Expanding the Functionality of Min()/Max(): Finding Tied Minimum or Maximum Values and Their Contributing Variables

Kent Nassen, Pfizer Global Research and Development, Ann Arbor, MI
Subrahmanyam Pilli, Pfizer Global Research and Development, Ann Arbor, MI
Luai Alzoubi, Pfizer Global Research and Development, Ann Arbor, MI

ABSTRACT
SAS provides functions to find the minimum or maximum value within a variable or across a set of variables. But what happens when you need to determine, across several variables in a record, which variable is the minimum or, if there are ties at the minimum value, which variables had the tied value? Three SAS macros are presented and compared, which go beyond the functionality of the MIN() or MAX() functions by determining the minimum or maximum value or tied values, as well as indicating the source variable(s) that account for the minimum or maximum value(s). Methods demonstrated are macro loops, arrays, and PROC SQL. These macros were originally developed to produce a derived data set for a clinical Positron Emission Tomography (PET) cardiovascular study where comparisons across regions of the heart were needed and tied minimums or maximums in the data would be handled differently, depending on the region(s) with the minimum or maximum value(s).

INTRODUCTION
Finding a minimum or maximum value from within a group of variables is quite easy. Just use the MIN() or MAX() functions. However, we needed to find the minimum or maximum value as well as any tied minimums or maximums and identify which variables accounted for those, within a record or observation. In the context of tied minimum or maximum values, the MIN() and MAX() functions alone return a single value. They tell you nothing about which variable contributed the minimum or maximum value, nor if there are tied minimum or maximum values. We realized that we were not getting all the information we needed to further process our input data from MIN() and MAX(). The results of our experiments to provide this additional information are described. Three SAS macros are presented and demonstrated. Efficiency considerations are also explored.

CONCEPTS AND CODE
Our purpose was to find tied minimum or maximum values from a set of user-specified variables in a SAS data set, at a case or observation level. It was necessary not only to find the minimum or maximum value, which a MIN() or MAX() function could easily do, but also to find tied values, and indicate the source variable(s). Based on this information, further data manipulation could be performed depending on which source variable(s) contributed to the ties. Our design goals included not altering the original values in the input data and keeping the minimum or maximum value information in the output data set.

Our first example of a working algorithm in macro code, findties, is presented in Listing 1. It uses nested macro DO loops to find the minimums or maximums by comparing the values of the source variables. This macro contains comments, input error checking, and code for generating debugging output which make the macro more general and understandable, but longer.

The part that actually computes the minimum or maximum values and generates the output variables is eight lines of macro DO loop processing. Those nested loops generate a series of IF/ELSE statements that compare the desired variable values and retain only those values that are minima or maxima. The user specifies whether to find minima or maxima and the variables to be compared when calling the macro. The information about which variables are minimums or maximums is kept in a series of work variables. These are set to the minimum or maximum value, or to missing if the corresponding variable does not contain a minimum or maximum value. Missing input data is not considered, since by definition, a missing value should not be compared (it is unknown).

The advantages in using macro do loops to create SAS code are that the amount of macro source code is greatly reduced, and the code is generalizable to accommodate more than a fixed number of variables by increasing the upper bounds of the macro DO loops dynamically. On the other hand, as the number of variables being processed increases, the number of SAS statements generated increases as the combinations of the number of variables, n, choose 2, since we are making paired comparisons. That is, \(C_2^n = \frac{n!}{2!(n-2)!}\). For the size of problem we faced (4 variables), this was not a problem since only \(C_2^n = 6\) sets of SAS IF/ELSE comparison statements were generated, executed, and logged.

Listing 1.

************************************************
Program: findties.sas
Developer: Kent Nassen
Date: 3/26/01
Purpose: Find the tied value(s) from a set of n variables. Returns the min or max values in R1-Rn. R1-Rn indicate the variable(s) with the minimum or maximum value(s). R1-Rn are assigned based on the srcvars macro var in the macro call. Order of the variables is important if you wish to
know, for example, which variable is the minimum or maximum, and use that later in the program.

Operating System: Windows NT V4 SP6
(should be portable to Unix)

Macro variables used:
- dsn --> data set name (the data set to process)
- type --> set to MIN for finding min, to MAX for finding max (the equal case is handled by the macro) DEFAULT: MIN
- srcvar --> the variables to process (space delimited list)
- numvars --> the number of variables to be processed (computed by the macro)

i and j are loop counter variables used in several places
r&i, r&j --> the ith and jth variables from &srcvars
ltgt --> internal macro var to set the comparison based on type
debug --> DEFAULT: 0 (no debugging output). Sets amount of debugging output (1=list the ties, 2=ties and full listing).

Output file list: input data set
Output file location: work or permanent data set determined by &dsn variable

Variables created: R1-Rn
Submission method: interactive or batch SAS

Example of usage:
%findties(tran,srcvars=superior septal lateral inferior,debug=2);

*****************************************
%macro findties(dsn,type=MIN,srcvars=,debug=0);
options mprint symbolgen mlogic;

data &dsn;
  set &dsn;
  /* Find number of variables in &srcvars*/
  %let numvars=1;
  %do %while(%scan(&srcvars, ,') ne );
    %let numvars=%eval(&numvars+1);
  %end;
  %let numvars=%eval(&numvars-1);
  /* The type parameter must be either MIN or MAX */
  %let ltgt=LT; * MIN is the default *
  %if %upcase(&type)=MAX %then %do;
    %let ltgt=GT;
  %end;
  %else %if %upcase(&type)=MIN %then %do;
    put "ERR" " OR The type parameter must be set to either MIN or MAX. ";
    stop;
  %end;
  /* Must have at least two variables for this algorithm */
  %if %eval(&numvars) < 2 %then %do;
    put "ERR" " OR This macro needs at least two variables defined in srcvars. ";
    stop;
  %end;
  /* Set R1-Rn to the specified variables. Rx <-- Variable x */
  %do i=1 %to &numvars;
    r&i=%scan(&srcvars,&i,' ');
  %end;
  /* Find the minimum or maximum value(s) by setting R1-Rn equal to missing for values which are not minima/maxima. */
  %do i=1 %to &numvars;
    %do j=&i+1 %to &numvars;
      if r&i>. and r&j>. then do;
        /* Do not compare missing values */
        if r&i &ltgt r&j then r&j=.;
        /* r&i is the current min/max */
        else if r&i ne r&j then r&i=.;
        /* r&i is not a min nor max nor equal */
      end;
    %end;
  %end;
run;

/* Print diagnostic output (always print summary of tied values if debug output is requested, print more if debug value is higher than 1) */
%findties(dsn,type=MIN,srcvars=,debug=0);
  %do i=1 %to &numvars;
    %let i=%eval(&numvars+1);
    if r&i>. and r&j>. then do;
      /* Do not compare missing values */
      if r&i &ltgt r&j then r&j=.;
      /* r&i is the current min/max */
      else if r&i ne r&j then r&i=.;
      /* r&i is not a min nor max nor equal */
    end;
  %end;
run;
title "Cases with tied &type values in R1-R&numvars"
run;

%if &debug gt 1 %then %do;
proc print data=&dsn;
title "All &dsn data with R1-R&numvars computed (type=&type)"
run;
%end;
%end;
%mend findties;

Listing 2 illustrates the usage of the findties macro presented in Listing 1. Four variables are processed for minimums in the first findties call, and for maximums in the second findties call.

Listing 2.

options mprint mlogic symbolgen ls=64;
%include 'findties.sas';

data test;
  input var1 var2 var3 var4;
cards;
  10 10 10 10
  10 12 13 14
  12 13 12 13
  12 13 12 14
  15 12 19 19
  8 9 10 8
  8 9 8 10
  22 24 25 25
  19 99 18 18
  . 18 . 18
; run;

proc print data=test;
title 'test data'
run;

/* Find ties at the minimum values (default type) and little debug output */
%findties(test,srcvars=var1 var2 var3 var4,debug=1);

/* Find ties at the maximum value and get debug output */
%findties(test,type=MAX,srcvars=var1 var2 var3 var4,debug=2);
run;

The output of the Listing 2 code is provided in Listing 3, which shows how the macro computes and indicates the minimum or maximum values in the output data set. VAR1-VAR4 are the original variables. Variables R1-R4 are created to retain the minimum or maximum values and to correspond to VAR1-VAR4. For example, the last PROC PRINT in Listing 3 shows maximums being determined. Variables R1-R4 indicate the variable(s) that account for the maximum(s) and are set to missing when the original value is not a maximum. Single or tied maximums are found. Missing values in original variables are ignored. The original data in VAR1-VAR4 is left unchanged.

Listing 3.

<table>
<thead>
<tr>
<th>OBS</th>
<th>VAR1</th>
<th>VAR2</th>
<th>VAR3</th>
<th>VAR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>13</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>12</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>24</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>9</td>
<td>19</td>
<td>99</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>.</td>
<td>18</td>
<td>.</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>.</td>
<td>22</td>
<td>25</td>
<td>.</td>
</tr>
</tbody>
</table>

Cases with tied MIN values in R1-R4

<table>
<thead>
<tr>
<th>OBS</th>
<th>VAR1</th>
<th>VAR2</th>
<th>VAR3</th>
<th>VAR4</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>12</td>
<td>18</td>
<td>19</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>10</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>24</td>
<td>25</td>
<td>25</td>
<td>.</td>
<td>.</td>
<td>25</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>19</td>
<td>99</td>
<td>18</td>
<td>18</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>.</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>.</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>.</td>
<td>22</td>
<td>25</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>25</td>
</tr>
</tbody>
</table>

Cases with tied MAX values in R1-R4

<table>
<thead>
<tr>
<th>OBS</th>
<th>VAR1</th>
<th>VAR2</th>
<th>VAR3</th>
<th>VAR4</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>13</td>
<td>12</td>
<td>13</td>
<td>.</td>
<td>13</td>
<td>.</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>24</td>
<td>25</td>
<td>25</td>
<td>.</td>
<td>.</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>.</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>.</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>.</td>
<td>22</td>
<td>25</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>25</td>
</tr>
</tbody>
</table>

All test data with R1-R4 computed (type=MAX)

<table>
<thead>
<tr>
<th>OBS</th>
<th>VAR1</th>
<th>VAR2</th>
<th>VAR3</th>
<th>VAR4</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>12</td>
<td>18</td>
<td>19</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>10</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>24</td>
<td>25</td>
<td>25</td>
<td>.</td>
<td>.</td>
<td>25</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>19</td>
<td>99</td>
<td>18</td>
<td>18</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>.</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>.</td>
<td>18</td>
<td>.</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>.</td>
<td>22</td>
<td>25</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>25</td>
</tr>
</tbody>
</table>

OTHER APPROACHES

There is often more than one way to do the same thing in SAS. The same minimum- or maximum-finding operations can be also be accomplished using arrays as shown in Listing 4, macro findtie2. This code is also generalized.
and quite short. This algorithm uses the result of the usual min() or max() functions to find the ties. The data set output from this and the third macro approach is the same as for the first example.

Listing 4.

******************************************************************************
Program: findtie2.sas
SAS Version 6.12
Developer Subra Pilli
Date: 9/10/01
Purpose Find the tied value(s) from a set of n variables. Returns the min or max values in R1-Rn. R1-Rn indicate the variable(s) with the minimum or maximum value(s).

Example of usage:
%findtie2(tran,srcvars=superior septal lateral inferior,debug=2);
******************************************************************************;
%macro findtie2(dsn,type=MIN,srcvars=, debug=0);
options mprint symbolgen mlogic;
/* Find number of variables in &srcvars */
%let numvars=1;
%do %while(%scan(&srcvars,&numvars,' ') ne )
%let numvars=%eval(&numvars+1);
%end;
%let numvars=%eval(&numvars-1);
/* The type parameter must be either MIN or MAX */
%if %upcase(&type)^=MIN and
 %upcase(&type)^=MAX %then %do;
 put "ERR" OR The type parameter must be set to either MIN or MAX. ";
 stop;
%end;
/* Must have at least two variables for this algorithm */
%if %eval(&numvars) < 2 %then %do;
 put "ERR" "OR This macro needs at least two variables defined in srcvars. ";
 stop;
%end;
/* Use arrays to find the minimum or maximum value or tied values */
data &dsn(drop=i x);
 set &dsn;
x=&type(of &srcvars);
array b[&numvars] r1-r&numvars;
do i=1 to &numvars;
 if x=a[i] then b[i]=a[i];
 else b[i]=.;
end;
run;
/* Print diagnostic output (always print summary of tied values if debug output is requested, print more if debug value is higher than 1) */
%if (%eval(&numvars) > 1 and &debug gt 0)
 %then %do;
 /* These cases have tied values (sum the number of tied values using the sign function—prevent a return of -1 by using the abs function, in case there are negative minima/maxima). */
 proc print data=&dsn;
 where sum( %do i=1 %to &numvars;
 sign(abs(r&i))
 %if &i<&numvars %then %do;
 ,
 %end;
%end;
) > 1; /* more than one minimum or maximum means ties */
 title "Cases with tied &type values in R1-R&numvars";
run;
%if &debug gt 1 %then %do;
 proc print data=&dsn;
 title "All &dsn data with R1-R&numvars computed (type=&type)";
%end;
%end;
%mend findtie2;

A third method of producing the same results can be generated using PROC SQL as shown in Listing 5. Although a bit longer in terms of actual code, it is generalized in terms of number of variables that may be processed. Here the PROC SQL min or max functions are used.

Listing 5.

******************************************************************************
Program: findtie3.sas
SAS Version 6.12
Developer Subra Pilli and Kent Nassen
Date: 9/10/01
Purpose Find the tied value(s) from a set of n variables. Returns the min or max values in R1-Rn. R1-Rn indicate the variable(s) with the minimum or maximum value(s).
Example of usage:
%findtie3(tran, srcvars=superior septal lateral inferior, debug=2);
**************************************************************;
%macro findtie3(dsn, type=MIN, srcvars=, debug=0);
  options mprint symbolgen mlogic;
  /* Find number of variables in &srcvars */
  %let numvars=1;
  %do %while(%scan(&srcvars,&numvars,' ') ne );
    %let numvars=%eval(&numvars+1);
  %end;
  %let numvars=%eval(&numvars-1);
  /* The type parameter must be either MIN or MAX */
  %if %upcase(&type)^=MIN and %upcase(&type)^=MAX %then %do;
    put "ERR" "OR The type parameter must be set to either MIN or MAX.";
    stop;
  %end;
  /* Must have at least two variables for this algorithm */
  %if %eval(&numvars) < 2 %then %do;
    put "ERR" "OR This macro needs at least two variables defined in &srcvars.";
    stop;
  %end;
  proc sql;
    create table &dsn as select *,
      &type(%do i=1 %to &numvars-1;
        %scan(&srcvars,&i,' ')
        %end;
      %scan(&srcvars,&numvars,' '))
      as &type.x,
      %do k=1 %to &numvars-1;
        case
          when %scan(&srcvars,&k,' ') =
            &type(%do l=1 %to &numvars-1;
              %scan(&srcvars,&l,' ')
              %end;
            %scan(&srcvars,&numvars,' ')
            )
            then %scan(&srcvars,&k,' ')
          else .
        end as r&k,
      %end;
    %end;
  %mend findtie3;

EFFICIENCY CONSIDERATIONS
For small data sets and a relatively small number of variables, any of these macros should perform adequately. For larger data sets, however, there may be significant differences in run times.

A simple benchmark test of each macro was run on two different systems (PC and Unix) using a data set of 1,045,800 cases of test data in six variables. Two hundred runs of each method were generated with run times and memory usage provided by the FULLSTIMER SAS option. There were 10,084 cases with tied minimum values (~1% of the total number of cases).

The PC system used for these tests was a single-user machine with a Pentium III 733 MHz CPU and 256 MB RAM running PC SAS 6.12 on Windows NT4 SP6. A batch file was used to submit each run and SAS was started and stopped for each run. The SAS ran consisted of a single call to the macro, computing tied minimum...
values, with no debugging information. The times recorded represent only the running of the data step or proc within the macro that computed the minimums. Other data steps and procs were not included in these times. Table 1 compares the mean run times and memory usage of the three macros under PC SAS.

Table 1. Times & Memory Usage for PC-SAS

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Time (seconds)</th>
<th>Std Dev</th>
<th>Min, Max</th>
<th>Memory (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO Loops</td>
<td>12.38</td>
<td>3.26</td>
<td>10.18, 27.53</td>
<td>5131264</td>
</tr>
<tr>
<td>Arrays</td>
<td>13.76</td>
<td>4.44</td>
<td>9.88, 33.03</td>
<td>5127168</td>
</tr>
<tr>
<td>Proc SQL</td>
<td>17.13</td>
<td>3.01</td>
<td>15.43, 30.27</td>
<td>5505536</td>
</tr>
</tbody>
</table>

n=600 each method

For Unix, a similar procedure as for the PC SAS benchmarking was used. The system used was an HP 9000/800 two-PA-RISC processor system with 4096 MB RAM running HP-UX 11.00 and SAS 6.12. Here, the FULLSTIMER option gives more detailed timing information, splitting the run time into Real, User CPU, and System CPU times. The time most useful for our purposes is the Real Time statistic. It is the clock time taken to run the data step or proc. These jobs were run overnight to minimize the number of active users and accompanying competition for CPU time that is present during the day.

Table 2. Times for Unix

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Real Time (seconds)</th>
<th>Std Dev</th>
<th>Min, Max</th>
<th>Memory (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO Loops</td>
<td>4.01</td>
<td>0.17</td>
<td>3.86, 5.73</td>
<td></td>
</tr>
<tr>
<td>Arrays</td>
<td>7.19</td>
<td>0.12</td>
<td>7.09, 9.16</td>
<td></td>
</tr>
<tr>
<td>Proc SQL</td>
<td>11.21</td>
<td>0.12</td>
<td>11.10, 12.67</td>
<td></td>
</tr>
</tbody>
</table>

n=600 each method

Table 2. Times for Unix

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean User CPU Time (seconds)</th>
<th>Std Dev</th>
<th>Min, Max</th>
<th>Memory (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO Loops</td>
<td>2.90</td>
<td>0.07</td>
<td>2.83, 3.34</td>
<td></td>
</tr>
<tr>
<td>Arrays</td>
<td>6.05</td>
<td>0.03</td>
<td>6.00, 6.14</td>
<td></td>
</tr>
<tr>
<td>Proc SQL</td>
<td>10.13</td>
<td>0.04</td>
<td>10.07, 10.77</td>
<td></td>
</tr>
</tbody>
</table>

n=600 each method

This machine is a multi-user system and there was no way to keep other users from running processes or to keep system automated processes, such as backups, from running, that might slow down the SAS job at any given point during the timing runs. In order to put each method at an equal disadvantage relative to time-dependent system activity effects, the run order of each method was varied so that every method was tried in each possible order—first, second or third. Thus, 600 runs of each method were generated.

Which of the three is easiest to modify and maintain?
In terms of ease of maintenance and modification, the arrays algorithm might be the easiest to understand with few variables and operations to comprehend. The macro DO loops algorithm is probably easiest to understand if you trace through the loops to see how the values are affected. If additional tests, such as absolute values, need to be compared for minimums or maximums they can be added fairly easily. The PROC SQL code appears to be much more complicated (there are more statements and %SCAN is used many times), plus you need to understand SQL in addition to SAS to modify it. If you had to explain the code to someone else, probably the DO loops or arrays would be the easiest.

Which is fastest?
In our tests using PC SAS (Table 1), the macro DO loops algorithm was significantly faster than the PROC SQL method and only slightly faster than the arrays method. In general, all methods took only few seconds to process over a million cases and six variables, so judge according to your quantity of data and system resources.

In the tests using HP-UX Unix (Table 2), the overall results were similar to the PC-SAS testing. On a multiprocessor machine with more memory, the runs took even less time than on the PC. The macro DO loops method was the fastest of the three algorithms. Here, in comparison to the PC tests, the difference between macro DO loops and arrays was more pronounced. The PROC SQL method was again significantly slower than the other two methods.

Which is most efficient in terms of memory usage?
As shown in Table 1, there were no significant differences between the macro DO loops and arrays methods in terms of total memory usage observed in the PC SAS benchmarks. Both were data steps. The PROC SQL macro used slightly more memory than either macro DO loops or arrays. On the HP-UX system, the macro DO loops used 61K, arrays used 43K, and PROC SQL used 61K, none of which is very significant in terms of memory usage on today’s systems. Since FULLSTIMER gives significantly different views of memory usage based on the operating system, such output should not be compared across systems.

CONCLUSION
Three SAS macros that find tied minimum or maximum values have been presented. They provide advantages over the more common MIN() and MAX() functions by providing information regarding the source of the minimum or maximum values, including any tied values. They are all short in terms of necessary code and are generalized to process any number of variables (subject to SAS system limitations). Using macro code reduced the number of data steps and procs necessary to accomplish this task. Benchmarking tests were completed and showed that the operating system environment can make a difference in performance, and that the macro DO loops were generally the fastest, followed closely by the arrays.
algorithm, with the PROC SQL algorithm significantly slower than the other two algorithms. These methods are potentially useful in many industries including pharmaceutical and financial, anywhere there is a need to find tied values based on some function such MIN() or MAX() and the corresponding variables that account for these.

REFERENCES


ACKNOWLEDGMENTS
The authors would like to acknowledge Mary Cowmeadow, who inspired the first algorithm in this paper.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the authors at:

Kent Nassen
Work Phone: (734) 622-2048
Email: Kent.Nassen@pfizer.com

Subrahmanyam Pilli
Work Phone: (734) 622-3265
Email: Subra.Pilli@pfizer.com

Luai Alzoubi
Work Phone: (734) 622-2168
Email: Luai.Alzoubi@pfizer.com

Address:
Pfizer Global Research and Development
Ann Arbor Laboratories
2800 Plymouth Road
Ann Arbor, MI 48105

Web: www.pfizer.com