Contemplate Sorting with Columnsort

Lester I. McCann

mccann@uwp.edu

Computer Science Department

University of Wisconsin — Parkside

Kenosha, WI

ACM SIGCSE

March 5, 2004

Outline

- Motivation
- The Columnsort Algorithm
  - History
  - Prerequisites
  - The Eight Steps of Columnsort
  - Does Columnsort Actually Work?
  - Is Columnsort Actually Efficient?
- Columnsort Programming Exercises
  - Year One
  - Year Two
- Other Ideas for Assignments
- Conclusion

Motivation

- Sorting is an essential programming technique
  - Discussed in many core CS courses
- Traditional sorting algorithms are ubiquitous
  - Great examples of programming techniques
  - Good fodder for algorithm analysis
  - Different algorithms solve the same problem
- A “new” algorithm can provide a fresh perspective
  - But beware of losing sight of objectives!

A Brief History of Columnsort

- Introduced by Leighton in a 1984 extended abstract
- A generalization of Odd–Even Mergesort
- Designed as a parallel algorithm for a rectangular processor mesh
- Recently the motivation for the work of Chaudry at Dartmouth College
- Used in our work as an internal sorting algorithm

Prerequisites of Columnsort

- Quantity of items to be sorted $= n = r \cdot s$
- Items exist in an $r \times s$ matrix
- Restrictions:
  - $r \,\%\, s = 0$
  - $r \geq 2(s-1)^2$
- Resulting matrix will be sorted in Column–Major Order

Eight Steps of Columnsort

- Step 1: Sort the Columns

$$\xrightarrow{\textbf{Start}} \begin{bmatrix} 23 & 15 & 9 \\ 26 & 11 & 3 \\ 2 & 18 & 7 \\ 22 & 5 & 14 \\ 17 & 20 & 25 \\ 24 & 10 & 1 \\ 8 & 13 & 21 \\ 27 & 12 & 4 \\ 19 & 16 & 6 \end{bmatrix} \xrightarrow{\textbf{Step 1}} \begin{bmatrix} 2 & 5 & 1 \\ 8 & 10 & 3 \\ 17 & 11 & 4 \\ 19 & 12 & 6 \\ 22 & 13 & 7 \\ 23 & 15 & 9 \\ 24 & 16 & 14 \\ 26 & 18 & 21 \\ 27 & 20 & 25 \end{bmatrix}$$

Eight Steps of Columnsort (cont.)

- Step 2: Transpose (Turn Columns Into Rows)

$$\xrightarrow{\textbf{Step 1}} \begin{bmatrix} 2 & 5 & 1 \\ 8 & 10 & 3 \\ 17 & 11 & 4 \\ 19 & 12 & 6 \\ 22 & 13 & 7 \\ 23 & 15 & 9 \\ 24 & 16 & 14 \\ 26 & 18 & 21 \\ 27 & 20 & 25 \end{bmatrix} \xrightarrow{\textbf{Step 2}} \begin{bmatrix} 2 & 8 & 17 \\ 19 & 22 & 23 \\ 24 & 26 & 27 \\ 5 & 10 & 11 \\ 12 & 13 & 15 \\ 16 & 18 & 20 \\ 1 & 3 & 4 \\ 6 & 7 & 9 \\ 14 & 21 & 25 \end{bmatrix}$$

Eight Steps of Columnsort (cont.)

- Step 3: Sort the Columns Again

$$\xrightarrow{\textbf{Step 2}} \begin{bmatrix} 2 & 8 & 17 \\ 19 & 22 & 23 \\ 24 & 26 & 27 \\ 5 & 10 & 11 \\ 12 & 13 & 15 \\ 16 & 18 & 20 \\ 1 & 3 & 4 \\ 6 & 7 & 9 \\ 14 & 21 & 25 \end{bmatrix} \xrightarrow{\textbf{Step 3}} \begin{bmatrix} 1 & 3 & 4 \\ 2 & 7 & 9 \\ 5 & 8 & 11 \\ 6 & 10 & 15 \\ 12 & 13 & 17 \\ 14 & 18 & 20 \\ 16 & 21 & 23 \\ 19 & 22 & 25 \\ 24 & 26 & 27 \end{bmatrix}$$

Eight Steps of Columnsort (cont.)

- Step 4: Reverse Step 2's Transposition

$$\xrightarrow{\text{Step 3}} \begin{bmatrix} 1 & 3 & 4 \\ 2 & 7 & 9 \\ 5 & 8 & 11 \\ 6 & 10 & 15 \\ 12 & 13 & 17 \\ 14 & 18 & 20 \\ 16 & 21 & 23 \\ 19 & 22 & 25 \\ 24 & 26 & 27 \end{bmatrix} \xrightarrow{\text{Step 4}} \begin{bmatrix} 1 & 6 & 16 \\ 3 & 10 & 21 \\ 4 & 15 & 23 \\ 2 & 12 & 19 \\ 7 & 13 & 22 \\ 9 & 17 & 25 \\ 5 & 14 & 24 \\ 8 & 18 & 26 \\ 11 & 20 & 27 \end{bmatrix}$$

Eight Steps of Columnsort (cont.)

- Step 5: Sort the Columns Yet Again

$$\xrightarrow{\textbf{Step 4}} \begin{bmatrix} 1 & 6 & 16 \\ 3 & 10 & 21 \\ 4 & 15 & 23 \\ 2 & 12 & 19 \\ 7 & 13 & 22 \\ 9 & 17 & 25 \\ 5 & 14 & 24 \\ 8 & 18 & 26 \\ 11 & 20 & 27 \end{bmatrix} \xrightarrow{\textbf{Step 5}} \begin{bmatrix} 1 & 6 & 16 \\ 2 & 10 & 19 \\ 3 & 12 & 21 \\ 4 & 13 & 22 \\ 5 & 14 & 23 \\ 7 & 15 & 24 \\ 8 & 17 & 25 \\ 9 & 18 & 26 \\ 11 & 20 & 27 \end{bmatrix}$$

Eight Steps of Columnsort (cont.)

- Step 6: Shift 'Forward' by $\lfloor \frac{r}{2} \rfloor$ Positions

$$\xrightarrow{\textbf{Step 5}} \begin{bmatrix} 1 & 6 & 16 \\ 2 & 10 & 19 \\ 3 & 12 & 21 \\ 4 & 13 & 22 \\ 5 & 14 & 23 \\ 7 & 15 & 24 \\ 8 & 17 & 25 \\ 9 & 18 & 26 \\ 11 & 20 & 27 \end{bmatrix} \xrightarrow{\textbf{Step 6}} \begin{bmatrix} -\infty & 7 & 15 & 24 \\ -\infty & 8 & 17 & 25 \\ -\infty & 9 & 18 & 26 \\ -\infty & 11 & 20 & 27 \\ 1 & 6 & 16 & \infty \\ 2 & 10 & 19 & \infty \\ 3 & 12 & 21 & \infty \\ 4 & 13 & 22 & \infty \\ 5 & 14 & 23 & \infty \end{bmatrix}$$

Eight Steps of Columnsort (cont.)

- Step 7: Sort the Columns One Last Time

$$\xrightarrow{\textbf{Step 6}} \begin{bmatrix} -\infty & 7 & 15 & 24 \\ -\infty & 8 & 17 & 25 \\ -\infty & 9 & 18 & 26 \\ -\infty & 11 & 20 & 27 \\ 1 & 6 & 16 & \infty \\ 2 & 10 & 19 & \infty \\ 3 & 12 & 21 & \infty \\ 4 & 13 & 22 & \infty \\ 5 & 14 & 23 & \infty \end{bmatrix} \xrightarrow{\textbf{Step 7}} \begin{bmatrix} -\infty & 6 & 15 & 24 \\ -\infty & 7 & 16 & 25 \\ -\infty & 8 & 17 & 26 \\ -\infty & 9 & 18 & 27 \\ 1 & 10 & 19 & \infty \\ 2 & 11 & 20 & \infty \\ 3 & 12 & 21 & \infty \\ 4 & 13 & 22 & \infty \\ 5 & 14 & 23 & \infty \end{bmatrix}$$

Eight Steps of Columnsort (cont.)

- Step 8: Shift 'Back' by $\lfloor \frac{r}{2} \rfloor$ Positions

$$\xrightarrow{\textbf{Step 7}} \begin{bmatrix} -\infty & 6 & 15 & 24 \\ -\infty & 7 & 16 & 25 \\ -\infty & 8 & 17 & 26 \\ -\infty & 9 & 18 & 27 \\ 1 & 10 & 19 & \infty \\ 2 & 11 & 20 & \infty \\ 3 & 12 & 21 & \infty \\ 4 & 13 & 22 & \infty \\ 5 & 14 & 23 & \infty \end{bmatrix} \xrightarrow{\textbf{Step 8}} \begin{bmatrix} 1 & 10 & 19 \\ 2 & 11 & 20 \\ 3 & 12 & 21 \\ 4 & 13 & 22 \\ 5 & 14 & 23 \\ 6 & 15 & 24 \\ 7 & 16 & 25 \\ 8 & 17 & 26 \\ 9 & 18 & 27 \end{bmatrix}$$

Does Columnsort Actually Work?

- Steps 1 – 4 place items within $(s-1)^2$ positions of their final locations
  - See Leighton for the proof

$$\xrightarrow{\text{Step 4}} \begin{bmatrix} 1 & 6 & 16 \\ 3 & 10 & 21 \\ 4 & 15 & 23 \\ 2 & 12 & 19 \\ 7 & 13 & 22 \\ 9 & 17 & 25 \\ 5 & 14 & 24 \\ 8 & 18 & 26 \\ 11 & 20 & 27 \end{bmatrix} \begin{bmatrix} 0 & -4 & -3 \\ +1 & -1 & +1 \\ +1 & +3 & +2 \\ -2 & -1 & -3 \\ +2 & -1 & -1 \\ +3 & +2 & +1 \\ -2 & -2 & -1 \\ 0 & +1 & 0 \\ +2 & +2 & 0 \end{bmatrix}$$

Does Columnsort Actually Work? (cont.)

- Steps 5 – 8 place items into their destination columns in sorted order.
  - $r \geq 2(s-1)^2$, and $\lfloor \frac{r}{2} \rfloor \geq (s-1)^2$

$$\xrightarrow{\textbf{Step 5}} \begin{bmatrix} 1 & 6 & 16 \\ 2 & 10 & 19 \\ 3 & 12 & 21 \\ 4 & 13 & 22 \\ 5 & 14 & 23 \\ 7 & 15 & 24 \\ 8 & 17 & 25 \\ 9 & 18 & 26 \\ 11 & 20 & 27 \end{bmatrix} \xrightarrow{\textbf{Step 6}} \begin{bmatrix} -\infty & 7 & 15 & 24 \\ -\infty & 8 & 17 & 25 \\ -\infty & 9 & 18 & 26 \\ -\infty & 11 & 20 & 27 \\ 1 & 6 & 16 & \infty \\ 2 & 10 & 19 & \infty \\ 3 & 12 & 21 & \infty \\ 4 & 13 & 22 & \infty \\ 5 & 14 & 23 & \infty \end{bmatrix}$$

Is Columnsort Actually Efficient?

- Doesn't *feel* efficient!
  - Troubles some students; only want to learn the "best"
  - *Why should we spend time implementing an algorithm that no one has ever heard of?*
- Efficiency depends on the implementation decisions
  - There are plenty to be made
  - Makes Columnsort the source of many examples and activities

Columnsort Programming Exercises

- Used in consecutive offerings of a Data Structures and Algorithm Analysis course
- Common student background:
  - Completed CS 1, CS 2, Prob & Stats
  - Lectures on standard $O(n^2)$ and $O(n\, log_2 n)$ sorts
  - Columnsort presentation and complete example

Year One

- Assignment: Sort supplied data in least time
  - Test data: One million integers (more or less)
  - Timing measurements:
    - Java: `System.currentTimeMillis()`
    - C++: `clock()`
  - 20% / 10% / 5% bonuses for fastest three in each language
    - Worth less than 1% of total course points

Year One Results

- 29 of 31 completed assignment on–time
  - Highest completion percentage of the semester
- Performance of implementations was quite varied:
  - Java: 3 sec $\leftrightarrow$ 2+ hours (median: 7 sec)
  - C++: $<$ 1 sec $\leftrightarrow$ 14+ minutes (median: 6 sec)
- Notable quote:
  - *"I didn't know a[n] n–squared sort took so long!"*

Year Two

- Assignment: How much data can be sorted in a second?
  - Inspired by SIGMOD's SecondSort contest
  - Records have 10–byte sort keys
- All students used Java and `System.currentTimeMillis()`
- Meager incentive: A prize for the 'winner'

Trinkets Make Students Happy!

SecondSort with Columnsort 'winner' Peter Joachim (on left)

Year Two Results

- 15 of 15 completed assignment on–time
  - Matched by only one other assignment
- Again, performance varied considerably:
  - 34K — 180K records in one second (median: 57K)
- Notable quote:
  - *“Is Columnsort a failed attempt at a sorting algorithm?”*

Year Two Results (cont.)

- Students completed pre–/post–assignment surveys
  - Amount of interest in assignment unchanged
    - 4.4 on a 0–6 scale
  - Which common sort is most like Columnsort?
    - Mergesort: Before: 31% After: 58%
    - Shellsort: Before: 25% After: 17%
  - $> 50\%$ reported that $s = 1$ is the best choice
    - Instructor's implementation worked best with $s = 2$ or $s = 3$, and was faster than Java's sorts

Year Two Attitudes

- About one–third of the students...
  - ...would have preferred to code Quicksort instead
  - ...conducted a Columnsort literature search
- About half of the students...
  - ...cited the mystery prize as a motivating factor
  - ...compared their implementation to another sort
- Over half of the students...
  - ...believed their implementations to be $O(n^2)$

    (although no one verified that feeling)

Other Ideas for Assignments

- Choice of representation:
  - 1D or 2D array?
- How to “shortcut” even–numbered steps?
- Choice of algorithm for sorting columns:
  - Use a library sort or implement your own?
  - Should the Step 1 choice match that of Step 7?
- Under which circumstances can Columnsort be stopped after Step N?

Conclusion

- Columnsort...
  - ...can be the source of a variety of exercises and analyses
  - ...motivates students to explore alternatives
  - ...shows that ‘algorithm archeology’ is worthwhile

Any Questions?

mccann@cs.uwp.edu

These full–screen PDF slides were created in LaTeX using the `prosper` class.