18.4 Sorting Data Sets too Large to Fit in Memory

A. Purpose

Sort a set of data that is too large to be contained in main memory. The structure of the records, and the criteria for ordering them, are determined by the calling program.

B. Usage

B.1 Program Prototype

INTEGER N, L(≥N), OPTION, OUTFIL
EXTERNAL DATAOP

Assign values to N and OPTION. Require N ≥ 4.

CALL EXSORT (DATAOP, N, L, OPTION, OUTFIL)

B.2 Argument Definitions

DATAOP [in] A SUBROUTINE subprogram that is used to perform operations on the data. DATAOP is referenced by CALL DATAOP (IOP, I1, I2, IFLAG), where all arguments are integer scalars. IOP defines the action to perform, I1 and I2 provide parameters necessary to those actions, and IFLAG returns a status. When I1 or I2 index a record area, we have 1 ≤ I1, I2 ≤ N.

The subroutine DATAOP must have capacity to store at least N ≥ 4 records, and access to four external sequential files, such as tape or disk files. Using the argument IOP, EXSORT requests DATAOP to execute operations such as comparing records in the internal region, and moving records between the internal region and specified external files. These actions are defined below.

IOP **ACTION**

1 Place a datum from the set to be sorted into the record area indexed by I2. I1 is irrelevant. Set the value of IFLAG to zero if a datum is available, or to any nonzero value if there are no more data in the set. If there are no more data in the set, do not modify any record areas since EXSORT assumes they are intact, and may ask DATAOP to output from them using IOP = 6.

2 Write the datum in the record area indexed by I2 onto the intermediate file indexed by I1, 1 ≤ I1 ≤ 4. The value of IFLAG is irrelevant.

3 Place an end-of-sequence mark to be recognized during performance of operation 4 below onto the intermediate file indexed by I1. The values of I2 and IFLAG are irrelevant.

4 Read a datum from the intermediate file indexed by I1 into the record area indexed by I2. Set the value of IFLAG to any nonzero value if the end-of-sequence mark written by operation 3 is detected, else set the value of IFLAG to zero. If the end-of-sequence mark is detected, do not modify any record areas.

5 Rewind the intermediate file indexed by I1. The values of I2 and IFLAG are irrelevant. This operation is the first operation performed on the intermediate files; thus, they should be opened here if they are not already opened.

6 If I1 is zero the next datum from the sorted set is in the record area indexed by I2. At this point you can print the item, save it, or process it in some other way. If I1 is nonzero, all of the records of the sorted set have previously been passed to DATAOP with IOP = 6 and I1 = zero. The last action performed by EXSORT is to invoke DATAOP with IOP = 6 and I1 ≠ zero.

7 Move the datum in the record area indexed by I1 to the record area indexed by I2. The value of IFLAG is irrelevant.

8 Compare the datum in the record area indexed by I1 to the datum in the record area indexed by I2. If the datum in the record area indexed by I1 is to appear in the sorted sequence before the datum in the record area indexed by I2, set the value of IFLAG to −1 or any negative integer; if the order of the records in the areas indexed by I1 and I2 is immaterial, set the value of IFLAG to zero; if the record in the area indexed by I1 is to appear in the sorted sequence after the datum in the record area indexed by I2, set the value of IFLAG to +1 or any positive integer.

EXSORT does not have access to the data. Since DATAOP is a dummy procedure, it may have any name. Its name must appear in an EXTERNAL statement in the calling program unit.

N [in] The maximum number of records DATAOP can hold in its internal space. When the I1 or I2 argument of DATAOP indexes a record, its value will be in the range from 1 to N. We must have N ≥ 4. Larger values for N will tend to give faster sorts.

L() [scratch] An array of length at least N.

OPTION [in] The first stage of sorting a large volume of data consists of sorting parts of it that are as large as can be contained in memory. These parts are then written onto intermediate files. If the first record of a part that has been sorted in main memory could appear in the sorted order after the last record that has been written onto an intermediate file, the newly sorted part will be concatenated onto that sequence. Thus it may happen that all of the data are contained in a single sorted sequence on one intermediate file. This event is called fortuitous completion.

If fortuitous completion occurs and OPTION = 0, EXSORT will set OUTFIL to the index of the intermediate file on which the completely sorted sequence was discovered ($1 \leq$ OUTFIL ≤ 4). If fortuitous completion does not occur and OPTION = 0, EXSORT will set OUTFIL = 0 and return the sorted records by calling DATAOP with IOP = 6. If OPTION is not zero EXSORT will not alter OUTFIL, and the sorted records are always returned by calling DATAOP with IOP = 6, even if fortuitous completion occurs.

To take advantage of fortuitous completion to prevent needless copying of the sorted data, set OPTION = 0, arrange DATAOP to write data on the intermediate files in the same format as they are written on the final output file (when IOP = 6), and allow the consumer of the sorted data to read them from a file specified by a variable that is computed depending on the value of OUTFIL.

OUTFIL [out] See OPTION.

C. Examples and Remarks

The program DREXSORT illustrates the use of EXSORT to sort 10000 randomly generated real numbers. The output should consist of the single line

"EXSORT succeeded using n compares", where n is about 126000.

Stability

A sorting method is said to be *stable* if the original relative order of equal elements is preserved. This subroutine uses a merge sort algorithm, which is not inherently stable. To impose stability, add a sequence number to each record, and include the sequence number as the least significant ordering criterion when DATAOP is called with IOP = 8.

D. Functional Description

The basic strategy of external sorting is to sort as much of the data as possible in main memory. If the data set is larger than can be contained in main memory, write a sorted subset onto a file. Continue writing sorted subsets until the entire set has been sorted into blocks that are ordered, but elements in different blocks might not be ordered. Then merge the sorted blocks until one sorted sequence is produced. There are several strategies for each of these steps.

The number of passes necessary to merge the sorted blocks is proportional to the logarithm of the number of blocks. Since data transfer using files is slower than sorting in memory, one should minimize the number of blocks by making them as large as possible.

Data are sorted using calls to INSRTX which has the same functionality and calling sequence as INSORT (Chapter 18.3), but the COMPAR subprogram has a different specification. It is a SUBROUTINE with four arguments, IOP, I, J and IFLAG as described for DATAOP above. IOP is 8, and I, J and IFLAG are used as described for DATAOP above.

Sorted data are distributed onto two intermediate files, with the file that receives a datum changed whenever the datum to be put onto a file would be less than the preceding datum. If the first datum of a sorted subset is less than any of the last elements already recorded on intermediate files, but the last is greater, the sorted subset is split, with as much as possible concatenated onto one of the sequences under construction on an intermediate file. As a result the number of ordered sequences on the two intermediate files is different by at most one, but there is no bound (other than the size of the data set) on the difference between the numbers of records on the intermediate files. It is also possible that only one sorted sequence will be produced, although the data could not all be contained in memory. See the explanation of the argument OPTION for a discussion of this possibility.

After the data have been sorted into blocks, and distributed onto the two intermediate files, the sequences on the intermediate files are merged repeatedly until only one sequence remains. When more than four sequences remain, the result of merging sequences is written to another intermediate file. When four or fewer sequences remain, the result of merging sequences is passed to DATAOP with IOP = 6. During the merge process, there will generally be two sequences being merged. But when an odd sequence remains at the end of a pass it will be merged with the first two sequences that were produced in that pass, using a three-way merge, before the remaining sequences produced in that pass are merged using two-way merges.

E. Error Procedures and Restrictions

EXSORT neither detects nor reports error conditions. The only limitations are those imposed by the capacity of external storage, and those imposed by INSRTX (see

Section 18.3.E).

Designed and coded by W. V. Snyder, JPL 1974. Adapted for MATH77, February 1990.

F. Supporting Information

The source language is ANSI Fortran 77.

Entry	Required Files
EXSORT	EXSORT, INSTRX, PVEC

DREXSORT

```
c>>    1996-06-24  DREXSORT  Krogh   Added code for conversion to C.
c>>    1995-05-28  DREXSORT  Krogh   Converted SFTRAN to Fortran
c>>    1990-02-09  DREXSORT  Snyder  Initial code.
c
c      Test driver for EXSORT.
c
c      Sort LDATA=10000 random numbers using EXSORT.
c      Check whether they are in order.
c
       integer LENBUF
       parameter (LENBUF=1000)
       integer L(1:LENBUF),OPTION,OUTFIL
       parameter (OPTION=1)
       external DATAOP
       real R(1:LENBUF)
       common /RCOM/ R
c
       call exsort (dataop,LENBUF,l,option,outfil)
       stop
       end

       subroutine DATAOP (IOP, I, J, IFLAG)
       integer LDATA, LENBUF
       parameter (LDATA=10000, LENBUF=1000)
       integer IOP, I, J, IFLAG
       logical ISopen(4)
       save ISOPEN
       integer NCOMP, NDATA
       logical OK
       save NCOMP, NDATA, OK
       real PREV,R(1:LENBUF),SRANU
       save PREV
       external SRANU
       common /RCOM/ R
       data ISOPEN, NCOMP, NDATA, OK, PREV /4*.FALSE., 0, 0, .TRUE., -1./
c++ CODE for .C. is inactive
c%% static float end_of_seq[1] = {-1.0e0};
c%% static char *fname[4]={"scratch1","scratch2","scratch3","scratch4"};
c%% static FILE *fp[4];
c++ END
c
       go to (10, 20, 30, 40, 50, 60, 70, 80), IOP
       return
c       case 1 @ initial input into record J
   10  continue
         ndata = ndata + 1
         if (ndata.le.LDATA) then
            r(j)=sranu()
            iflag=0
         else
            iflag=1
```

```
          end if
          return
c       case 2 @ write scratch from J onto file I
   20   continue
c%%       fwrite( &rcom.r[j-1], sizeof(rcom.r[0]), 1L, fp[i-1]);
          write (i+10) r(j)
          return
c       case 3 @ write end-of-sequence onto file I
   30   continue
c%%       fwrite( end_of_seq, sizeof(rcom.r[0]), 1L, fp[i-1]);
          write (i+10) -1.0E0
          return
c       case 4 @ read scratch into J from file I
   40   continue
c%%       fread( &rcom.r[j-1], sizeof(rcom.r[0]), 1L, fp[i-1]);
          read (i+10) r(j)
          iflag=0
          if (r(j).lt.0.0) iflag=1
          return
c       case 5 @ rewind file I
   50   continue
          if (.not. isopen(i)) then
c%%          fp[i-1] = fopen(fname[i-1], "wb+");
             open (i+10, status='scratch',form='unformatted')
             isopen(i)=.true.
          end if
c%%       rewind( fp[i-1] );
          rewind (i+10)
          return
c       case 6 @ output from record J
   60   continue
          if (i.ne.0) then
             if (ok) then
                print '('' EXSORT succeeded using'',i7,'' compares'')',
     1             ncomp
             else
                print *,'EXSORT failed'
             end if
          else
             if (r(j).lt.prev) ok=.false.
             prev=r(j)
          end if
          return
c       case 7 @ move record I to record J
   70   continue
          r(j)=r(i)
          return
c       case 8 @ iflage I and J
   80   continue
          ncomp=ncomp+1
          if (r(i)-r(j)) 110,120,130
  110     iflag=-1
          return
  120     iflag=0
          return
  130     iflag=+1
          return
c
       end
```