INTRODUCTION:

In Chapter 46 of the SAS User's Guide Basics, there is a small note on large data sets. "Sorting Large Data Sets. Occasionally a data set is too large for the sort utility to handle. If your installation uses an IBM or equivalent system sort utility, PROC SORT prints a message explaining the situation when this happens. In the notes are various suggestions on how to calculate the appropriate "$s" or sort size for your data set. However, there are no suggestions on what to do when the size of the data set exceeds the capacity of the system.

Sorting data sets with large numbers of records presents various problems. In larger sets, such as those with more than 1 million records, certain systems will not have the capacity to handle a sort, no matter what capacity or sort size you request. This paper presents a strategy for sorting massive data sets in severely limited environments.

For the purpose of this paper a large set is defined as one in excess of ½ million records. Different systems will approach their limits with larger or smaller numbers of records. The methods presented have been successfully used with a 2.7 million record data set.

WHAT IS PROC SORT?

It may vary from place to place but where I work PROC SORT is one of the most commonly used SAS Institute procedures. PROC SORT takes a data set and rearranges the records according to the ascending or descending values of a chosen variable, or variables. The values can be numeric or alphabetic, so we can sort a data set by both name and age.

PROC SORT is used for three primary reasons:

1. To find a "Top" or "Bottom" group of records.
   Example: Find the top 10 sales people, as ranked by a variable "TOTSAL", in a data set with 500 salespeople.

2. To prepare a data set for later use, with a BY statement in a PROC such as PROC MEAN or PROC FREQ.

3. To arrange a data set in numeric or alphabetic order for display or printing purposes.

There is a fourth reason that PROC SORT is used; habit. Many times beginning, and not so beginning programmers will sort a data set unnecessarily but automatically in order to avoid the SAS error message "DATA SET NOT SORTED."

Planning prior to programming is preached by instructors at all levels, but when dealing with large data sets and sorts, it is planning that will help avoid unneeded sorts. By planning, the programmer can create a strategy tailored to the specific data set that will decrease the overall size of the set, eliminate unneeded variables, and cut down on the demands placed on the system.

LARGE DATA SETS MEAN LARGE PROBLEMS:

The problem with using PROC SORT either necessarily or by habit, is that sorting uses a lot of computer time, memory and hard storage capacity. As the data sets increase in size it becomes possible to exceed the capacity of the system and "kill" the job.

In my department we commonly deal with data sets in excess of ½ million records and occasionally in excess of 1 million. Your system will dictate what is "large".

PROC SORT creates, at least temporarily, a duplicate data set, distinguished from the original and unsorted set only by the order of the records. If you use the OUT=SASdataset option in PROC SORT, the unsorted set remain on the system taking up space. There are times when this is unavoidable, but if you do not need the original data set the OUT=SASdataset option is to be avoided. Omitting this option will cause the sorted set to replace the original data set.

PROC SORT uses up time, both CPU and actual. Computing environments differ, but in most places there are limits on jobs and separate classes dependent in part on CPU time. The computer operators can help you to determine when is the best time to run large jobs but if the time requested is quite large you might find yourself running large jobs in a 3:00 AM Saturday time slot.

Any strategy for sorting massive
sets must aim at cutting down on the storage, memory and time requirements of the sort, or at least manipulate the system so as to allow for the large demands.

MAYBE YOU SHOULDN'T SORT AT ALL:

PROC SUMMARY and TABULATE allow for the use of a CLASS statement in place of a BY statement. Not all of the mathematical formulae are available on these two PROCs but many are.

If you are programming for someone else make sure that the sort is needed. Know your data, the desired results and plan ahead.

SOME RULES TO SORT BY:

1. KNOW WHAT YOU ARE SORTING AND WHY.
   It all goes back to planning. If you are not familiar with the data set it is worth the time to take a sample of the set and look at a few hundred of the records. The time required to print out a small set & "desk edit" is minimal compared to the time required to sort.

2. AVOID MULTIPLE SORTS. Use multiple sort variables when possible, rather than sorting the data set one time for each variable. It is not always feasible but should be considered.

PROC SORT DATA=AA OUT=BB;
BY SEX AGE WEIGHT;

This large sort is more efficient than sorting three times and the variables can each be used independently in later BY statements.

3. DELETE UNEEDED RECORDS. The first goal in dealing with a large data set is to make it smaller. Before starting to write out code, determine if there are any factors that will make a record invalid or unneeded and then by using IF statements in a data set eliminate the the records prior to sorting. For example if you wanted to look at the maternal age of first time mothers in a state population there would be no reason to look at records of males, anybody under or over certain ages or records where age was missing.

DATA ALLSTATE; SET FROM TAPE;
IF SEX='F';
IF AGE>6 AND AGE<65;

The above code eliminates males, young girls and women over child bearing age and also eliminates missing values for the variable AGE because a missing value in the SAS system is smaller than all numbers. This data step could cut out as many as half of the records before the sort.

4 DELETE UNEEDED VARIABLES. Because the sorted set will need to be stored the smaller the set the better. If we are only interested in sales figures there is no need to store the salesperson's age, weight and marital status.

5. SORT ONLY A PORTION OF THE SET.
   If the purpose of the sort is to find the top or bottom portion of a data set it is possible to eliminate some of the records prior to the sort. Here is where a knowledge of the data is essential. By looking at the data the programmer can set a safe "cutoff point" beyond which he or she knows that no records will be of interest. For example if we wanted to find the top 100 salespeople by total sales in a set of 5,000 and a sample of 100 records showed that approximately 5 the total sales were below $10,000, we could safely delete all records where total sales were less than $10,000 and sort a much smaller set.

6. ALWAYS SAVE YOUR SORTED SET.
   Few things are more frustrating than successfully sorting a massive data set only to have the program "bomb" because of a missing semi-colon during the last print statement. Save the sorted set to a disk or a tape so that you do not have to resort if something goes wrong.

HOW BIG IS A SORT?

Sort routines work by making comparison of selected variables a rearranging the records according to the values of the variables. Depending on the system the sort procedure will vary slightly but it is possible to estimate the number of comparisons that will be needed for a data set of known size. The formula is:

Approx. Comparisons = Observations * LOG(Observations).

A 10,000 record data set, sorted by one variable, will require about 40,000 comparisons, a 100,000 record data set will require about 500,000 and the 2.7 million record data set I sorted would have required about 17.3 million comparisons; enough to overload virtually any system, if sorted normally.

THE SUM OF THE PARTS IS LESS THAN THE WHOLE:

If after deleting records and dropping variables the data set is still too large for the system there is an alternative method of sorting. I have named it the "Pseudo-Sort" for lack of a better term. The principal behind the
Pseudo-Sort is that any data set can be divided into ordered subsets, and these subsets can be sorted independently. The strategy for a Pseudo-Sort follows (see Figure #1 for a sample code):

1. "Clean" the data set, eliminating unneeded records and variables.
2. In a Data Step, use multiple outputs to create subsets. These subsets must be divided by values of variables that would have been used in the BY statement of a normal PROC SORT.
   For instance, if we are sorting a data set of all schools in the country by student population (we will call the variable SPOP) we could divide the data set into subsets as follows:
   ```
   DATA SMALL MEDIUM LARGE;
   SET TAPE.SCHOOLS;
   IF SPOP<100 THEN OUTPUT SMALL;
   ELSE IF SPOP>99 AND SPOP<500 THEN OUTPUT MEDIUM; ELSE OUTPUT LARGE;
   ```
   The subsets are internally unsorted but each subset contains within it only those values larger than the previously created set and lower than the next set. In this way we have presorted the data.
3. Use PROC SORT (with NO OUT=...) to sort each subset.
   ```
   PROC SORT DATA=SMALL; BY SPOP;
   PROC SORT DATA=MEDIUM; BY SPOP;
   PROC SORT DATA=LARGE; BY SPOP;
   ```
   The highest value in the subset SMALL will be the value immediately below the lowest value in the subset MEDIUM. For an example of how the records move see Figure #2.
4. Use a SET command in a DATA SET to create a new and sorted large data set.
   ```
   DATA NOWSORT;
   SET SMALL MEDIUM LARGE;
   ```
   Note the Lack of any BY statement. Use PROC DATASETS to get rid of the subsets that are now only taking up room.
   The new large/sorted set perfectly mimics how the original data set would have looked if the original set had been sorted with a single PROC SORT. The subsets, because of their smaller size (variables and records) can be sorted on limited systems and during busy times of day. They can also be sorted as separate programs if necessary, allowing for a number of small programs to be run when a single large program could not be scheduled.
5. The new large set NOWSORT, can be used as if it had been sorted regularly. This method has been tested with PROC FREQ, MEANS, SUMMARY and UNIVARIATE with BY statements.
   Pseudo-Sort requires more lines of code than a standard PROC SORT and if used unnecessarily (on data sets small enough to be sorted normally) will consume more computer resources than PROC SORT.
   Planning, sampling of data to be sorted, elimination of unneeded records and variables and the use of a Pseudo-Sort when all else fails, will allow the SAS System programmer to sort data sets of virtually any size.

* * * * *

Figure 1.

```
DATA TESTSET (KEEP=CNTY AGE WEIGHT HEIGHT SEX);
SET TAPE.CENSUS;
IF CNTY=. OR AGE=. OR SEX=.
   THEN DELETE;
IF AGE<12;
DATA TEST1 TEST2 TEST3; SET TESTSET;
   IF AGE<23 THEN OUTPUT TEST1; ELSE IF AGE>22 AND AGE<42 THE OUTPUT TEST2; ELSE IF AGE>41 THEN OUTPUT TEST3;
PROC SORT DATA=TEST1; BY AGE;
PROC SORT DATA=TEST2; BY AGE;
PROC SORT DATA=TEST3; BY AGE;
DATA SAVET.COMBINED;
   SET TEST1 TEST2 TEST3;
PROC MEANS; DATA=SAVET.COMBINED;
   BY AGE;
   VAR WEIGHT;
   OUTPUT OUT=NEWSET N=COUNT;MEAN=M;
PROC PRINT TITLE 'PSEUDEO-SORT';
```

Notes:

The "IF AGE>41...." statement is not needed as all ages left at this point will be greater than 41. It has been placed in the program for illustrative reasons.

If the subsets were very large I would have saved them to a permanent location.
Figure 2.

Record Movement during Pseudo-Sort.

<table>
<thead>
<tr>
<th>Unsorbed Subsets Data Set</th>
<th>Sorted Subsets Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Alice</td>
</tr>
<tr>
<td>Bertha</td>
<td>Bertha</td>
</tr>
<tr>
<td>Charles</td>
<td>Charles</td>
</tr>
<tr>
<td>Iris</td>
<td>Donna</td>
</tr>
<tr>
<td>Edna</td>
<td>Donna</td>
</tr>
<tr>
<td>Frank</td>
<td>Frank</td>
</tr>
<tr>
<td>Donna</td>
<td>Frank</td>
</tr>
<tr>
<td>Henry</td>
<td>Henry</td>
</tr>
<tr>
<td>George</td>
<td>Henry</td>
</tr>
</tbody>
</table>

* * * * * * * *

Direct Questions To..David S. Rubin
Empire Blue Cross and Blue Shield
P.O. Box 8650 Albany, NY 12208
518/471-5173

SAS and the SAS System are registered trademarks of SAS Institute Inc. Cary, NC, USA.