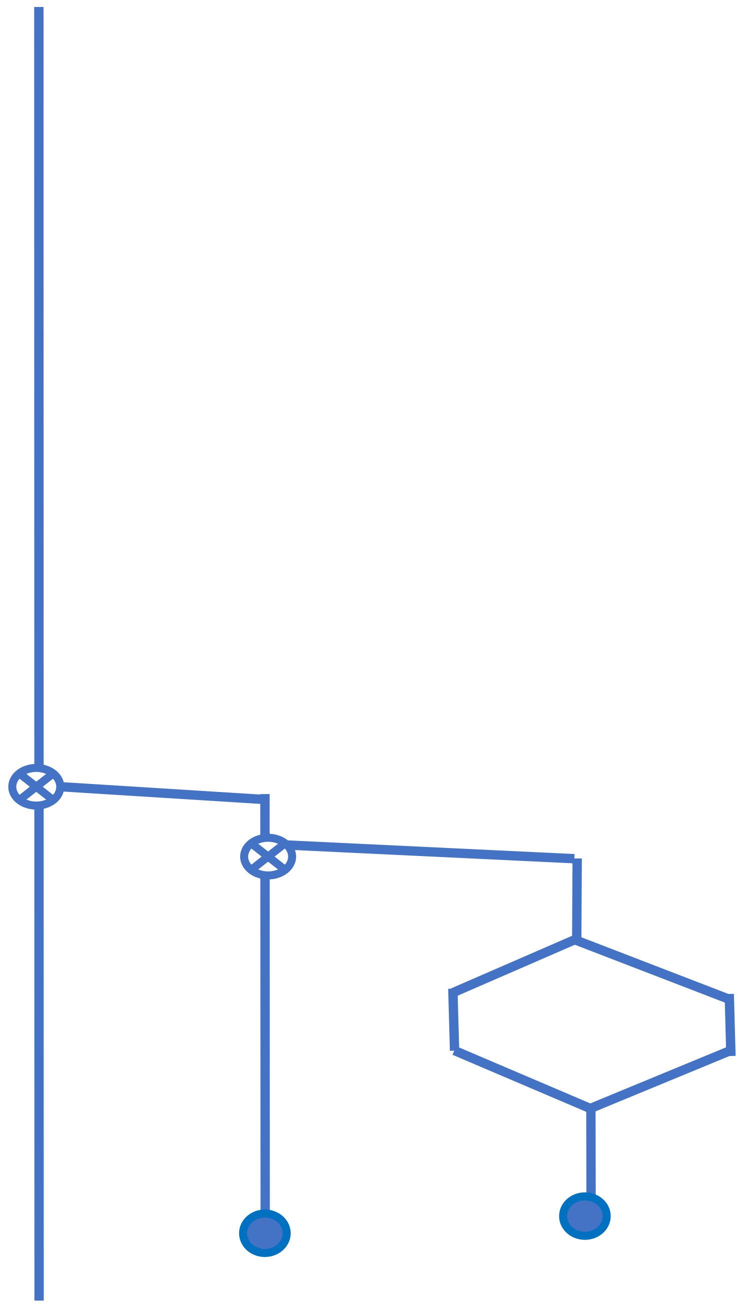

A Merge Sort In Forth

SVFIG
Mar. 22, 2025
Bill Ragsdale

The Need

Sorts for a Forth matrix system.

- Bubble Sort, last month. n-squared.
- Merge Sort, this month. $n\log_2(n)$.

Results, effort worthwhile?

Rows	Bubble	Merge
10	< 1 msec	< 1 msec
100	16 - 32 msec	2 - 4 msec
1,000	2.3 - 3.7 sec	15 - 16 msec
10,000	5 min 29 sec 6 min 41 sec	65 - 110 msec

Merge-Sort Summary

Scan locating a pair of runs.

Merge-sort that pair into one run.

If at an odd run at the matrix end, carry it across.

Repeat until two runs remain.

Final merge-sort pass.

End.

How: }Scanner

Set last matrix row as the default exit value.

From the starting row.

Compare its sort value to that of the following row.

Finding the row with the lessor or equal value ,

  return row number just above and exit,

  otherwise, increment row and loop.

If no ending value qualifies, return the default value.

}Scanner

```
: }Scanner ( x{ r1 C --- r2 flag ) \ locate run end
  N' 1- to result                \ default ending value
  3dup }}@                       \ sort value of the origin row
  N' 1-  rot                     \ start at r row
?do                              \ scan to matrix end
   2dup  i 1+ swap  }}@          \ value from target row
   Fdup Frot F<                  \ lesser marks new run.
   if  i to result leave then  \ end of run
 loop
   2drop  Fdrop  result  dup N' 1- =  ;
```

Run }Locator

Use }scanner to find a run.

Set 1st run A′ start and 1st Ax′ end.

Return a code if Ax′ ends the run.

If second run is found,

    Set 2nd run start B′ and 2nd run Bx′ end.

Return a code if Bx′ ends the matrix.

}Locator

```
: }locator ( x{ r C --- code )
  over to  A'  -1 to Ax' -1 to B' -1 to Bx'
  A'  0= if 0 to w-row then
                                 3dup }Scanner
  swap to Ax'  if  3drop  0 exit then  \ at matrix end.
  Ax' 1+ to B' nip B' swap }scanner     \ row flag
  swap to Bx'                           \ mark 2nd end
  if  -1  else 1  then    ;
```

}2-merge

Use }locator to determine the locations of two runs.

Find the row with the smallest 'C' value.

Copy identified row to the working{ matrix.

Test if that run has been exhausted.

If so, transfer the remaining portion of the <u>other</u> run
    to the working{ matrix.

Set the next start row into A'.

}2-merge

```
: }2-merge ( x{ C --- )    \ merge two runs
begin  2dup A' swap  }}@      \ get B' row sort cell float
       2dup B' swap  }}@  F<= \ get A' row sort cell
if over A’ 0 }}
   working{ w-row 0 }}   \ r-addr r-addr
   x{bytes cmove          \ move row
   1 +to A'  1 +To w-row  A' Ax'  >  \ compare A' Ax' >
   if drop }copy-second-run exit then
      else  over B' 0 }}              \ setup transfer
      working{ w-row 0 }}  x{bytes cmove
      1 +to B'  1 +to w-row B' Bx'  >  \ at 2nd run end
      if drop }copy-first-run exit then then
again  ;
```

Elements, run actions

A: One run found, matrix already sorted. Exit.

C: Only two runs found, do one merge. Exit.

D: Two ending runs found, merge and restart sort.

E: Two runs found, merge and continue.

B: One ending run found, carry it across and restart sort.

}Msort

Initialize control parameters A’ Ax’ B’ Bx’

Run }locator setting A’ Ax’ B’ Bx’.

Enter a case statement on the }locator return value.

Select from E (most likely), AB, CD

From the returned value either loop or end.

}Msort

```
: }Msort  ( x{ C --- )      \  merge sort
    2dup 0 swap }locator      \ locate and receive code
 begin
     case                  \ select action for }selector code
      1 of 2dup }case-E     endof    \ interior merge
      0 of 2dup }case-AB    endof    \ sorted or ending run
     -1 of 2dup }case-CD    endof    \ two runs or end
     cr cr abort" error in }Msort"
     endcase
   until                   \ upon flag from a merge word
  2drop   ;
```

Elements, run actions

A: One run found, matrix already sorted. Exit.

C: Only two runs found, do one merge. Exit.

D: Two ending runs found, merge and restart sort.

E: Two runs found, merge and continue.

B: One ending run found, carry it across and restart sort.

Input, 12 Rows

```
2.0000 15.000
5.0000 13.000
7.0000 7.0000
19.000 6.0000
17.000 7.0000
19.000 17.000
1.0000 8.0000
10.000 15.000
19.000 19.000
5.0000 9.0000
16.000 15.000
5.0000 15.000
```

Sort On Column Two

```
19.000 6.0000
7.0000 7.0000
17.000 7.0000
1.0000 8.0000
5.0000 9.0000
5.0000 13.000
2.0000 15.000
10.000 15.000
16.000 15.000
5.0000 15.000
19.000 17.000
19.000 19.000
```

Sort On Column Two Then One

```
1.0000 8.0000
2.0000 15.000
5.0000 9.0000
5.0000 13.000
5.0000 15.000
7.0000 7.0000
10.000 15.000
16.000 15.000
17.000 7.0000
19.000 6.0000
19.000 17.000
19.000 19.000
```

Time Results

Rows	Bubble	Merge
10	< 1 msec	< 1 msec
100	16-32 msec	2 - 4 msec
1,000	2.3-3.7 sec	15 - 16 msec
10,000	5 min 29 sec 6 min 41 sec	65 - 110 msec

Machine Loading Results, passes

Rows	Bubble	Merge
10	50	4
100	5,000	7
1,000	500,000	10
10,000	50,000,000	17

Onward

Discovery: If starting with an odd number of runs, a short run will remain at the end until the last merge.

This will demand a final merge for that very short run.

So, upon finding an odd number of runs, do an in-place bubble sort of the first run pair.

Will decrease sort time by $\log_2(n-1)/\log_2(n)$.

This tends to keep all runs of a similar size.

Also

This merge sort requires an identical added working space for the sort values and associated row values.

I allocate a temporary space to fully duplicate the source matrix.

Space is rows x columns x 8 bytes.

A matrix of 1,000 rows by 5 columns 40 Kbytes.

Some other sorts use a smaller working space.

Conclusion

The bubble sort is so compact, I’ll leave it the core matrix code,

And make the merge-sort a load module.

Sorting and optimization can be an infinite time sink.

I think I’m done.