Enhanced Insertion Sorting

August 7, 2014

Background for this Enhanced Insertion Sort Routine

The procedure was first written in FORTRAN when I worked for Grumman Aerospace.

I was introduced to the insertion sort by my mentor Francis J. Nolan. He told me that he had to compose the algorithm for a job interview.

As I was coding the algorithm I realized that the procedure would have a much broader application if it were written to generate a permutation array which could then be applied as a sort operator.

I made use of this algorithm, which I named AORDER, during my entire carrier at Grumman.

Now that I am a Senior Excel VBA Developer, I started to refractor a number of my FORTRAN codes.

As I began work I realized that just converting FORTRAN to VBA would not be the complete answer.

I saw that the all the refactored routines needed to deal with Arrays for both input and output.

In the context of Excel, the ultimate input and output would be to spreadsheet Ranges, but there would be many circumstances when the output array from one process would be passed directly to another procedure without having to cycle back to a spreadsheet .

An Enhanced VBA Insertion Sort Procedure

The basic insertion sort algorithm is very direct.

Given a partial list of values which are sorted the algorithm seeks to add a new value to the sorted list.

This task takes two steps:

1. Search over the sorted list to determine where the new value belongs.
2. Split the sorted list to make room for the new value.

These two steps are repeated until the entire list of data values is sorted.

The essential features of this procedure are:

1. It generates a **stable sort** which keeps elements that have the same value in the same order as they were in the original list. **The Excel Spreadsheet Sort is a stable sort.**
2. It generates **a permutation array** which can be applied as a **sorting operator**. The original data is unchanged. This resulting sorting operator can be used to generate **compound sort operators**.

A number of operational tweaks have been made to enhance the performance of the sorting process:

1. The procedure is presented as two distinct subroutines, one for **Hi_To_Lo** sorting and another for **Lo_To_Hi** The benefit of this modification lets the algorithm to keep track of the latest inserted value and its location in the permutation array.

This information facilitates the search for the next insertion location. Instead of having to search over the entire list of sorted values, it can search from either the minimum to the latest or from the latest to the maximum.

2. The storage for the permutation array is initially set to double the size of the values array. By starting the permutation array in the middle, the permutation array can be split by moving the shortest part, moving the permutation down below the insertion point or moving the permutation up above the insertion point.

One set of routines are required for sorting strings and another for sorting numerical values.