Some Fast and Efficient PostScript Sorting Utilities

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552

http://www.tinaja.com
don@tinaja.com
(928) 428-4073

I have long had a pair of **BUBLSORT.PS** and a **INSORT.PS** utilities over on my **PostScript** library page. I recently needed to improve and update these for our new **Log File Analyzer** and **eBay Reporter** routines. The actual code can be found in our new **PRESORT1.PSL** utility. This **GuruGram** gives you an intro tutorial.

Classic Alphanumeric Bubble Sort

The **Bubble Sort** is the oldest and simplest of computer sorting algorithms. It is easy to understand and implement, uses few resources, and is largely data independent. Here is one bubble version…

> **Go through a list or array of N items.**
> **Compare the PRESENT item against the PREVIOUS one.**
> **Swap the two if PRESENT is SMALLER than PREVIOUS.**
>
> **Repeat for the first N-1 items on the list. Then for N-2.**
> **Continue until no unsorted items remain.**

On the first pass, the **largest** item ends up at the **end** of the list. Several other items also may have their position improved, working towards where they belong. On pass two, the second largest item is second from the end, and so on. After all passes, the list is sorted. The smallest items "bubble up" to the list start.

The only little problem with bubble is that it takes a **lot** of comparisons. These go up with the **square** of **n**, making bubble excruciatingly slow for very large **n**. Ferinstance, sorting **400** items may take almost **80,000** comparisons, while **1000** items will demand nearly **500,000**.

Although bubble is usually the laughing stock of all sort algorithms, it really is hard to beat for **low n** sorts. Where its simplicity, data independence, minimal resources, and low overhead completely blow everything else away.

Here is a new alphabetical bubble sort for **PostScript** string arrays…

```
/alphabubblesort2 { /curmat1 exch store curmat1 length
1 sub -1 1 {curmat1 0 get exch 1 exch 1 exch {/posn exch
store curmat1 posn get 2 copy lt {exch} if curmat1 exch
posn 1 sub exch put} for curmat1 exch posn exch put } for
curmat1 } bind store
```

When using **Distiller as a General Purpose PostScript Computer**, this utility sorts 400 items in a quarter of a second on an 800 MHz Pentium host machine. And is thus more than fast enough for routine use on shorter lists.

More detailed comments on this routine appear in **PRESORT1.PSL**. The initially assumed data format will be an array of strings to be sorted alphabetically. Such as **[(string0) (string1) (string2) ... (stringN)]**.

In general, there is a loop within a loop. The inside loop compares, tests, and sometimes swaps sequential items using **exch**. The outside downcounting loop decides how many progressively fewer sequential items to test.

The utility keeps the **previous** list item on the stack top, gets the next item, compares them, and swaps them if needed. The **smaller** item then gets resaved to the original array, and the **larger** item becomes previous for the next pass. No additional resources are used except for a few stack positions.

Note that the **PostScript search** command does a full **lexigraphic** "dictionary" search when applied to two strings. But does **not** combine upper and lower case.

Normally, you will insert this utility into your larger **PostScript** program, save it as a **standard ASCII textfile**, and send it to **Acrobat Distiller** following the full details shown in **DISTLANG.PSL** or **GuruGram #28**.

Popularity Bubble Sort

A simple modification lets you sort on item **popularity** . This is useful for **Logfile Analysis** where you want a list of files downloaded by user interest.

Assume your data is an **array of arrays** of form **[[(string0) popcount0][(string1) popcount1][(string2) popcount2] ... [(stringN) popcountN]]**. This time, instead of directly comparing present and previous, you instead add a **0 get** if you want to sort alphanumerically. Or a **1 get** if you want to sort by popularity…

```
/popbubblesort2 { /curmat1 exch store curmat1 length
1 sub -1 1 {curmat1 0 get exch 1 exch 1 exch {/posn exch
store curmat1 posn get 2 copy 1 get exch 1 get lt {exch}
if curmat1 exch posn 1 sub exch put} for curmat1 exch
posn exch put } for curmat1 } bind store
```

In this example, the **most popular** string appears at the **start** of the list. You can easily reverse the list by changing the **lt** to **gt**.

Presorting for Speed

There's a surprisingly easy extension to bubble that can ridiculously speed it up for longer sorts. Letting it approach the **n*log(n)** speed of **fancier sorts** rather than the **n^2** of bubble…

> **Pre-sorting your data into K bins can speed up a bubble sort by over K times!**

Which really lets bubble hunt with the big dawgs. Say you had a 260 entry string array consisting only of uppercase letters. Sort them into 26 bins A-Z. Temporarily assume further the best case of each letter being equally likely. You'll end up with ten strings in each bin. Now sort them.

With bubble, **n** sorts will need **n(n-1)/2** comparisons. Without the bins, you will need 260*259/2 = **33,670** bubble comparisons. Any given **n=10** bin now needs only 10*9/2 = **45** bubble comparisons. But since there are 26 bins, you will need 26*45 = **1170** comparisons total. Which is less than **1/26th** of what you would need without the bins!

Naturally, the extra overhead and any nonuniform bin distribution cuts into your speed gains. But bin presorting almost always will give you a dramatic speedup. There's also no need to sort any empty bins or those with only a single entry.

There's little overhead for using a lot of bins. And **Acrobat Distiller** usually has tons of resources to spare. A **PostScript** string might be presorted into 128 bins for low ASCII only. Or even into 256 bins if the string involves high ASCII or is used for general data storage.

Here's a sample alphanumeric string sorting routine using bins…

```
/alphabubwithbins {/curdat2 exch store /matmat mark 256
{[]} repeat ] store curdat {dup 0 get  /curint exch
store mark exch matmat curint get aload pop ] matmat
exch curint exch put} forall mark matmat { dup length 1
ge {dup length 2 ge {alphabubblesort2 { } forall} if}
{pop} ifelse} forall ]} bind store
```

You first save your input array, followed by defining an **empty** array of 256 subarrays. Next, you fill your bins. Do this by viewing one string at a time and extracting the 0-255 ASCII character to find out which bin to add the string to. When finished, all your "**A**" strings will be in the "**A**" bin and so on.

Additional string and array conversion info appears in **STRCONV.PDF** in our **GuruGram** library.

Finally, you build an output array that ignores empty bins, uses single string bins as is, and bubble sorts bins having more than one string. The overhead on all this seems low enough that you are likely to gain even for **n=25** strings. Anything higher gets much faster. And anything less does not matter.

A Presorted Popularity Variation

Using bins gets even faster and more interesting when sorting on popularity. Many times, you will have an exponential or a **Raleigh** data distribution in which you will have lots and lots of single hits, fewer doubles, some triples, and progressively fewer of the higher numbers.

This time, you once again use 256 bins. Only you make bin 256 special in that it will hold **all** popularities of 256 or higher. Chances are good it will remain empty with typical **web log data** popularity counts. You can easily bubble sort it if you have to.

On the lower bins, you only have to bubble sort if you want, say, all the "three" popularity items to be **further** alpha sorted. Very often, no bubble sorting at all will be needed! In this case, what you really have done a very fast variation on an **insertion sort**.

I'll save details on this one as an exercise for the student.

For More Help

Additional **PostScript** and **Acrobat** and assistance is available per the previously shown web links. Custom programming and design services are now available at our standard consulting rates. Per our **InfoPack Services**. Or you can directly **email** me.

Additional **GuruGrams** columns await your ongoing support as a **Synergetics Partner**.