The Program to Sort Data, Data Support, NCAR, Jan 2004

Sort program at Data Support, NCAR (Jenne, 2002, 4 p)

Quadratic Sort (Spangler, 2002, 2 p)

This NCAR sort program is good for quickly sorting large volumes of data.
- It has a flexible way to set up the sort key, which determines the order of the sort output.

Flow chart for merge part of sort (Jenne, Oct 1966)

List and test of blocked (Rptout) sort merge (Jenne, 1966, 8 p)
- This has a 2-way merge. The present one has a 7-way merge.

Test of unblocked sort merge (Jenne, 1966, 8 p)

Information about binary tree sort programs
- None of this code is given here.

The listing of the Jan 2005 sort program is on a CD-ROM

Ready to scan Jan 5, 2004 (25 p), RJ0325

Roy Jenne
Wilbur Spangler
Jan 5, 2004

RJ0325
I started to work at NCAR in Jan 1965. It soon became clear that we needed a sort program to sort grid analyses and to sort observations. A student assistant cobbled up a binary tree sort, but that became too complicated and it was never checked out.

About Aug – Sep 1966 we settled on a quadratic sort design. For each report in memory, a sort key is generated. The sort key is kept together with a word that tells the location of the report in memory; also the length of the report is kept here. The program keeps the sort keys in several small arrays. If only 100 reports will fit into memory there will be 10 short arrays of sort keys for the sort job, each with data for 10 reports. If 900 reports will fit in memory, then there will be 30 arrays of sort keys (30 is square root of 900). To illustrate the process, three arrays are used in Figure 1. First we want to find the sort key with the smallest value and then output the report that it points to. The way to do this is to find the smallest sort key in each array and save it. Then we look at the smallest sort key in each of the ten arrays and find the smallest key for any report that is in memory. Then we output that report that has the smallest key. We read in one more report and put its sort key into the slot just vacated by the key for the report written out. This buffer of sort keys has had a change so we determine a new smallest sort key in this buffer. Then we compare the smallest keys from each of the buffers. That process identifies the next report in memory with the smallest sort key. We output this “smallest key” report, read in another report, and complete the process. We are building a series of reports in the output file that are in the proper sort order.

At some point, all of the sort keys in each buffer of keys will be smaller than the last one that we put onto the output file. We do not output this report onto this string because it would cause the present string to be out of sort. So we close out the first output file (or “string”) and start another output file that will also be in sort from small sort keys to big ones. We keep up this process until all reports are contained in a series of strings that are each in sort. On a big job we may have 100 or more strings of reports. Each string of data is in sort order, but to complete the sort task, we must merge together all of the strings.

THE NUMBER OF STRINGS AND TIMING FOR THE SORT.

There is an advantage to having each sorted string as long as possible. With long strings, there will be fewer strings and then there will be fewer data passes in the merge process. If there are more reports in memory, the strings will be somewhat longer. However, with more reports there will also be more work done in the comparison of sort keys. Now with faster computers, this extra work may not matter too much.

THE MERGE PART OF THE SORT.

We need to change from many strings that are each in sort, to one string of reports that is in the proper sort order. We could do this by merging 2 strings together to create one string in sort. If there were 100 strings, we would first do these merges of pairs to produce 50 strings. Then we
would repeat the process to give 25 strings, then 13, then 7, then 4, 2, 1. Finally we have the one
sorted string of data that we need. In this example, we would need to make seven passes through all
of the data to achieve one string that is in sort.

THE 7-WAY MERGE.

After a few years we (Will Spangler) changed the merge process to a 7-way merge to get a faster
convergence of the merge process.

Consider the example of 100 strings. By merging 7 at once, we can reduce 100 strings to 15 on the
first pass. This reduces to 3 on the second pass, and then only one more pass through all of the data
is needed to complete the sort. In this case, we have to make 3 passes through all of the data,
compared with 7 passes if we use the two-way merge described above.

THE ALLOCATION OF MEMORY FOR THE SORT.

The user needs to declare how much computer memory is available for the sort. The program will
decide how to allocate memory for the sort key buffers and for the actual reports. With too little
memory, then too many strings will be generated. With short reports and too much memory, the
length of the buffers of sort keys will be longer, and a little more time will be taken for the compar-
isons of the sort keys. The sort needs to have a reasonable amount of memory, but not a ton of it.

How much memory do we use? NCAR had the Control Data 7600 from 07/1971 – 04/1983. It was
the big fast computer until the Cray 1A started operations in 07/1977. The 7600 had 65,000 words,
each 60 bits long. We shared the computer with hundreds of other users. When we ran a big sort
job, we usually used about 6,000 to 10,000 words of memory. Modern computers have much more
memory, but we still usually use about the same number of words (64 bits each) of memory.

WHAT DATA DO WE SORT?

We sort either variable length reports written out in the Report Out blocking structure, or we can
sort fixed block reports.

SORT FIXED LENGTH GRID DATA.

In most cases, the grids would all be the same length. If not, the storage “boxes” in memory for the
reports being sorted would be the size of the largest grid.

THE SORT KEY.

The sort key for a report is always generated within a subroutine. This makes it easy and very
flexible to generate whatever sort key is needed. To define the sort key, the user just changes a little
code within this sort key subroutine. The sort key is often just one word long, but it can have
several words. If you are sorting ASCII records, you can still build a compact binary sort key.
SORT VARIABLE LENGTH RECORDS.

We store variable length records in a blocking structure called report-out (RPTOUT). This includes the length of each report and the blocks of data have checksum error protection. When we sort this data, all of the intermediate strings of data (and the final sorted string) are in this same blocking structure.

For the sort, the user declares the maximum size of any one record.

DO WE WANT TO USE BOEING’S SORT PROGRAM?

Boeing contacted me about 1995 – 96 to see if we might want to purchase the use of their sort program for about $200,000 to $250,000. After listening to their story for a while, it seemed to me that our sort was just as good and maybe better.

WHO CODED UP THE NCAR SORT PROGRAM?

Dennis Joseph wrote the code for the Quadratic Sort program soon after he arrived in Sep 1966. This version used a 2-way merge. Dennis remembers (in 2002) that he definitely coded the sort part of the code. We think that both Dennis and Roy Jenne prepared code for the merge part of the sort. Will Spangler arrived in July 1971. He changed the program to a 7-way merge in 1974.

ONE DEVELOPMENT THAT COULD GIVE LONGER STRINGS.

The sort merge program is very fast now. Therefore, it is not worth the effort to do much more work on it, if any. Also, any new changes could involve a significant amount of time to make certain that it has no bugs.

We noted that a new string must be started when all of the “lowest value” sort keys for the arrays are all smaller than the sort key for the report just output. However, at this time there will probably be a number of reports in memory that have sort keys larger than the one just output. In the present practice, we can no longer see these somewhat bigger sort keys, because each sort key array always calculates the smallest key. Some of the new reports being read in during the sort can have sort keys that are very small, and these will keep us from seeing reports that could still be put onto the present string.

There are some ways to fix this, but it would take a few more calculations. When a report has been output from a sort key array, we could calculate the smallest value for that array in a new way. We could choose the smallest sort key value that is still bigger than the key of the last report just output. This would still permit a little “blockage” but it would reduce it a lot, and make the strings longer.
**Figure 1.** This example has 3 arrays of sort keys. In each array we find the smallest sort key. Suppose that a report is output that is associated with a sort key in array #2. We read a new report into memory, and put its sort key into the empty slot in array #2. Then we calculate a new "smallest key" for that array.
QUADRATIC SORT/MERGE

The quadratic sort is intended to be used to sort data records from multiple files of large volumes, although it works well with files of any number or size.

'N-squared' records of maximum length (LWDS) are loaded into a work buffer (NARRAY) in 'N' stacks. 'N-squared' actual record lengths are stored in 'N' stacks. And 'N' stacks of sort keys of one or more words (NK) are generated. 'N' is constrained by the size (LENGTH) of the work buffer (NARRAY).

Comparison of sort keys from each record determines which sort key in each of these stacks is smallest. Subsequent comparisons among the smallest sort keys of each stack identifies the record in the buffer with the smallest sort key. That record is written to an output unit and a new record is read into its stack to fill the vacancy. Sort keys in the replenished stack are compared again to identify the new smallest sort key of that stack. Then comparison of the smallest sort keys from each stack again identifies the record with the smallest sort key and the record is written to the output unit. This continues as long as the sort key of each record written is larger than that of the previous record written.

However, if the sort key is smaller than that of the previous record written (i.e., out of order), a new output unit is started. The total number of output units (NUNIT, 2 to 7) is usually set to 7. Each time a record to be written has a smaller sort key than the previous written record a new output unit is started. When the chosen number of units have been used, the first is written to again and recycling through the units proceeds as needed. This creates ordered subsets of records ('strings') on the output units. The sort phase is completed in one pass; i.e., after all records are read and written once. The merge phase is used to transform these ordered strings into a single ordered string.

Beginning the merge phase, a record is read from each of the just written units. Merge keys (the same as the sort keys) are generated for each of these records, and the record with the smallest merge key is written to the first of a new bank of output units, totaling the same in number as the sort (NUNIT). So it is usually a 7-way merge. The same input unit of the record just written is used to read the replacement record, for which merge keys are generated, and comparisons made with the merge keys for records from the other units to determine the record with the smallest merge key. If the merge key is smaller than that of the record previously written, a new output unit is started and the same procedure, as in the sort phase, is followed in starting new output units and recycling through output units as needed until all the data is passed. Therefore, there may be NUNIT input units and NUNIT output units active at anyone time. In both the sort and merge phases only the number of units needed are used.
More than one pass through the data may be required to reduce the number of sorted strings to one. Each pass through the data will reduce the number of strings by at least a factor of NUNIT. For subsequent passes, the output units become the input units and the former input units become the output units.

Some factors to consider when running the sort/merge program

The merge would only take one pass if one could write, in the sort phase, as many units as there are strings, but the computer system is limited by memory and disk space.

Choosing the size of the work buffer affects both sort and merge phase performances. Providing too much space may increase the number of comparisons for each write in the sort phase, and too little space may increase the number of ordered strings and the number of passes required in the merge phase. Summarized as follows,

<table>
<thead>
<tr>
<th></th>
<th>size</th>
<th>sort time (comparisons)</th>
<th>number ordered strings</th>
<th>merge time (passes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>too much</td>
<td>increase</td>
<td>increase</td>
<td>decrease</td>
<td>decrease</td>
</tr>
<tr>
<td>too little</td>
<td>decrease</td>
<td>decrease</td>
<td>increase</td>
<td>increase</td>
</tr>
</tbody>
</table>

Another factor which affects over all performance is the innate order of the input data. The more closely it approximates the desired order, regardless of the size of the work space, the fewer will be the number of ordered strings and merge passes.

Variables chosen by the software engineer are:

- NUNIT - maximum number of units for either input or output (2-7)
- LENGTH - length of the single dimensioned work array (10000 64-bit words work well)
- LWDS - maximum length of a record (records lengths may be variable)
- NK - number of words in sort/merge key (maximum of 5)

Code for subroutine GKEY, which generates the sort/merge keys, must be supplied. Note: It is possible to create sort/merge keys that are mal-formulated and as such will cause the merge to never converge to a single ordered string. For example, an erroneous index to a location that that changes in value may cause this. Also, the value in the first word of the sort key may not be negative.
Merge section of the sort (Data Support, NCAR)

- Assume the records are unblocked system records
- The blocked record merge is like this too, but the buffers aren't switched

Set up tapes & constants

20 Rewind the inputs & outputs

Get an initial record from each input; IT1 and IT2

IF T1 is an EOF

IF T2 is an EOF

Print about tapes, things, records, & words

IF nothing

400

T2 EOF

30

Do we have key?

15

IF yes, End file outputs

Do we have key?

15

Set J1

IF Key 1 < prev key

Set F1 = 1

35

Do T1 an EOF

40

IF T2 EOF

PRINT key on error message

quit

50

Do Key 2 < prev key

32

Start a new string or order output tape a way Keys are greater than prev. Key

50

Wait, print of Key

200

Switch buffer 32A7F3B

switch buffer IT1F71S

200

Wait, print of Key

300

Start new string

etc

Note: If a key is once < prev key, it will stay there until we start a new string.
Merge of blocked report
- use Rptout

Good test of Variable Length Blocked Record MERGE

For multi-word sort Keys

This sort just has a 2-way merge. The recent code has a 7-way merge.

R. J. Jenne Jan 04

Date now:
Jan 3, 2004
R. J. Jenne
10/26/66 *FORTRAN

PROGRAM SORTBB
COMMON NSTRING, LWDS, IHALT, ITAPE, IOUT, NT1, NT2, NT3, NT4, NK,
2 LKEY(10), KEY1(10), KEY2(10), IBUF1(360), IBUF2(360), KBUF(360)
DIMENSION IREP1(100), IREP2(100), NARRAY(1000)
C THE SORT READS RECORDS FROM ITAPE AND WRITES SORTED STRINGS
C ALTERNATELY ON NT1 AND NT2
C THE MERGE MERGES STRINGS BACK AND FORTH BETWEEN NT1, NT2 AND
C NT3, NT4 UNTIL THE STRING COUNT IS ONE.
C
C USE SORTB AND MERGB FOR RECORDS BLOCKED BY RPTOUT
C USE SORT AND MERGE FOR UNBLOCKED RECORDS, SPECIFY MODE AND NTYPE
C THE FINAL OUTPUT TAPE IS IOUT, IS PRINTED, AND IS NT1 OR NT3
C WRITTEN OCT 1966 BY ROY JENNE AT NCAR

ITAPE = 1
NT1 = 2
NT2 = 3
NT3 = 8
NT4 = 7
C LWDS IS MAX LENGTH OF A LOGICAL RECORD, LENGTH IS WORK AREA OF SORT
C DIM IBUF1, IBUF2, KBUF BY 360 FOR BLOCKING RECORDS
C DIMENSION IREP1(LWDS), IREP2(LWDS), NARRAY(LENGTH)
C LENGTH IS MORE THAN 4 * LWDS
C NK WORDS OF LKEY, KEY1, AND KEY2 ARE USED FOR SORT KEYS

LWDS = 100
LENGTH = 1000
NK = 1
KPASS = 0
C IF KPASS = 1, BYPASS SORT AND DO JUST ONE MERGE PASS
IF (KPASS .EQ. 1) GO TO 50

CALL SORTB(NARRAY, LENGTH, IREP1, IREP2)
IF (IHALT .EQ. 1) GO TO 500
IF (NSTRING .EQ. 1) GO TO 510
C THE TWO TAPES TO MERGE ARE ON NT1 AND NT2, ONE MAY BE EMPTY, BUT
C EACH HAS AN END OF FILE
CALL MERGB(IREP1, IREP2, KPASS)
IF (IHALT .EQ. 1) GO TO 500
PRINT 925, IOUT
925 FORMAT (1H, 'THE OUTPUT TAPE IS', I4)
GO TO 520
920 FORMAT (1H, 'THE SORT IS COMPLETE ON*', I4, ' WITHOUT A MERGE')
GO TO 520
520 REWIND IOUT
DO 521 I = 1, 7
KBUF(I) = 0
CALL RPTIN(IOUT, KBUF, IREP1, LWDS, NT1)
IF (NST1 .EQ. 1) CALL EXIT
PRINT 991, NST1, LWDS, IREP1(2), IREP1(10)
991 FORMAT (1H, 'SEQ OUTPUT*', 2I4, 2X, 2I8)
GO TO 525
END

LENGTH OF ROUTINE SORTBB
002431

SUBROUTINES CALLED

10
SUBROUTINE MERGEB(IREP1, IREP2, KPASS)

COMMON NSTRING, LWDS, IHALT, ITAPE, IOUT, NT1, NT2, NT3, NT4, NK

2 LKEY(10), KEY1(10), KEY2(10), IBUF1(360), IBUF2(360), KBUF(360)

DIMENSION IREP1(2), IREP2(2)

C MERGE FOR VARIABLE LENGTH BLOCKED RECORDS
C THE TWO TAPES TO MERGE ARE ON NT1 AND NT2. ONE MAY BE EMPTY, BUT
C EACH HAS AN END OF FILE
C WRITTEN BY ROY JENNE AT NCAR IN OCT 66
C THE FINAL OUTPUT TAPE WILL BE IT1 OR JT1
C NK WORDS OF LKEY, KEY1, AND KEY2 ARE USED FOR SORT KEYS
C THE REAL DIMENSIONS OF LKEY, KEY1, AND KEY2 ARE SET IN THE MAIN
C USE KPASS = 1 TO DO ONLY 1 MERGE PASS

000010 IT1 = NT1
000011 IT2 = NT2
000013 JT1 = NT3
000014 JT2 = NT4

C START A NEW MERGE PASS HERE

000016 20 REWIND IT1
000020 REWIND IT2
000022 REWIND JT1
000024 REWIND JT2
000026 IOUT = JT1
000027 NSTRING = 1
000031 DO 23 I = 1, 7
000032 IBUF1(I) = 0
000033 IBUF2(I) = 0
000034 23 KBUF(I) = 0
000037 ISWIT = 0
000040 CALL RPTIN(IT1, IBUF1, IREP1, NWDR1, 10, LWDS, IST1)
000045 CALL RPTIN(IT2, IBUF2, IREP2, NWDR2, 10, LWDS, IST2)

C

000055 30 IF (IST1 .EQ. 1) GO TO 400
000061 IF (ISWIT .EQ. 2) GO TO 35
000063 IF (IST1 .EQ. 0) GO TO 33
000064 PRINT 930, IT1, IBUF1(2)
000077 IHALT = 1
000100 930 FORMAT (1H *BAD READ*, 2I6)
000100 33 CALL GKEY(IREP1,KEY1)
000102 JK1 = 0
000103 IF (ISWIT .EQ. 0) GO TO 35
000106 DO 133 I = 1, NK
000107 IF (KEY1(I) .GT. LKEY(I)) GO TO 35
000113 IF (KEY1(I) .EQ. LKEY(I)) GO TO 133
000115 JK1 = 1
000116 GO TO 35
000116 133 CONTINUE

C

000121 35 IF (IST2 .EQ. 1) GO TO 200
000123 40 IF (ISWIT .EQ. 1) GO TO 45
000125 IF (IST2 .EQ. 0) GO TO 43
000126 PRINT 930, IT2, IBUF2(2)
000141 IHALT = 1
000142 43 CALL GKEY(IREP2,KEY2)
000144 JK2 = 0
000145 IF (ISWIT .EQ. 0) GO TO 45
000150 DO 143 I = 1, NK
000151 IF (KEY2(I) .GT. LKEY(I)) GO TO 45
000155 IF (KEY2(I) .EQ. LKEY(I)) GO TO 143
000157 JK2 = 1
GO TO 45
CONTINUE
45 IF (IST1 .EQ. 1) GO TO 300
IF (JK1 .EQ. 1) GO TO 60
IF (JK2 .EQ. 1) GO TO 200
DO 72 I = 1,NK
IF (KEY2(I) .LT. KEY1(I)) GO TO 300
IF (KEY2(I) .EQ. KEY1(I)) GO TO 72
GO TO 200
72 CONTINUE
GO TO 200
GO TO 70
GO TO 300

C OUTPUT A RECORD FROM IT1
200 IF (JK1 .EQ. 0) GO TO 220
C SWITCH OUTPUT TAPES FOR A NEW STRING
NSTRING = NSTRING + 1
JK2 = 0
CALL RPTOUT (IOUT,KBUF,IREP1,NWDR1,2)
IF (IOUT .EQ. JT1) GO TO 210
IOUT = JT1
GO TO 220
210 IOUT = JT2
DO 221 I = 1,NK
LKEY(I) = KEY1(I)
GO TO 223
PRINT 985, NWDR1,LWDS,IBUF1(2)
IHALT = 1
FORMAT (1H,*REC SIZE PROBS*,316)
CALL RPTOUT (IOUT,KBUF,IREP1,NWDR1,0)
CALL RPTIN(IT1,IBUF1,IREP1,NWDR1,10,LWDS,IST1)
ISWIT = 1
GO TO 30

C OUTPUT A RECORD FROM IT2
300 IF (JK2 .EQ. 0) GO TO 320
C SWITCH OUTPUT TAPES FOR A NEW STRING
NSTRING = NSTRING + 1
JK1 = 0
CALL RPTOUT (IOUT,KBUF,IREP2,NWDR2,2)
IF (IOUT .EQ. JT1) GO TO 310
IOUT = JT1
GO TO 320
310 IOUT = JT2
DO 321 I = 1,NK
LKEY(I) = KEY2(I)
GO TO 323
PRINT 985, NWDR2,LWDS,IBUF2(2)
IHALT = 1
323 CALL RPTOUT (IOUT,KBUF,IREP2,NWDR2,0)
CALL RPTIN(IT2,IBUF2,IREP2,NWDR2,10,LWDS,IST2)
ISWIT = 2
GO TO 30

400 IF (IST2 .EQ. 1) GO TO 500
GO TO 40

12
10/26/66 *FORTRAN

SUBROUTINE SORTB(NARAY, LENGTH, IREP1, IREP2)
COMMON NSTRING, LWDS, IHALT, ITAPE, OUT, NT1, NT2, NT3, NT4, NK,
1 LKEY(10), KEY1(10), KEY2(10), INBUF1(360), INBUF2(360), KBUF(360)
2 DIMENSION IREP1(2), IREP2(2)

C THE REAL DIMENSIONS OF IREP1 AND IREP2 ARE SET IN THE MAIN

000012  IT1 = NT1
000013  IT2 = NT2
000015  REWIND IT1
000016  REWIND IT2
000020  DO 610 J=1,10
000022  DO 610 I=0,18,2
000023  IREP1(2) = I + 220 - 20*J
000027  IREP1(10) = I + 220 - 20*J
000032  IREP2(2) = I + 221 - 20*J
000036  IREP2(10) = I + 221 - 20*J
000041  IF ( (J/2)*2 .EQ. J) GO TO 6
000043  CALL RPTOUT(IT1, INBUF1, IREP1, 10, 0)
000046  CALL RPTOUT(IT2, INBUF2, IREP2, 13, 0)
000055  CONTINUE
610 CONTINUE
000064  CALL RPTOUT(IT1, INBUF1, IREP1, 10, 2)
000067  CALL RPTOUT(IT2, INBUF2, IREP2, 12, 2)
000075  END FILE IT1
000101  END FILE IT2
000103  RETURN
000104  END

LENGTH OF ROUTINE SORTB
000132

SUBROUTINES CALLED
REWIND
RPTOUT
ENDFIL
END

VARIABLE ASSIGNMENTS
LKEY = 000012  KEY1 = 000024  KEY2 = 000036  INBUF1 = 000050
INBUF2 = 000020  KBUF = 001370  IT1 = 000126  NT1 = 000005
IT2 = 000127  NT2 = 000006  J = 000130  I = 000131

Generated string on 2 tapes for the test

\[ 10 \times 10 \times 13 = 1300 \]
\[ 5 \times 10 \times 10 = 500 \]

Net Words/Week: 1800 Words / 150 LOC
10/26/66 *FORTRAN

SUBROUTINE GKEY(IREP, KEY)
DIMENSION IREP(20)

C GET THE SORT KEY
KEY = IREP(2)
RETURN
END

LENGTH OF ROUTINE GKEY
000016

SUBROUTINES CALLED
END

VARIABLE ASSIGNMENTS

TOTAL CORE USED 013321

ENTRY POINT LOCATION ROUTINE ORIGIN
RPTOUT 003007 003000
RPTIN 003232 003221
CNTIN 003577 003577
GETCNT 003603 003577
CHKSUM 003606 003606
MOVE 003615 003615
SORTBB 003633 003630
MERGEA 010426 010421
SORTB 011242 011234
GKEY 011372 011366
EXIT 011404 011404
END 011404 011404
STOP 011404 011404
KODEA 011407 011407
OUTPIC 012272 012272
BACKSP 012754 012754
IFEND 013055 012754
REWINM 012757 012754
ENDFIL 013072 013072
RDTAPE 013114 013114
WRTAPE 013117 013114
IOWAIT 013140 013114

COMMON BLOCKS LOCATION
006261

Get sort key (Gkey)
replies the sort key
C WE HAVE HIT END FILE ON BOTH MERGE INPUTS
000364 500 CALL RPTOUT(IOUT,KBUF,IREP1,NWDR1,2)
000370 END FILE JT1
000374 END FILE JT2
000376 IREC = IBUF1(2) + IBUF2(2)
000400 IPREC = IBUF1(3) + IBUF2(3)
000402 IWDS = IBUF1(4) + IBUF2(4) - 2*IPREC
000406 JWDS = KBUF(4) - 2*KBUF(3)
000412 PRINT 980,IT1,IT2,IREC,IWDS,IPREC
000431 980 FORMAT(1H,**INPUT TAPES **213,** WITH*,15,** REC AND WITH**,I7,*  
          * WORDS,**23X,15,** PHYS RECORDS*)
000431 PRINT 982,IT1,IT2,KBUF(2),JWDS,NSTRING,KBUF(3)
000456 982 FORMAT(1H,**OUTPUT TAPES**213,** WITH*,15,** REC*,7X,15,** WORDS,**  
          * I6,* STRINGS**,8X,15,** PHYS RECORDS*)
000456 IF (IREC .NE. KBUF(2)) IMALT = 1
000461 IF (IWDS .NE. JWDS) IMALT = 1
000464 IF (NSTRING .EQ. 1) GO TO 520
C SWITCH INPUT TAPES AND START A NEW MERGE PASS
000466 IF (KPASS .EQ. 1) GO TO 518
000467 NWORK = IT1
000470 IT1 = JT1
000472 JT1 = NWORK
000473 NWORK = IT2
000474 IT2 = JT2
000475 JT2 = NWORK
000476 GO TO 20
C WE WANTED TO DO MERGE IN 1 PASS BUT CANT
000476 518 PRINT 988
000503 IMALT = 1
000504 988 FORMAT (1H,**ABORT, CANT DO MERGE IN ONE PASS**) 
000504 520 IOUT = JT1
000506 RETURN
000506 END

LENGTH OF ROUTINE MERGEB
000613

SUBROUTINES CALLED
REWINM
RPTIN
OUTPTC
GKEY
RPTOUT
ENDFIL

VARIABLE ASSIGNMENTS
LKEY = 000012  KEY1 = 000024  KEY2 = 000036  IBUF1 = 000050
IBUF2 = 000620  KBUF = 001370  IT1 = 000572  NT1 = 000005
IT2 = 000573  NT2 = 000006  JT1 = 000574  NT3 = 000007
JT2 = 000575  NT4 = 000010  IOUT = 000004  NSTRING = 000000
I = 000576  ISWIT = 000577  NWDR1 = 000600  LWDS = 000001
IST1 = 000601  NWDR2 = 000602  IST2 = 000603  IMALT = 000002
JK1 = 000604  NK = 000011  JK2 = 000605  IREC = 000606
IPREC = 000607  IWDS = 000610  JWDS = 000611  NWORK = 000612
The output from the sort

CPU TIME = 3 SECONDS

In each merge pass, the number of strings is reduced. Get 1 string when we are done. - Ray Dana Jan 2004
Run the sort

Good fast Unblocked MERGE

For multi-word sort keep
PROGRAM SORTUU
DIMENSION IREP1(1000), NARRAY(1000)
COMMON NSTRING, LWDS, IHALT, ITAPE, IOUT, NT1, NT2, NT3, NT4, MODE, NTYPE,
       2 NK, LKEY(10), KEY1(10), KEY2(10)
C THE SORT READS RECORDS FROM ITAPE AND Writes sorting STRINGS
C ALTERNATELY ONTO NT1 AND NT2
C THE MERGE Merges strings back and forth between NT1, NT2 AND
C NT3, NT4 UNTIL THE STRING COUNT IS ONE.
C USE SORT AND MERGE FOR UNBLOCKED RECORDS. SPECIFY MODE AND NTYPE
C USE SORTE AND MERGER FOR RECORDS BLOCKED BY RPTOUT
C THE FINAL OUTPUT TAPE IS IOUT, IS PRINTED, AND IS NT1 OR NT3
C WRITTEN OCT 1966 BY ROY JENNE AT NCAR

      ITAPE = 1
      NT1 = 2
      NT2 = 3
      NT3 = 8
      NT4 = 7
C NTYPE = 0 FOR SYSTEM RECORDS. ASSUME UNBLOCKED RECORDS
      MODE = 1
      NTYPE = 0
C LWDS IS MAX LENGTH OF A LOGICAL RECORD, LENGTH IS WORK AREA OF SORT
C DIMENSION IREP1(LWDS), NARRAY(LENGTH)
C LENGTH IS MORE THAN 4 * LWDS
C NK WORDS OF LKEY, KEY1, AND KEY2 ARE USED FOR SORT KEYS
      LWDS = 100
      LENGTH = 1000
      NK = 1
      KPASS = 0
C IF KPASS = 1, BYPASS SORT AND DO JUST ONE MERGE PASS
      IF (KPASS .EQ. 1) GO TO 50

      CALL SORT (NARRAY, LENGTH, IREP1)
      IF (IHALT .EQ. 1) GO TO 500
      IF (NSTRING .EQ. 1) GO TO 510
C THE TWO TAPES TO MERGE ARE ON NT1 AND NT2. ONE MAY BE EMPTY, BUT
C EACH HAS AN END OF FILE
      50 CALL MERGE (NARRAY, KPASS)
      IF (IHALT .EQ. 1) GO TO 500
      PRINT 925, IOUT
      925 FORMAT (1H, * THE OUTPUT TAPE IS*, I4)
      GO TO 520
      PRINT 930
      930 FORMAT (1H, * WE HAD A SORT OR MERGE PROBLEM* )
      GO TO 520
      PRINT 910
      910 FORMAT (1H, * THE SORT IS COMPLETE ON*, I4, * WITHOUT A MERGE*)
      520 REWIND IOUT
      520 FORMAT (1H, * THE SORT IS COMPLETE ON*, I4, * WITHOUT A MERGE*)
      CALL RDTAPE (IOUT, MODE, NTYPE, NARRAY(I), LWDS)
      CALL IOWAIT (IOUT, NST1, JWDS)
      IF (NST1 .EQ. 1) CALL EXIT
      PRINT 991, NST1, JWDS, NARRAY(I), NARRAY(10)
      991 FORMAT (1H, * SEQ output*, 2I4, 2X, 2I8)
      GO TO 521
      END
SUBROUTINE MERGE (NARAY,KPASS)

DIMENSION NARAY(4)
COMMON NSTRING,LWDS,ILHAT,ITAPE,IOUT,NT1,NT2,NT3,NT4,MODE,NTYPE,
     2 NK,LKEY(10),KEY1(10),KEY2(10)

C MERGE SECTION OF SORT
C WRITTEN BY ROY JENNE AT NCAR IN SEPT 66
C THE TWO INPUT TAPES TO MERGE ARE ON IT1 AND IT2. ONE MAY BE EMPTY
C BUT EACH HAS END FILE.
C THE FINAL OUTPUT TAPE WILL BE IT1 OR JT1
C NK WORDS OF LKEY, KEY1, AND KEY2 ARE USED FOR SORT KEYS

IT1 = NT1
IT2 = NT2
JT1 = NT3
JT2 = NT4
I1A = 1
I1B = I1A + LWDS
I2A = I1B + LWDS
I2B = I2A + LWDS
PRINT 935
935 FORMAT (1H0,** MERGE SECTION OF THE SORT*)

C START THE MERGE
20 REWIND IT1
20 REWIND IT2
20 REWIND JT1
20 REWIND JT2
IOUT = JT1
NSTRING = 1
IREC = 0
JREC = 0
IWDS = 0
JWDS = 0
ISWIT = 0
CALL RDTAPE (IT1,MODE,NTYPE,NARAY(I1A),LWDS)
CALL RDTAPE (IT2,MODE,NTYPE,NARAY(I2A),LWDS)
CALL IOWAIT (IT1,I1T1,NWDR1)
CALL IOWAIT (IT2,I2T2,NWDR2)

30 IF (IST1 .EQ. 1) GO TO 400
30 IF (ISWIT .EQ. 2) GO TO 35
IREC = IREC + 1
IWDS = IWDS + NWDR1
30 IF (IST1 .EQ. 0) GO TO 33
PRINT 930,IT1,IREC
IHALT = 1
930 FORMAT (1H0,**BAD READ#, 2I6)
33 CALL GKEY(NARAY(I1A),KEY1)
JK1 = 0
35 IF (ISWIT .EQ. 0) GO TO 35
35 DO 133 I = 1,NK
35 IF (KEY1(I) .GT. LKEY(I)) GO TO 35
35 IF (KEY1(I) .EQ. LKEY(I)) GO TO 133
35 IF (KEY1(I) .LT. LKEY(I)) GO TO 133
JK1 = 1
35 GO TO 35
133 CONTINUE

35 IF (IST2 .EQ. 1) GO TO 200
40 IF (ISWIT .EQ. 1) GO TO 45
IREC = IREC + 1
IWDS = IWDS + NWDR2
40 IF (IST2 .EQ. 0) GO TO 43

PRINT 930, IT2, IREC
IHALT = 1
43 CALL GKEY(\text{\textit{NARAY(I2A)},KEY2})
JK2 = 0
IF (ISWIT \cdot EQ. 0) GO TO 45
DO 143 I = 1,NK
IF (KEY2(I) \cdot GT, LKEY(I)) GO TO 45
IF (KEY2(I) \cdot EQ, LKEY(I)) GO TO 143
JK2 = 1
GO TO 45
143 CONTINUE
45 IF (IST1 \cdot EQ. 1) GO TO 300
IF (JK1 \cdot EQ. 1) GO TO 60
IF (JK2 \cdot EQ. 1) GO TO 200
70 DO 72 I = 1,NK
IF (KEY2(I) \cdot LT, KEY1(I)) GO TO 300
IF (KEY2(I) \cdot EQ, KEY1(I)) GO TO 72
GO TO 200
72 CONTINUE
GO TO 200
60 IF (JK2 \cdot EQ. 1) GO TO 70
GO TO 300

C
C SWITCH BUFFERS AND OUTPUT A RECORD FROM IT1
200 NWORK = I1A
I1A = I1B
I1B = NWORK
IF (JK1 \cdot EQ. 0) GO TO 220

C SWITCH OUTPUT TAPES FOR A NEW STRING
NSTRING = NSTRING + 1
JK2 = 0
IF (IOUT \cdot EQ. JTI) GO TO 210
IOUT = JTI
GO TO 220
210 IOUT = JTI2
DO 221 I = 1,NK
LKEY(I) = KEY1(I)
KOWD = NWD1
CALL WRRTAPE(IOUT,MODE,NTYPE,NARAY(I1B),KOWD)
CALL RDTAPE(IT1,MODE,NTYPE,NARAY(I1A),LWDS)
CALL IOWAIT(IOUT,IOUT,IOUT,KKWD)
JREC = JREC + 1
JWDS = JWDS + KKWD
ISWIT = 1
CALL IOWAIT(IT1,IST1,NWD1)
IF (KKWD \cdot NE, KOWD) GO TO 223
IF (ISTOUT \cdot EQ. 0) GO TO 30
223 PRINT 985,JREC,NSTRING,KKWD
IHALT = 1
985 FORMAT (1H \#TROUBLE ON A WRITE\# 317)
GO TO 30

C
C SWITCH BUFFERS AND OUTPUT A RECORD FROM IT2
300 NWORK = I2A
I2A = I2B
I2B = NWORK
IF (JK2 \cdot EQ. 0) GO TO 320

C SWITCH OUTPUT TAPES FOR A NEW STRING
NSTRING = NSTRING + 1
JK1 = 0
IF (IOUT \cdot EQ. JTI) GO TO 310
IOUT = JTI
GO TO 320
310 IOUT = JT2
320 DO 321 I = 1, NK
321 LKEY(I) = KEY2(I)
322 KOWD = NWD2
323 CALL WRTAPE (IOUT*MODE*NTYPE*NARAY(I2B)*KOWD)
324 CALL RDTAPE (IT2,MODE,NTYPE*NARAY(I2A),LWDS)
325 CALL IOWAIT(IOUT*ISTOUT,KKWD)
326 JREC = JREC + 1
327 JWD5 = JWD5 + KKWD
328 ISWIT = 2
329 CALL IOWAIT (IT2,IST2,NWD2)
330 IF (KKWD .NE. KOWD) GO TO 323
331 IF (ISTOUT .EQ. 0) GO TO 30
332 PRINT 985, JREC, NSTRING, KKWD
333 IHALT = 1
334 GO TO 30
335 400 IF (IST2 .EQ. 1) GO TO 500
336 GO TO 40
337 C
338 C WE HAVE HIT END FILE ON BOTH MERGE INPUTS
339 500 END FILE JT1
340 END FILE JT2
341 PRINT 980, JT1, JT2, JREC, JWD5
342 980 FORMAT (1H **INPUT TAPES**, 2I3, **WITH** I5, **REC AND WITH** I7, X * WORDS,*)
343 PRINT 982, JT1, JT2, JREC, JWD5, NSTRING
344 982 FORMAT (1H **OUTPUT TAPES**, 2I3, **WITH** I5, **REC**, I6, **STRINGS,**) C
345 IF (JREC .NE. JREC) IHALT = 1
346 IF (JWD5 .NE. JWD5) IHALT = 1
347 IF (NSTRING .EQ. 1) GO TO 520
348 C Switch input tapes and start a new merge pass
349 IF (KPASS .EQ. 1) GO TO 518
350 518 PRINT 988
351 NWORK = IT1
352 IT1 = JT1
353 JT1 = NWORK
354 NWORK = IT2
355 IT2 = JT2
356 JT2 = NWORK
357 GO TO 20
358 C WE WANTED TO DO MERGE IN 1 PASS BUT CANT
359 518 PRINT 988
360 IHALT = 1
361 988 FORMAT (1H **ABORT, CANT DO MERGE IN ONE PASS**) 518
362 520 IOUT = JT1
363 REWIND IOUT
364 RETURN
365 520 RETURN
366 END
367 END

LENGTH OF ROUTINE MERGE
000623

SUBROUTINES CALLED
OUTPC
REWIND
RDRTAE
IOWAIT
GKEY
WRRTAE
ENDFIL
END
SUBROUTINE SORT (NARAY, LENGTH, IREPI)

DIMENSION NARAY(4), IREPI(2)
COMMON NSSTRING, LWDS, IHALT, ITAPE, IOUT, NT1, NT2, NT3, NT4, MODE, NTYPE,
                2 NK, KEY(10), KEY1(10), KEY2(10)

IT1 = NT1
IT2 = NT2
REWIND ITI
REWIND IT2
IIA = 1
I2A = 50
DO 610 J = 1, 10
DO 610 I = 0, 18, 2
NARAY(I1A) = I + 220 - 20*J
NARAY(I1A+9) = I + 220 - 20*J
NARAY(I2A) = I + 221 - 20*J
NARAY(I2A+9) = I + 221 - 20*J
IF ((J/2)*2 .EQ. J) GO TO 6
CALL WRTAPE (IT1, MODE, NTYPE, NARAY(I1A), 10)
CALL WRTAPE (IT2, MODE, NTYPE, NARAY(I2A), 10)
CALL IOWAIT(IT1, NST1, IWDS)
CALL IOWAIT(IT2, NST2, JWDS)
IF (NST1 * NST2 .EQ. 0) GO TO 610
PRINT 990
FORMAT (1H, *HELP*)
CONTINUE
END FILE IT1
END FILE IT2
RETURN
END

LENGTH OF ROUTINE SORT

SUBROUTINES CALLED
REWIND
WRRTAPE
IOWAIT
Outputs
ENDFILE
END
10/26/66 *FORTRAN

```fortran
SUBROUTINE GKEY(IREP,KEY)

DIMENSION IREP(16)

C GET THE SORT KEY.
KEY = IREP(1)
RETURN
END

LENGTH OF ROUTINE GKEY
000016

SUBROUTINES CALLED
END

VARIABLE ASSIGNMENTS

+-------------------------------------------+----------------+----------------+
| ENTRY POINT    | LOCATION       | ROUTINE ORIGIN |
+----------------+----------------+----------------+
| SORTUU         | 003055         | 003052         |
| MERGE          | 005337         | 005333         |
| SORT           | 006163         | 006156         |
| GKEY           | 006333         | 006327         |
| EXIT           | 006345         | 006345         |
| END            | 006345         | 006345         |
| STOP           | 006345         | 006345         |
| KODER          | 006350         | 006350         |
| OUTPTC         | 007233         | 007233         |
| BACKSP         | 007715         | 007715         |
| IFENDF         | 010016         | 007715         |
| REWINM         | 007720         | 007715         |
| ENDFIL         | 010033         | 010033         |
| ROTAPE         | 010055         | 010055         |
| WRTAPE         | 010060         | 010055         |
| IOWAIT         | 010101         | 010055         |
+----------------+----------------+----------------+

COMMON BLOCKS    LOCATION
                    003000
```

23
### MERGE SECTION OF THE SORT

<table>
<thead>
<tr>
<th>INPUT TAPES</th>
<th>OUTPUT TAPES</th>
<th>REC AND WITH</th>
<th>1500 WORDS</th>
<th>STRINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 3 WITH</td>
<td>8 7 WITH</td>
<td>150 REC</td>
<td>1500 WORDS</td>
<td>10</td>
</tr>
<tr>
<td>8 7 WITH</td>
<td>2 3 WITH</td>
<td>150 REC</td>
<td>1500 WORDS</td>
<td>5</td>
</tr>
<tr>
<td>2 3 WITH</td>
<td>8 7 WITH</td>
<td>150 REC</td>
<td>1500 WORDS</td>
<td>3</td>
</tr>
<tr>
<td>2 3 WITH</td>
<td>8 7 WITH</td>
<td>150 REC</td>
<td>1500 WORDS</td>
<td>2</td>
</tr>
<tr>
<td>8 7 WITH</td>
<td>2 3 WITH</td>
<td>150 REC</td>
<td>1500 WORDS</td>
<td>1</td>
</tr>
</tbody>
</table>

**THE OUTPUT TAPE IS 8**

<table>
<thead>
<tr>
<th>SEQ OUTPUT</th>
<th>0 10</th>
<th>21 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>23 23</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>25 25</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>27 27</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>29 29</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>31 31</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>33 33</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>35 35</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>37 37</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>39 39</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>40 40</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>41 41</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>42 42</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>43 43</td>
</tr>
<tr>
<td>SEQ OUTPUT</td>
<td>0 10</td>
<td>44 44</td>
</tr>
</tbody>
</table>

---

**CPU TIME = 7 SECONDS**
This is the first NMC A sort listing from Alan Kay. (This is not a
DIAGRAM - he took listing to try to debug it)

Test 1: Tried to parallel a sort of YL260
that was done on the 3600. Data was
queued in some records - (it did compared but
rest of data in record didn't). Sort tape
also had more rec than input. Test A1-A3

Test 2: Tried to parallel sort of YL257
We lost records 4 made hash!

(1 I toyed with testing much later)

1) We first got our quadratic sort running

2) Years later we coded up a binary tree sort,
it was not faster than the quadratic sort
so we dropped it. The indexing within
binary tree probably adds extra overhead.

-Ray Kerns, Jan 2004