of length greater than or equal to C. In particular, 'replacement selection' [1] could be employed. This will generate runs with expected length 2C.

This procedure communicates its results to the main procedure through the single parameter INP. INP is initialized to point to the first record in the file and is advanced during the execution of GETSTRING to point to the next unprocessed record. When GETSTRING has reached the end of the file, subsequent calls will be ignored and INP will remain pointing to the first non-existent record beyond the end of the file.

```
Begin
Integer INP; /* Pointer to next record to be processed
                by GETSTRING */
Procedure SORT (LEVEL);
Integer LEVEL;
Begin
/* The B's and E's point to the first record and the record following
the run*/
Integer B1, E1, B2, E2;
If LEVEL > 0 then
Begin B1 := INP: SORT (LEVEL-1); E1 := INP;
      B2 := INP; SORT (LEVEL-1); E2 := INP;
      MERGE (B1, E1, B2, E2, B1, TRUE);
end
                        else GETSTRING
end SORT;
INP := θ;
SORT ( Entier ( Log2 ( S/C ) ) );
end;
```

Figure 1. Sort procedure.

The execution environment for the merge procedure is described by the sort procedure in Fig. 1.

Clearly, this algorithm builds a binary tree and executes MERGE at each node, combining in a single run the runs represented by the nodes at a greater depth in the tree. This single run then forms one of the inputs for the next level. At the leaves of the tree, the starting runs are created by a call to GETSTRING. The algorithm terminates when a single run is created at the root of the tree.

2. The Merge Algorithm

The MERGE procedure is described in Fig. 2.
This procedure divides the sequence of records on which it must operate into three runs, IR1, TR and IR2. Initially IR1 and IR2 contain the two input runs of sorted records defined by the record pointer pairs (I1,E1) and (I2,E2) respectively. The temporary run TR, defined by (IS,ES), is empty initially and is located after the first run. The position where the next merged record must be written is defined by OUTP.
At the lowest level call (INIT = TRUE) the first record from the second run is read into BUF2, thus making space available for the growth of TR. Upon a recursive call to the procedure, the memory buffers BUFARRAY and BUF2 will already be filled.
Records from the first run will all be read into TEMP, thus making output space available, and then transferred to BUFARRAY, if space is available, or otherwise stored in TR (as an extension of BUFARRAY).
When there is no more space in which the output run may grow, or in which records read from IR1, in order to make output space available, can be stored, special steps have to be taken before the process can continue.

```
Procedure MERGE (I1, E1, I2, E2, OUTP, INIT);
Value I1, E1, I2, E2, OUTP;
Integer I1, E1, I2, E2, OUTP;
Boolean INIT:
Begin
Integer IS,ES:
Boolean STR1,STR2,TSTR;
Procedure WFIRST; /* output element originally from first run */
Begin
Write (first record from BUFARRAY at OUTP);
OUTP := OUTP + 1;
If IS < ES then
Begin
Read (read record at IS into BUFARRAY);
IS := IS + 1;
end             else
If TSTR then
Begin Move (record in TEMP into BUFARRAY);
    TSTR := FALSE;
end             else
If Empty (BUFARRAY) then STR1 := FALSE;
end WFIRST;
Procedure WLAST; /* write record originally from second run */
Begin
Write (record in BUF2 at OUTP)
OUTP := OUTP + 1;
If 12 < E2 then
Begin
Read (record at I2 into BUF2);
 I2 := I2 + 1;
end
                    else STR2 := FALSE;
end WLAST;
If 12 = E2 then
/* exit because the second run is empty */
                                else
Begin
TSTR := FALSE;
IS := ES := I2;
If INIT then
Begin Set_Empty (BUFARRAY);
      STR1 := FALSE;
      Read (record at I2 into BUF2);
      I2 := I2 + 1;
      STR2 := TRUE:
end
       else STR1 := STR2 := TRUE;
While STR1 or STR2 do
Begin /*make space for next output record*/
If OUTP = IS then
Begin If IS = ES then IS := ES := ES + 1
                                else
Begin I1 := IS; E1 := ES; IS := ES; end;
end;
If I1 < E1 and OUTP = I1 then
BeginIf IS < > ES and ES = I2 then
Begin MERGE (IS, ES, I2, E2, E1, FALSE);
      IS := E1; ES := E1; I2 := E1;
      Read (record at I2 into BUF2);
      I2 := I2 + 1;
      STR2 := TRUE;
      Set empty (BUFARRAY);
      STR1 := FALSE;
end;

Read (record at I1 into TEMP);
I1 := I1 + 1;
If not Full (BUFARRAY) then
Begin Move (TEMP into BUFARRAY);
STR1 := TRUE;
end
                                else
TSTR := TRUE;
end;
If STR1 and STR2 then /* output next record*/
Begin If Before (BUFARRAY, BUF2) then WFIRST
else WLAST
end       else
If STR1 then WFIRST else WLAST;
If TSTR then
Begin Write (record from TEMP at ES);
      ES := ES + 1;
end;
end;
end;
end MERGE;
```

Figure 2. Merge Algorithm

These critical situations that may occur are the following.
Case 1: The first run has been exhausted and the output pointer reaches the start of TR. If TR is empty, its starting position is pushed backwards one record, into the unused sequence of C + 1 record positions between TR and IR2. Should TR be non-empty, it is renamed IR1 and a new TR is created between it and IR2.
Case 2: The first run is non-empty and TR bumps into IR2. In this case a recursive call is made to MERGE with input TR and the remaining part of IR2. Upon return from this call, the initial conditions again exist, the input being the remaining part of IR1 and the newly merged run.
It should be noted that if the second or both runs are empty, the algorithm effectively ignores the call. The parameters are chosen in such a way that the first run will be returned as the result of the MERGE. This property guarantees correct operation of the SORT algorithm on an incomplete binary tree.

3. Run Time Estimation

To evaluate the performance of an external sort algorithm it is usual to choose the count of either read, write or seek operations as a measure of the complexity of the algorithm. Since this algorithm will reposition the file for every operation on the file and execute exactly the same number of read and write operations, any one of these three measures can be chosen.
The best run time for sorting a file of n records is O(n log n). This is easily shown to be the case when no use is made of TR, and GETSTRING never generates a null string. Each record will be read (or written) exactly once at each level of the merge tree, giving a total number of operations equal to n*(number of levels). It can be noted that when replacement selection is used by GETSTRING, on a sorted file, all the MERGE operations will be trivial, since only one run is generated. This case requires only O(n) operations. This is, of course, a property of the particular algorithm for GETSTRING, and does not reflect on the best run time attainable by MERGE.

The worst-case run time is much more difficult to calculate, and is as yet unknown. A class of examples can be constructed such that each instance requires O(n * n) operations. Assume that the file consists of $n = C * 2^L$ records and that GETSTRING creates runs of C records. If at each level the runs to be merged are such that a recursive merge repeatedly occurs at the first possible opportunity, there will be O(n*n) operations. Two runs which have this property can be constructed as follows.

Let the runs be $A_1A_2A_3 \ldots A_m$ and $B_1B_2B_3 \ldots B_m$.
If $B_1,B_2,B_3 \ldots B_C < A_1$ BUFARRAY will be filled by successive iterations of the inner loop of MERGE. Should also $B_{C+1} < A_1$ then one record will be pushed onto TR. It is now necessary that $A_1,A_2 \ldots A_{c+1} < B_{C+2}$ for TR to be shifted up against IR2, and a recursive merge executed.
Let R(1,m) indicate the number of operations to merge runs of length 1 and m. For the instance described above we must calculate

$$R(mC,mC) = R(C * 2^i, C * 2^i)$$

for the cost of merging two runs at level i+1. Since the runs are assumed to have the structure described above, we can write:

$$R(mC,mC) = 2mC + \text{cost of recursive merges.}$$

The number of such recursive merges is approximately m-1, giving

$$R(m*C,m*C) = 2mC + \sum_{j=1}^{m-1} R(C+1,mC-j(C+1)).$$

In the case R(C+1,x) no recursive merge is possible and the run time has the value C + 1 + x. This gives

$$R(mC,mC) = 2mC + (m-1)(C+1) + mC(m-1) - m(m-1)(C+1)/2.$$

The two important terms on the right-hand side are

$$m*mC/2 - m*m/2.$$

Calculating their effect on the total sort operation time we get

$$R=\sum_{i=1}^{L} 2^{L-1} (2^{2i-2}C/2 - 2^{2i-2}/2).$$

Noting that L = log (n/C), this reduces to

$$R = n*n/4C - n*n/4C*C + n/4C - n/4$$

This may not be the worst-case run time for the algorithm, since it was assumed that the recursive merges occur at the first possible opportunity. Should this condition be relaxed, it might be possible for merge recursion to develop to a greater depth.

4. Conclusions

Two aspects of the algorithm were not addressed.
Firstly, the algorithm can be made into a stable sorting algorithm, i.e. one which retains the original sequence of records with identical keys. It is only necessary to ensure that GETSTRING is stable and that the function 'Before' selects from BUFARRAY when it compares records with identical keys. A careful study of the algorithm will show that this increases the likelihood of a recursive merge occurring.
It should also be clear that the algorithm does not preclude the use of 'blocking' on the file to be sorted. The buffers will have to be large enough to contain a block of records. An additional buffer would be required for building the output block. It is in this respect that the algorithm differs from any internal sorting algorithm modified to access disk devices randomly.
The fact that the algorithm is only executing a two-way merge will probably restrict its use in any environment where sufficient temporary storage is available.

References

[1] KNUTH, D.E. (1973). The Art of computer programming, vol. 3, *Sorting and Searching,* Addison-Wesley.

Notes for Contributors

The purpose of this Journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles, exploratory articles or articles of general interest to readers of the Journal. The preferred languages of the Journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be in double-spaced typing on one side only of A 4 paper and submitted to Dr. D. S. Henderson or Prof. M. H. Williams at

Rhodes University
Grahamstown 6140
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. The original ribbon copy of the typed manuscript should be submitted. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name, and the affiliation and address. Each paper must be accompanied by a summary of less than 200 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Illustrations and tables should not be included in the text, although the author should indicate the desired location of each in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Illustrations should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the Author's name and figure number. Original line drawings (not photoprints) should be submitted and should include all relevant details. Drawings, etc., should be about twice the final size required and lettering must be clear and "open" and sufficiently large to permit the necessary reduction of size in block-making.

Where photographs are submitted, glossy bromide prints are required. If words or numbers are to appear on a photograph, two prints should be sent, the lettering being clearly indicated on one print only. Computer programs or output should be given on clear original printouts and preferably not on lined paper so that they can be reproduced photographically.

Figure legends should be typed on a separate sheet and placed at the end of the manuscript.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters between the letter O and zero; between the letter l, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetical order of author's name, and cited in the text by number in square brackets. Journal references should be arranged thus:

1. ASHCROFT, E. and MANNA, Z. (1972). The Translation of 'GOTO' Programs to 'WHILE' Programs, in *Proceedings of IFIP Congress 71,* North-Holland, Amsterdam, 250-255.
2. BÖHM, C. and JACOPINI, G. (1966). Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM,* **9,** 366-371.
3. GINSBURG, S. (1966). *Mathematical Theory of Context-free Languages,* McGraw Hill, New York.

Proofs and reprints

Galley proofs will be sent to the author to ensure that the papers have been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged against the author. Corrected galley proofs, together with the original typescript, must be returned to the editor within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Fifty reprints of each article will be supplied free of charge. Additional copies may be purchased on a reprint order form which will accompany the proofs.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.

Hierdie notas is ook in Afrikaans verkrygbaar.

Questiones Informaticae

Partial proceedings of the first South African Computer Symposium on Research in Theory, Software, Hardware, organised by The Research Symposium Organising Committee of The Computer Society of South Africa. 4 & 5 September 1979, Pretoria.

Contents/Inhoud