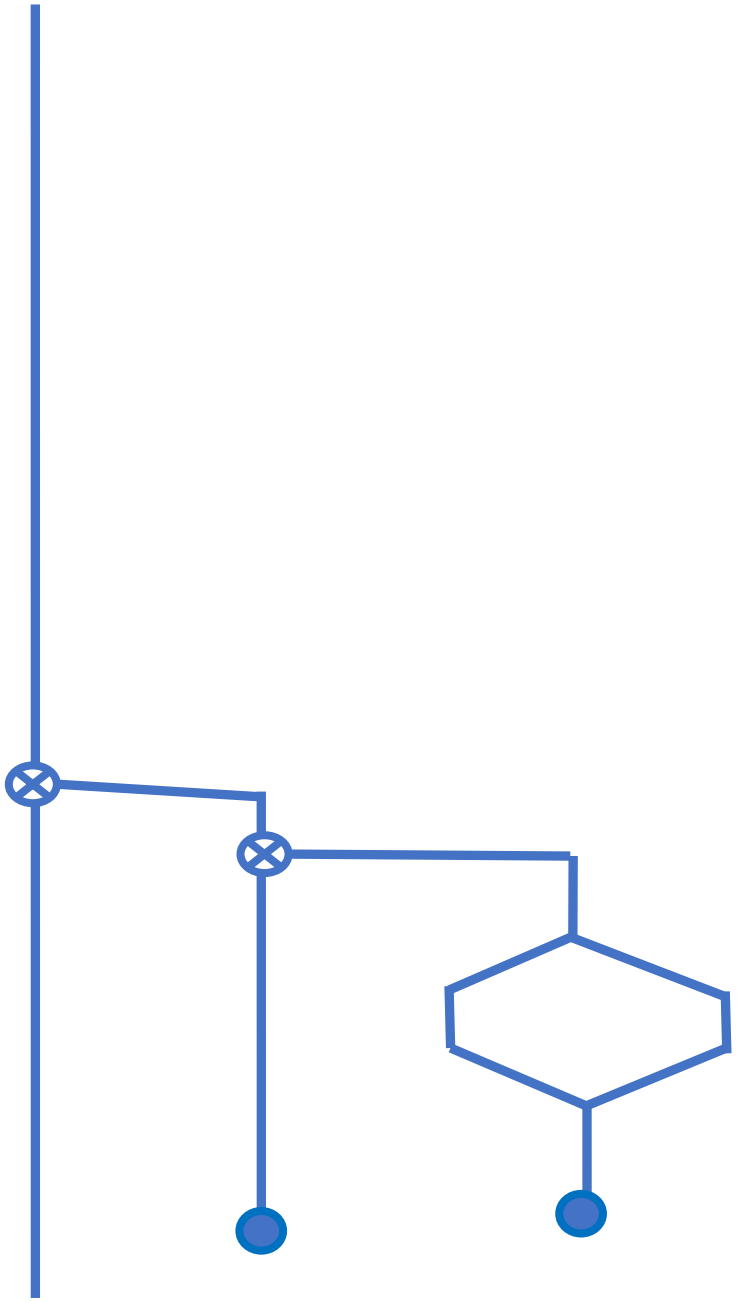

Simple Sorts

SVFIG
Feb. 22, 2025
Bill Ragsdale

The Need

A simple sort for tens to low hundreds of rows.

More complex sorts are suitable for larger matrices.

Four bubble sort strategies.

1. Full sort all rows. n-squared.
2. Full sort all rows with early exit.
3. Diminishing sort. n-squared /2.
4. Diminishing sort with early exit.

Comments.

Is often used as a cleanup after more complex sorts.

Sort time increases by n-squared.

Add a digit to number of rows, sort time increases by 100-fold.

Can sort in place with (almost) no temporary memory.

Comments.

Scan downward by row examining a column value.

If column entry is larger than the next column entry, exchange rows.

Larger values sink to the bottom.

Preserves the order of prior sorted columns.

Each scan can cover one less row as the last row is known to be the largest in that column.

Input, 10 Rows

```
7.0000 7.0000 1.0000
6.0000 9.0000 7.0000
1.0000 2.0000 2.0000
6.0000 8.0000 4.0000
8.0000 .00000 6.0000
3.0000 .00000 5.0000
7.0000 6.0000 4.0000
2.0000 5.0000 2.0000
7.0000 8.0000 7.0000
9.0000 8.0000 9.0000
```

Sort On Column Three

```
7.0000 7.0000 1.0000
1.0000 2.0000 2.0000
2.0000 5.0000 2.0000
6.0000 8.0000 4.0000
7.0000 6.0000 4.0000
3.0000 .00000 5.0000
8.0000 .00000 6.0000
6.0000 9.0000 7.0000
7.0000 8.0000 7.0000
9.0000 8.0000 9.0000
```

Sort On Column Three Then Two

```
3.0000 .00000 5.0000
8.0000 .00000 6.0000
1.0000 2.0000 2.0000
2.0000 5.0000 2.0000
7.0000 6.0000 4.0000
7.0000 7.0000 1.0000
6.0000 8.0000 4.0000
7.0000 8.0000 7.0000
9.0000 8.0000 9.0000
6.0000 9.0000 7.0000
```

Row Exchange Key Word

```
: }row-exchange ( x{ c  r1 r2 --- x{ c   )
\ Given the matrix, column and two row numbers
3pick   2pick 0 }}  ( FROM )  transient{ 0  0 }}  ( TO )
    transient{ }size  cmove
3pick   over 0  }}  ( FROM )  4pick  3pick 0  }}  ( TO )
    transient{ }size  cmove
transient{ 0 0  }}  ( FROM )  4pick  2pick 0  }}  ( TO )
    transient{ }size  cmove
2drop   ;
```

One Pass Through Matrix

```
: one-pass ( x{ c row-through --- x{ c  )
  1+  0
   do  over  i    2pick }}@   \ compare to i-th+1 cell
       over  i 1+ 2pick }}@  F>
       if  i  i 1+ }row-exchange then
     loop ;
```

Bubble Sort

```
: }bubble ( x{ col --- ) \ sort rows of x{ on column
1 2pick }cols  opentransient{
over }rows  2 -  0 swap
   do   i          \ x{ c  i of target row
        one-pass   \ x{ c
    -1 +loop
 2drop   closetransient{   ;
```

Results

Rows	Square			
10	< 1 msec			
100	40 msec			
1,000	4.4 sec			
10,000				

Results

Rows	Square	Square + Early Exit		
10	< 1 msec	< 1 msec		
100	40 msec	34 msec		
1,000	4.4 sec	3.7 sec		
10,000				

Results

Rows	Square	Square + Early Exit	Decreasing	
10	< 1 msec	< 1 msec	< 1 msec	
100	40 msec	34 msec	32 msec	
1,000	4.4 sec	3.7 sec	3.8 sec	
10,000			5 min 29 sec 6 min 41 sec	

Results

Rows	Square	Square + Early Exit	Decreasing	Decreasing + Early Exit
10	< 1 msec	< 1 msec	< 1 msec	< 1 msec
100	40 msec	34 msec	32 msec	16-32 msec.
1,000	4.4 sec	3.7 sec	3.8 sec	2.3 – 3.7 sec
10,000			5 min 29 sec 6 min 41 sec	

Onward

- I should move to the next more complex sort:
- Shell sort?
- Insertion sort?
- The sorts in Excel are fantastic. Hundreds and thousands in fractions of a second.
- I suspect they use large amounts of short-term memory.