Paper 2025-2014

Using Sorting Algorithms to Create Sorted Lists

Matthew Neft, Highmark Inc; Chelle Pronko, Highmark Inc.

ABSTRACT

When providing lengthy cost and utilization data to medical providers it is ideal to sort the report by descending cost (or utilization) so that the high cost / utilized categories are at the top. This task can be easily solved using PROC SORT. However, when your data is listed out horizontally and you need other variables (such as unit cost per procedure or national average) to follow the sort but not be sorted themselves the solution is not as intuitive. This paper looks at two sorting algorithms to solve this problem. First, we look at the basic bubble sort (which is effective for small data) that sets up arrays for each variable and then sorts on just one of them. Next, we discuss the quicksort algorithm which is effective for large data sets. The results of the sorts provide sorted data that can be easily manipulated and put into reports using DDE or ODS.

INTRODUCTION

In the health insurance industry pay-for-value programs are an emerging trend where healthcare providers are rewarded for meeting certain quality and cost benchmarks. In order to assess a provider's progress, quarterly "Cost & Utilization" reports can be provided that show where dollars are being spent for a provider's patient population. Figure 1 shows an example of such a report.

	PMPM	Trend	Util/1000	Cost/Procedure	Nat'l Avg PMPM
Professional					
Primary Care	21.24	2.1%	4,601.0	150.23	20.12
Primary Care - Ped	19.54	1.2%	4,623.5	169.54	17.45
Chiropractic	10.11	(2.9%)	254.5	205.80	7.55
Radiologist	9.55	6.3%	403.0	137.13	10.23
OBGYN	5.62	(5.3%)	12.7	154.00	4.12
General Surgery	5.61	10.0%	8.5	1,022.35	7.55
Orthopedics	4.46	(5.2%)	237.5	335.09	4.25
Ambulance	4.22	(4.0%)	17.0	400.21	4.02
Physical Therapy	4.12	(0.1%)	63.6	57.67	3.37
Ophthalmology	3.05	0.0%	93.3	123.40	2.09
Emergency Medicine	2.77	4.7%	343.6	255.31	3.01
Dermatology	2.69	(18.3%)	127.3	96.92	2.54
Neuro Surgery	2.50	2.2%	0.0	3,021.45	1.47
Psychologist	1.99	4.0%	93.3	136.50	3.06
Cardiologist	1.81	(5.4%)	4.2	5,012.00	0.98

Figure 1: Example of a Cost & Utilization report that is sorted in descending order by PMPM.

The report is sorted such that the highest PMPM (per member per month cost) services are at the top. Since PMPMs will vary among providers (meaning that Primary Care, for example, won't always account for the most dollars), how can we create this list for each provider such that the PMPM column is sorted? If our SAS® dataset already looked like the report in Figure 1 then a PROC SORT could easily solve this problem. But, our dataset is not in that format. All of the data is listed horizontally for each practice as can be seen in Figure 2.

Obs	Practice	Category1	PMPM1	Trend1	Util1	CostProc1	NatlAvg1	Category2	PMPM2	Trend2
1	ABC Primary Care	Primary Care	21.24	2.1	4601	150.23	20.12	Radiologist	9.55	6.3
2	ZYX Family Practice	Chiropractic	9.54	1.5	2405	145.28	10.19	Dermatology	3.35	-4.6

Figure 2: Subset of a SAS® data set containing data for each practice listed horizontally. We need to sort the data horizontally before we can transpose and put into a report using DDE or ODS.

Initially we considered transposing the data and then doing a sort but we couldn't get the data into a workable format and the results weren't acceptable. The process turned out rather messy. Therefore we decided to institute a bubble sort on the arrays of data that we had so that when we transposed the data by practice it would already be sorted. The transposed data could then be split up by practice and put into reports like Figure 1using an output delivery method.

BUBBLE SORT

The bubble sort is one of the simplest sorting algorithms. Given an array of values, the bubble sort works its way through the array comparing each value to its adjacent value and swapping them if they are in the wrong order. The bubble sort will pass through the list until no swapping is needed and the list is in proper order. Figure 3 illustrates a simple bubble sort. We will sort the list in descending order so that we match the method used for the report in Figure 1. Starting with an unsorted list of numbers [5,1,6,9] the bubble sort will first compare 5 to 1, determine that 5 > 1 and move on to the next comparison. The second comparison is 1 to 6. Since 1 < 6 the bubble sort swaps their values and the resulting order is [5,6,1,9]. The algorithm continues like this until it reaches the end of the array, then it starts over at the beginning. It will continue like this until there is a clean sweep through the data without any swaps. In Figure 3 the bubble sort's third pass through the array results in the final result but a fourth pass would be required to confirm that no swaps are needed.

FIRST PASS THROUGH THE ARRAY	
5 1 6 9	The unsorted array
5 1 6 9	First comparison - no swap
5 1 6 9	Second comparison - results in a swap
5 6 1 9	Third comparison - results in a swap
5 6 9 1	Result of first pass

SECOND PASS THROUGH THE ARRAY	
5 6 9 1	Result of first pass
5 6 9 1	First comparison - results in a swap
6 5 9 1	Second comparison - results in a swap
6 9 5 1	Third comparison - no swap
6 9 5 1	Result of second pass

THIRD PASS THROUGH THE ARRAY	
6 9 5 1	Result of second pass
6 9 5 1	First comparison - results in a swap
9 6 5 1	Second comparison - no swap
9 6 5 1	Third comparison - no swap
9 6 5 1	Result of second pass

Figure 3. Starting with an unsorted array [5,1,6,9] the bubble sort will pass through the data until there are no swaps, resulting in the final array [9,6,5,1].

SAS CODE FOR A BUBBLE SORT OF ONE ARRAY

In order to code the bubble sort in SAS you will need to be familiar with array notation. Using the example illustrated in Figure 2 let's start out with one array of data. The numerical values will be contained in columns x1, x2, x3, and x4. This data set, *unsorted_list*, can be defined as follows:

```
1  data unsorted_list;
2  input x1 x2 x3 x4;
3  cards;
4  5 1 6 9
5  ;
6  run;
```

The next step is to define an array over x1 – x4 and perform the bubble sort – this can all be done in one data step.

```
 8  data sortarray (drop = n sorted temp i);
 9  array values(*) x1-x4;
10  set unsorted_list;
11  n = dim(values);
12  do until (sorted);
13  sorted=1;
14  do i = 1 to n - 1;
15      if values(i) < values(i+1) then do;
16          temp = values(i+1);
17          values(i+1) = values(i);
18          values(i) = temp;
19          sorted = 0;
20          end;
21      end;
22  end;
23  run;
```

Line 8: Define the new data set and drop any temp variables that get created.

Line 9: Define the array *values* over x1, x2, x3, and x4.

Line 10: Set the *unsorted_list* data set.

Line 11: Set n equal to the dimension of the array. This will be used on line 14 to set the limit of the DO loop.

Line 12 – 13: Set up a DO loop to run until the bubble sort iterations are complete. We set a binary variable *sorted* equal to 1. If the bubble sort has to swap any values it will set *sorted* to 0 (line 19) indicating that we need to go through the array again.

Line 14: Here we are defining the bubble sort DO loop to go from 1 to n – 1, where n is the dimension of the array. If an array has n elements there will be n – 1 adjacent element comparisons.

Line 15: This line is comparing two adjacent elements in the array. If the values satisfy the IF statement, that they are out of order, then lines 16 – 19 are run.

Line 16 – 19: Here is where the swap takes place. A temporary variable is set up to hold the larger value while the smaller value takes the larger value's place. The larger value is then inserted into the smaller value's location. The *sorted* variable is set to 0 indicating that another run through the array will be needed.

Line 20 – 23: Ending the IF statement and DO loops.

When printed out the final data set looks like this, a successful bubble sort!

Obs	x1	x2	x3	x4
1	9	6	5	1

SAS CODE FOR A BUBBLE SORT ON MULTIPLE ARRAYS

Now let's look at our initial problem, sorting the data in Figure 2. We can define an array for each of the sets of variables; Category1 – Category15, PMPM1 – PMPM15, etc. We can use the bubble sort to sort on the PMPM array and have all of the other arrays follow suit (but not be sorted themselves)! The SAS code below achieves this.

```
 8  data sortarray (drop = n sorted temp i);
 9  array category(*) category1 - category15;
10  array pmpm(*) pmpm1 - pmpm15;
11  array trend(*) trend1 - trend15;
12  array util(*) util1 - util15;
13  array costproc(*) costproc1 - costproc15;
14  array natlavg(*) natlavg1 - natlavg15;
15  set unsorted_list;
16  n = dim(category);
17  do until (sorted);
18  sorted=1;
19  do i = 1 to n - 1;
20      if pmpm(i) < pmpm(i+1) then do;
21          temp = pmpm(i+1);
22          temp1 = category(i+1);
23          temp2 = trend(i+1);
24          temp3 = util(i+1)
25          temp4 = costproc(i+1);
26          temp5 = natlavg(i+1);
27
28          pmpm(i+1) = pmpm(i);
29          category(i+1) = category(i);
30          trend(i+1) = trend(i);
31          util(i+1) = util(i);
32          costproc(i+1) = costproc(i);
33          natlavg(i+1) = natlavg(i);
34
35          pmpm(i) = temp;
36          category(i) = temp1;
37          trend(i) = temp2;
38          util(i) = temp3;
39          costproc(i) = temp4;
40          natlavg(i) = temp5;
41
42          sorted = 0;
43          end;
44      end;
45  end;
46  run;
```

Line 9 – 14: Assign arrays to all the variables that are part of the sort.

Line 20: Since we are sorting on PMPM alone, that is the only variable to get sorted. This line is the comparison.

Line 21 – 40: Here is where the swap is taking place just like before. But in this case we are swapping the values in all the arrays. But only the PMPM array is being sorted.

The results of this sort can be transposed by practice and output into a report similar to that found in Figure 1. For this specific report DDE was used.

BENEFITS AND DOWNSIDES

The simplicity of the bubble sort makes it very attractive to beginners and advanced programmers alike. The bubble sort works really well on small data sets where processing time will be negligible. For large data sets more efficient alternatives are available, such as the quicksort algorithm.

THE QUICKSORT ALGORITHM

The quicksort algorithm was developed by Tony Hoare, a British computer scientist, in 1960. It is also known as a 'divide and conquer algorithm' because the first step is to divide the list of data values into two sets of lists and then sorting those smaller lists by splitting them up into even smaller lists. The algorithm isn't as simple as the bubble sort but it is more efficient – especially with large data sets. To illustrate this we will use a larger array.

19	30	12	79	5	9	81	2	11	80

The first step in the quicksort algorithm is to pick a pivot. The pivot can be any value in the array but it helps if the pivot is a value somewhere in the middle range of values. For this example let's choose the first value of 19 as the pivot. Once we've chosen the pivot the next step is to start at the front of the array and find a value greater than the pivot, and then go to the back of the array to find a value smaller than the pivot. Starting at the front we find 30 which is greater than 19, and starting from the back we find 11 which is less than 19. Removing the pivot from the array we then swap indexes for the 30 and 11. The resulting array looks like:

	11	12	79	5	9	81	2	30	80

We then move on from the front, finding the next value larger than the pivot, 79, and from the back looking for the next value smaller than the pivot, 2. We swap their indexes and get a new array:

	11	12	2	5	9	81	79	30	80

After this step we are essentially done with the first step. All the values lower than 19 move to the left as the 19 is inserted in its correct position (all values to the left are less than it and all values to the right are greater than it).

11	12	2	5	9	19	81	79	30	80

We now have an unsorted set of low values, a 19 in the correct position, and an unsorted set of high values. We now perform the same function on the low values by choosing a pivot and swapping values; then we move to the high values and do the same thing. This is the divide and conquer portion of the algorithm. This is done until the array is completely sorted.

SAS CODE FOR A QUICKSORT

Since the quicksort algorithm is far more complicated than the bubble sort (and much longer) I did not include an example here. There is an excellent paper on the quicksort algorithm, including SAS code, written by Paul Dorfman. (See the references section for more information).

BENEFITS AND DOWNSIDES

The quicksort algorithm is a highly effective sorting method when dealing with large volumes of data that need arrays sorted. The ability for it to 'divide and conquer' allows it to run faster and not waste any processing time. The

downsides are that it is quite complicated to code for if you are working from scratch. You can easily find code on the internet or other SAS papers so this is not much of a hurdle anymore.

CONCLUSION

There are many sorting algorithms out there that can be used to sort data in arrays. You should choose a sorting algorithm that best suits your needs. If you are working a small set of data then the bubble sort is a more than ample method to use. The code is easy to write and the logic is fairly easy to follow. If you have large arrays that need sorted and are worried about processing time then a quicksort algorithm is the way to go. The coding is more difficult than the bubble sort but you can find a lot of resources on the internet to help you with.

REFERENCES

Dorfman, Paul. "QuickSorting and Array" Proceedings of SUGI 26. Available at http://www2.sas.com/proceedings/sugi26/p096-26.pdf.

Martin, David R. "Bubble Sort". 2007. Available at http://www.sorting-algorithms.com/bubble-sort.

SAS Support. "Sort variable values within an observation". Available at http://support.sas.com/kb/24/754.html.

Wikipedia. "Sorting Algorithm". Available at http://en.wikipedia.org/wiki/Sorting_algorithm.

CONTACT INFORMATION

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

Matt Neft
Highmark Inc
matthew.neft@highmark.com

Chelle Pronko
Highmark Inc
chelle.pronko@highmark.com