A A R D V A R K

DATA SHEET

SUBJECT: A Multiple purpose Memory Sorting Subroutine and Application Examples

PURPOSE AND INTRODUCTION

The Purpose of this Data Sheet is to define and describe an extremely simplified and compact general purpose BASIC subroutine that provides the nucleus for sequencing memory resident or 'direct access' file resident records containing one or many fields of string or numeric data in either ascending or descending order - without disturbing the original record contents.

SORTING - THE GENERAL PROBLEM

Sooner or later the 'computerist' will have a need to sort. There are a myriad examples that come to mind. To name a few, there is that mailing list you would like to sort by name and also by zip code, those 'x,y' coordinates you would like to sort by x's within y's to facilitate plotting, those random access file records for which you would like sorted multiple access keys, and so on. All of these sorting problems have common attributes. They involve object records of one or more fields that are treated as a unit; one or more of these fields may be used as the combined sort key or separate keys; and the purpose is to arrange the object records in a desired sequence or to be able to access these records in a desired sequence or sequences. Generally speaking, most sorting schemes end up physically rearranging the object records, or at least the sort key elements, thereby losing the original sequence.

BASIC APPROACH USED HEREIN

The basic theme of this sorting scheme is to construct sorted indexes to the object records. The actual record field contents and the sort key array contents are not disturbed by the sorting. This scheme results in faster sorts and the use of less memory to store the results of multiple sorts on the same or different object records.

As specified in Appendix A, the A%( ) array is the product of the sort subroutine. It constitutes the 'sorted index' for the object records. It contains the list of indexes to the object records that defines the last requested record sequence, as illustrated by the following table:

I	A%(I)
1	3 = index to 1st record in this sequence
2	2 = index to 2nd record in this sequence
3	4 = index to 3rd record in this sequence
4	1 = index to last record in the sequence

Assuming 4 object records; i.e., N=4

HOW TO USE THE SORTING SCHEME

Most sorting tasks may be accomplished with just one call to the sorting subroutine; i.e., when there is only one key field involved. Complex sorts involving two or more key fields may be accomplished by successive calls to the subroutine - one for each key field, in the reverse order of the key hierarchy. To illustrate how to use this scheme, actual example programs will be discussed - starting with the simple and progressing to the more complex. Please review Appendix A before reading the following examples. It is also best to key in and run the example programs to become familiar with them to better understand the following discussions.

APPLICATION PROGRAM EXAMPLES

1. Simple Single Field Record Sorts - Appendix B

This program allows entry of both string and numeric lists which are to be sorted. The string list is stored directly in the A$( ) array and the numeric list is transferred to the A( ) array - see line 110. On line 120 the function code 'A' is computed and on line 121 the A%( ) array is initialized to the unsorted sequence of the list to be sorted. Note on lines 130 and 140 that the original unsorted sequence of the respective list is displayed via the use of the subscript 'I' and the sorted sequence via use of subscript 'A%(I)'.

2. Multiple Field Record Sorts - using one key field - Appendix C

This program demonstrates how to list memory resident records by any one of the record fields. It is also a working program in that any type records may be defined and stored in the DATA statements on lines 2000-3999. You can generate mailing lists by name and zip code if you like.

Note the use of the 2 dimensional object record array namely REC$(REC#,FLD#). This facilitates record and field indexing. On line 45 the A%( ) array is initialized for both the sort subroutine call and before the listing of the unsorted records. The same subroutine - at line 500 - is used to list both the unsorted and sorted records. At lines 215 and 220 the selected key field is transferred to the appropriate string or numeric array respectively.

3. Multiple Field Record Sorts - using multiple Key - Appendx D

This program is a generalized version of the one shown in Appendix C. It allows identification of from one to all fields in a record type as the hierarchial sort key. It is the most complex program in the examples and does illustrate the general case for sorting memory resident records - or any records regardless of where they reside.

The program allows the selection of the number of fields to be included in the hierarchial sort key along with the field number and ascending or descending order for each hierarchy level. For example, a 2 key sort on zip code and name would be sequenced first by name and then by zip code to allow alphabetical listing of names within zip code.

Note the use a data type code in DATA lines 2001-2007 where S=string, N=numeric and B=both or either. Note also the use of the SK%( ), TF$( ) and SO%( ) arrays to identify the key field number, data type code and ascending or descending order code respectively, for each key hierarchy level. Line 260 forces the listing of the sorted records after each key level sort - for demonstration purposes. This can be changed to just list the final result by changing line 260 as follows:

```
260 NEXT SK: GOSUB500: GOTO 99
```

Note that the block of lines from 200 to 260 could have been written as a higher level sort subroutine, namely a subroutine to handle the general sort case involving one or many key levels. This higher level subroutine would require the additional reserved array names to serve the same functions performed by NK%, SK, SK%( ), TF$( ), SO%( ) and REC$( ). Notice that the FOR-NEXT loop started at line 200 indexes through the sort key hierarchy levels backwards, so that the highest level key is sorted last.

4. Simple Disk File Record Sorts - Appendix E

This is a mundane program whose sole function is to provide an example - particularly for 65D users - of how random access files may be sorted or accessed in a sequential manner. The program is not generic like the one in Appendix D; it cannot be used for the 'general records' case.

Although the whole file consists of records containing only the 2 key fields - name & zip code - it should be obvious that the record format could have been expanded to include other fields; e.g., street, city, state, telephone, etc. The point being that the whole file does not have to be read into memory to accomplish the sorts - only the key fields do.

The "retrieval" portion of the program statements are included on lines 300-420. Line 320 is used to fill both the A$( ) and A( ) arrays directly with name and zip code respectively, so there is no need to transfer the key field data to the appropriate sort key array prior to sorting. Line 355 satisfies all of the 'entry argument' set up described in Appendix A, prior to the name sort call; as does line 398 for the zip code sort call.

Note the use of the N%() array in line 360. This use was included solely to illustrate the multiple key index feature commonly referred to as "inverted indexes". For example, the combination of the A$() and N%() arrays provide an index to the example disk file by "name". The A$() array is not in name sequence, but when accessed using the N%() array as its index the effect is the same. Similarly the A() and A%() array combination provide an index to the file records by zip code.

A side note, when running this program it becomes obvious that the 65D operating system reads a track from the disk file for every 'DISK GET' call - and also writes a track for every 'DISK PUT' call - whether it needs to or not. This inefficiency and how to overcome it will be covered in a separate DATA SHEET on "65D data file accessing".

SORT SPEED

The "bubble sort" algorithm used in the subject sort subroutine is relatively simple to understand and is very compact, but it is not as fast as other algorithms. Speed was not the primary problem addressed here.

Speed can be increased considerably just by moving the sort subroutine from line 10000 down to the very beginnng of the program, because of the way BASIC resolves its GOSUBS's.

APPENDIX A

SUBJECT: Multiple Purpose Memory Sorting Subroutine Specifications

A. Synopsis of Variables and Calling Arguments Used

1. Variables Used

Calling arguments: A,N,AZ( ),A$( ),A( )
Local to subroutine: I,J,K,AZ

2. Argument Definitions

NAME	ENTRY	RETURNED	DEFINITION
A	x		Sort function code =0,1,2 or 3 0 = string sort - ascending 1 = string sort - descending 2 = numeric sort - ascending 3 = numeric sort - descending
		x	Returned unchanged if entry code is valid, else value is (-1)
A$( )	x		Key field array to be sorted - if A < 2
A( )	x		Key field array to be sorted - if A > 1
N	x		Number of array elements to be sorted
AZ( )			Record index array defining "last" sequence
	x		Original or intermediate sequence
		x	Sequence after "this" sort

Note: Variable is not changed if not noted in "RETURNED" column

3. Argument Definition Clarification

a. Sort Key array: A$( ) or A( )

The subject sort subroutine will perform either a string or numeric sort on any given call, which applies to either the A$( ) or A( ) array respectively. The alternate array is not used in the given call.

b. Sort array size

The array elements sorted will be elements 1 through "N" inclusive. "N" must be less than or equal to the dimensioned array size

c. Record index array: AZ( )

The elements of the sort Key array - A$( ) or A( ) - are not disturbed by the sort subroutine. The result of the sort is recorded in the record index array AZ( ), which will contain the sorted list of subscript references to the respective sort Key array. For example, the following construct would show the result of a given string sort:

```
FOR I=1TON:PRINT A$(AZ(I)):NEXTI
```

Since the sort key array elements should be in parallel correspondence with the associated object records, the A%() array would also provide the desired sequence index for those object records.

Before any sort task, the A%() array must be initialized to reflect the "original" sequence of the object records.

```
VIZ: FOR I=1TON:A%(I)=I:NEXTI
```

B. Main Program Initialization Requirements

The A$(),A() and A%() arrays must be dimensioned to accomodate the maximum number of records to be sequenced by the program.

```
Example: NM=100:DIM A$(NM),A(NM),A%(NM)
```

C. Calling Conventions

1. Entry argument set up

The set of "sort key" values must be constructed from the set of object records to be sequenced and must be placed in the appropriate A$() or A() array in a one to one correspondence with the "original" record sequence.

Example #1: String sort

```
REM R$(RI,1) = NAME; NR = # OF RECORDS; RI = RECORD INDEX
FOR RI=1TONR:A$(RI)=R$(RI,1):NEXTRI
```

Example #2: Numeric sort

```
REM R$(RI,6) = TEMPERATURE
FOR RI=1TONR:A(RI)=VAL(R$(RI,6)):NEXTRI
```

The number of records to be sequenced must be placed in "N". The function code corresponding to the desired sequencing task must be placed in "A". The "record index" array A%() must be initialized to the original unsorted sequence of the object records prior to the initial sort subroutine call for a given sequencing task. However, the A%() array should not be disturbed prior to successive calls involved in a multiple key sequencing task.

2. Return Argument Use

The sort subroutine will return a (-1) in "A" if the function code is not valid. The A%() array will contain the list of indexes that defines the the sorted sequence of the object records as a result of the "last" sort subroutine call, as illustrated by the following examples.

Example #1: Single field string records

```
FOR RI=1TONR:PRINT A$(A%(RI)):NEXTRI
```

Example #2: Multiple field records

```
FOR RI=1TONR:FORJ=1TONF:REM NF = # OF FIELDS
PRINT R$(A%(RI),J);" ";:NEXTJ:PRINT:NEXTRI
```

APPENDIX B

```
1 REM SIMPLE 'SINGLE FIELD' RECORD SORTS
2 REM
10 POKE2888,0:NM=100:DIMA$(NM),A(NM),A%(NM)
20 GOTO90
30 POKE2888,27:END
90 I=0:L$="STRING"
95 INPUT"STRING(0) OR NUMERIC(1) SORT";F1%:IFF1%THENL$="NUMBER"
100 I=I+1:PRINTI;L$;:INPUTA$(I):IFA$(I)<>""GOTO100
110 N=I-1:IFF1%THENFORI=1TON:A(I)=VAL(A$(I)):NEXTI
120 INPUT"ASCENDIN(0) OR DESCENDING(1) ORDER";F2%:A=F2%+2*F1%
121 FORI=1TON:A%(I)=I:NEXTI
122 GOSUB10000:IFA<0THENPRINT"FUNCTION CALL ERROR":GOTO30
125 PRINT:PRINT"UNSORTED";TAB(30)"SORTED":PRINT
127 IF F1%=0 GOTO 140
130 FORI=1TON:PRINTA(I);TAB(30)A(A%(I)):NEXTI:PRINT:GOTO30
140 FORI=1TON:PRINTA$(I);TAB(30)A$(A%(I)):NEXTI:PRINT:GOTO30
9999 REM BUBBLE SORTING SUBROUTINE
10000 K=1:IFNOT3ANDATHENA=-K:RETURN
10020 FORI=KTON-K:FORJ=I+KTON
10030 ONA+KGOSUB10050,10060,10070,10080
10040 ONK+NOTA%GOSUB10090:NEXTJ,I:RETURN
10050 A%=A$(A%(I))>A$(A%(J)):RETURN
10060 A%=A$(A%(I))<A$(A%(J)):RETURN
10070 A%=A(A%(I))>A(A%(J)):RETURN
10080 A%=A(A%(I))<A(A%(J)):RETURN
10090 A%=A%(I):A%(I)=A%(J):A%(J)=A%:RETURN
```

APPENDIX C

```
1 REM SIMPLE MULTIPLE FIELD RECORD SORTS
2 REM VIZ; USING ONLY ONE FIELD FOR SORT KEY
10 NMAX=50:FMAX=10
15 POKE2888,0:DIM A$(NM),A(NM),A%(NM)
20 DIM NF$(FM),REC$(NM,FM)
30 READ NF:FORI=1TONF:READ NF$(I):NEXTI
40 READ NR:FORI=1TONR:FORJ=1TONF:READ REC$(I,J):NEXTJ,I
50 GOTO100
99 POKE2888,27:END
100 INPUT"ENTER SORT KEY FIELD#";SK%:IFSK%<1ORSK%>NFGOTO100
110 INPUT"STRING(0) OR NUMERIC(1) SORT";F1%
115 IFF1%AND-2GOTO110
120 INPUT"ASCENDING(0) OR DESCENDING(1) ORDER";F2%
125 IFF2%AND-2GOTO120
126 FORI=1TONR:A%(I)=I:NEXTI
130 INPUT"DO YOU WANT TO SEE UNSORTED SEQUENCE";A$
140 IFLEFT$(A$,1)="Y"THENGOSUB500
200 FORI=1TONR
210 IF F1% GOTO 220
215 A$(I)=REC$(I,SK%):GOTO 230
220 A(I)=VAL(REC$(I,SK%))
230 NEXTI
250 A=F2%+2*F1%:N=NR:GOSUB10000
260 GOSUB500:GOTO99
500 PRINT:PRINT"REC# ";:FORI=1TONF:PRINTNF$(I);" | ";:NEXTI:PRINT:PRIN
510 FORI=1TONR:PRINTA%(I);TAB(4);:FORJ=1TONF
520 PRINT REC$(A%(I),J);" | ";:NEXTJ
530 PRINT:NEXT:INPUTA$:RETURN
2000 DATA 7:REM # OF FIELDS IN EACH RECORD
2001 DATA NAME
2002 DATA STREET
2003 DATA CITY
2004 DATA STATE
2005 DATA ZIP CODE
2006 DATA HEIGHT
2007 DATA WEIGHT
3000 DATA 5:REM # OF DATA RECORDS
3010 DATA "SMITH,JOHN J.",123 LONGLY,BOULDER,MASS
3015 DATA 44555,72,160
3020 DATA "OSBORN,ROGER T.",333 LAKEPORT,WATERBURY,NEW YORK
3025 DATA 32144,78,210
3030 DATA "THORNSBY, MARY T.",434 BROWN,SYRACUSE,CONNETICUT
3035 DATA 56344,62,125
3040 DATA "CRACKER,CHRIS P.",65645 CROSS,MUDDY WATERS,MICH
3045 DATA 48656,60,180
3050 DATA "JONES,ROBERT M.",9876 SHORE DR.,BENT LAKE,IDAHO
3055 DATA 56447,78,300
10000 K=1:IFNOT3ANDATHENA=-K:RETURN
10020 FORI=KTON-K:FORJ=I+KTON
10030 ONA+KGOSUB10050,10060,10070,10080
10040 ONK+NOTA%GOSUB10090:NEXTJ,I:RETURN
10050 A%=A$(A%(I))>A$(A%(J)):RETURN
10060 A%=A$(A%(I))<A$(A%(J)):RETURN
10070 A%=A(A%(I))>A(A%(J)):RETURN
10080 A%=A(A%(I))<A(A%(J)):RETURN
10090 A%=A%(I):A%(I)=A%(J):A%(J)=A%:RETURN
```

APPENDIX D

```
1 REM COMPLEX MULTIPLE FIELD RECORD SORTS
2 REM VIZ; USING MULTIPLE FIELDS AS "SORT KEY"
3 REM
10 NMAX=50:FMAX=10
15 POKE2888,0:DIM A$(NM),A(NM),A%(NM)
20 DIM NF$(FM),REC$(NM,FM),TF$(FM),SK%(FM)
21 DIM SO%(FM)
30 READ NF:FORI=1TONF:READ NF$(I),TF$(I):NEXTI
40 READ NR:FORI=1TONR:FORJ=1TONF:READ REC$(I,J):NEXTJ,I
50 GOTO100
99 POKE2888,27:END
100 INPUT"ENTER # OF SORT KEY FIELDS";NK%:IFNK%<1ORNK%>NFGOTO100
102 FORI=1TONK%
104 PRINT"ENTER KEY FIELD#";I;:INPUTSK%(I)
106 IFSK%(I)<1ORSK%(I)>NFGOTO104
107 IFI=1GOTO120
108 FORJ=1TOI-1:IFSK%(J)=SK%(I)THENPRINT"DUPLICATE"GOTO100
109 NEXTJ
120 INPUT"ASCENDING(0) OR DESCENDING(1) ORDER";F2%
125 IFF2%AND-2GOTO120
126 SO%(I)=F2%:NEXTI
130 FORI=1TONR:A%(I)=I:NEXTI
140 INPUT"DO YOU WANT TO SEE UNSORTED SEQUENCE";A$
150 IFLEFT$(A$,1)<>"Y"GOTO200
160 GOSUB500
200 FORSK=NK%TO1STEP-1
210 F2%=SO%(SK)
220 F1%=0:IFTF$(SK%(SK))="N"THENF1%=1
225 FORI=1TONR
227 IFF1%GOTO240
230 A$(I)=REC$(I,SK%(SK)):NEXTI:GOTO250
240 A(I)=VAL(REC$(I,SK%(SK))):NEXTI
250 A=F2%+2*F1%:N=NR
255 GOSUB10000
260 GOSUB500:NEXTSK:GOTO99
500 PRINT:PRINT"REC ";:FORI=1TONF:PRINTNF$(I);" : ";:NEXTI:PRINT:PRINT
510 FORI=1TONR:PRINTA%(I);TAB(4);:FORJ=1TONF
520 PRINT REC$(A%(I),J);" : ";:NEXTJ
530 PRINT:NEXTI:INPUTA$:RETURN
10000 K=1:IFNOT3ANDATHENA=-K:RETURN
10020 FORI=KTON-K:FORJ=I+KTON
10030 ONA+KGOSUB10050,10060,10070,10080
10040 ONK+NOTA%GOSUB10090:NEXTJ,I:RETURN
10050 A%=A$(A%(I))>A$(A%(J)):RETURN
10060 A%=A$(A%(I))<A$(A%(J)):RETURN
10070 A%=A(A%(I))>A(A%(J)):RETURN
10080 A%=A(A%(I))<A(A%(J)):RETURN
10090 A%=A%(I):A%(I)=A%(J):A%(J)=A%:RETURN
```

APPENDIX D (cont)

```
20000 DATA 7:REM # OF FIELDS PER DATA RECORD
20001 DATA NAME,S
20002 DATA CITY,S
20003 DATA ZIP CODE,B
20004 DATA COMPUTER,S
20005 DATA # FLOPPIES,B
20006 DATA #K RAM,N
20007 DATA RATING(+/-),N
30000 DATA 20:REM # OF DATA RECORDS
30001 DATA"BONE,REEVES",BELLEVILLE,48111,C4P,1,30,3
30002 DATA"BAYLOR,BILLY",LIVONIA,48152,C4P,1,24,0
30003 DATA"ZAPPO,ZEPPY",TROY,48084,C1P,0,8,-5
30004 DATA"BOLSEN,DODGER",WALLED LAKE,48088,CAP,2,48,5
30005 DATA"BARTIN,BILLY",STERLING HEIGHTS,48077,C4P,2,48,2
30006 DATA"SCAPEL,DIRTY",DRAYTON PLAINS,48020,C4P,2,24,3
30007 DATA"BATTLE,BERRY",ANN ARBOR,48013,C3P,2,48,5
30008 DATA"JOHNS,JIMMY",PLYMOUTH,48170,C8P,2,48,5
30009 DATA"REED,RICKY",GROSSE ILSE,48138,C8P,2,24,3
30010 DATA"BAYLOR,SALLY",BIRMINGHAM,48009,C4P,2,48,2
30011 DATA"CANTELLA,CHARLES",TRENTON,48183,C8P,2,24,-4
30012 DATA"GRAM,GRANNY",NORTHVILLE,48167,C4P,2,24,6
30013 DATA"SHOWMAN,SAMMY",EAST DETROIT,48021,C1P,0,8,-3
30014 DATA"BAMBOO,WALLY",SOUTHFIELD,48075,C4P,2,24,1
30015 DATA"BROWNSBERRY,BERRY",NORTHVILLE,48167,C4P,0,8,1
30016 DATA"BEECH,SANDY",PLYMOUTH,48170,SBD,0,?,-5
30017 DATA"FORTHRITE,MARY",DEXTER,48130,C4P,1,24,0
30018 DATA"GROOVE,INA",PARMA,49269,?,?,?,-6
30019 DATA"JABSON,JABBY",HAMTRAMCK,48212,SBD,0,8,-2
30020 DATA"MATCHLESS,MILLY",FLINT,48503,3/4,2,48,5
```

APPENDIX E

```
1 REM SIMPLE DISK FILE RECORDS SORT EXAMPLE
2 REM VIZ: BUILD SORT KEYS FOR 'RANDOM ACCESS' FILE
3 DIMA$(100),A(100),A%(100)
4 POKE2888,0
5 POKE2972,13:POKE2976,13
6 SP$="          ":SP$=SP$+SP$+SP$
8 INPUT"ENTER DATA FILE NAME";FILE$
10 DISK OPEN,6,FILE$
12 NL=25
15 POKE12076,5:POKE12042,96
16 INPUT"BUILD NEW FILE(1), APPEND(2), RETRIEVE(3)";X
17 ON X GOTO 20,200,300
18 POKE2888,27:POKE2972,58:POKE2976,44:END
20 PRINT#6,LEFT$(SP$,NL):PRINT#6,LEFT$(SP$,5)
25 B=1
35 FOR RN=B TO 100
40 INPUT"NAME";N$
50 IFN$=""GOTO100
55 INPUT"ZIP CODE";Z:PRINT
60 PRINT#6,RIGHT$(SP$+N$,NL)
65 PRINT#6,RIGHT$(SP$+STR$(Z),5)
80 NEXTRN:RN=101
90 PRINT:PRINT"FILE FULL"
100 DISKCLOSE,6:DISKOPEN,6,FILE$
110 PRINT#6,RIGHT$(SP$+"NAME&ZIP FILE",NL)
120 PRINT#6,RIGHT$(SP$+STR$(RN-1),5)
130 DISKCLOSE,6
140 GOTO 18
200 INPUT#6,A$,NR
210 IFNR<100GOTO220
215 PRINT"FILE FULL":DISKCLOSE,6:GOTO18
220 DISK GET,NR+1
230 B=NR+1:GOTO35
300 INPUT#6,A$,NR
302 PRINT:PRINT
305 PRINT"RECORDS IN FILE SEQUENCE":PRINT
310 FOR RN=1TONR
320 INPUT#6,A$(RN),A(RN)
325 PRINTA$(RN);TAB(30)A(RN)
330 NEXTRN
335 INPUTA$
355 N=NR:A=0:FORI=1TON:A%(I)=I:NEXTI:GOSUB10000
360 FORI=1TORN:N%(I)=A%(I):NEXTI
375 PRINT
377 PRINT:PRINT"RECORDS IN NAME SEQUENCE":PRINT
380 FORI=1TONR:RN=N%(I):DISKGET,RN
390 INPUT#6,A$,Z:PRINTA$;TAB(30)Z:NEXTI
395 PRINT:INPUTA$
398 A=2:FORI=1TON:A%(I)=I:NEXTI:GOSUB10000
399 PRINT:PRINT"RECORDS IN ZIP CODE SEQUENCE":PRINT
400 FORI=1TONR:RN=A%(I):DISKGET,RN
410 INPUT#6,A$,Z:PRINTA$;TAB(30)Z:NEXTI
420 DISKCLOSE,6:GOTO18
```

APPENDIX F

BUBBLE SORT SUBROUTINE

```
10000 K=1:IFNOT3ANDATHENA=-K:RETURN
10020 FORI=KTON-K:FORJ=I+KTON
10030 ONA+KGOSUB10050,10060,10070,10080
10040 ONK+NOTA%GOSUB10090:NEXTJ,I:RETURN
10050 A%=A$(A%(I))>A$(A%(J)):RETURN
10060 A%=A$(A%(I))<A$(A%(J)):RETURN
10070 A%=A(A%(I))>A(A%(J)):RETURN
10080 A%=A(A%(I))<A(A%(J)):RETURN
10090 A%=A%(I):A%(I)=A%(J):A%(J)=A%:RETURN
```