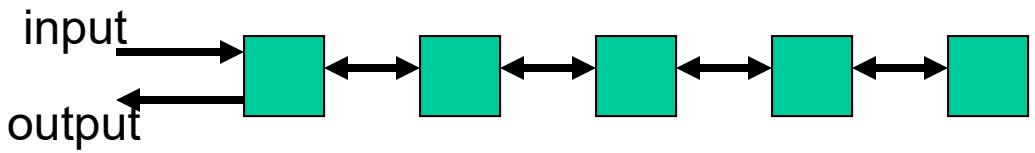
Lecture: Systolic Arrays I

- Topics: sorting and matrix algorithms

- Bloom Filters:
  False positive rate of 2.1% for a 2K bit filter holding 256 elements and using 5 hashes

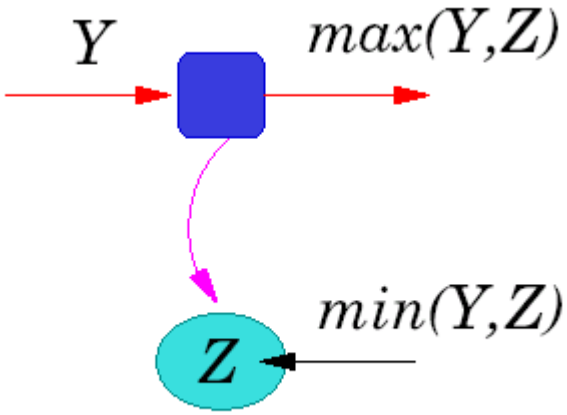
Systolic Model

- Each processor has bidirectional links to its neighbors
- All processors share a single clock (asynchronous designs will require minor modifications)
- At each clock, processors receive inputs from neighbors, perform computations, generate output for neighbors, and update local storage

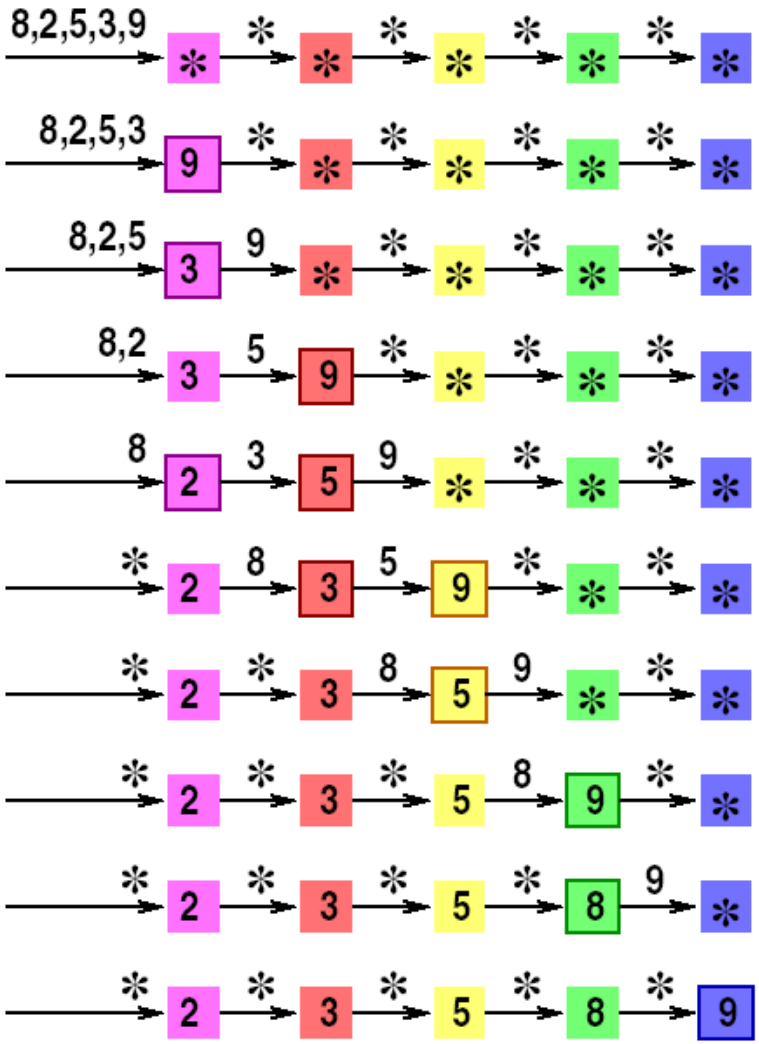


Sorting on a Linear Array

- Each processor stores the minimum number it has seen
- Initial value in storage and on network is “∗”, which is bigger than any input and also means “no signal”
- On receiving number Y from left neighbor, the processor keeps the smaller of Y and current storage Z, and passes the larger to the right neighbor



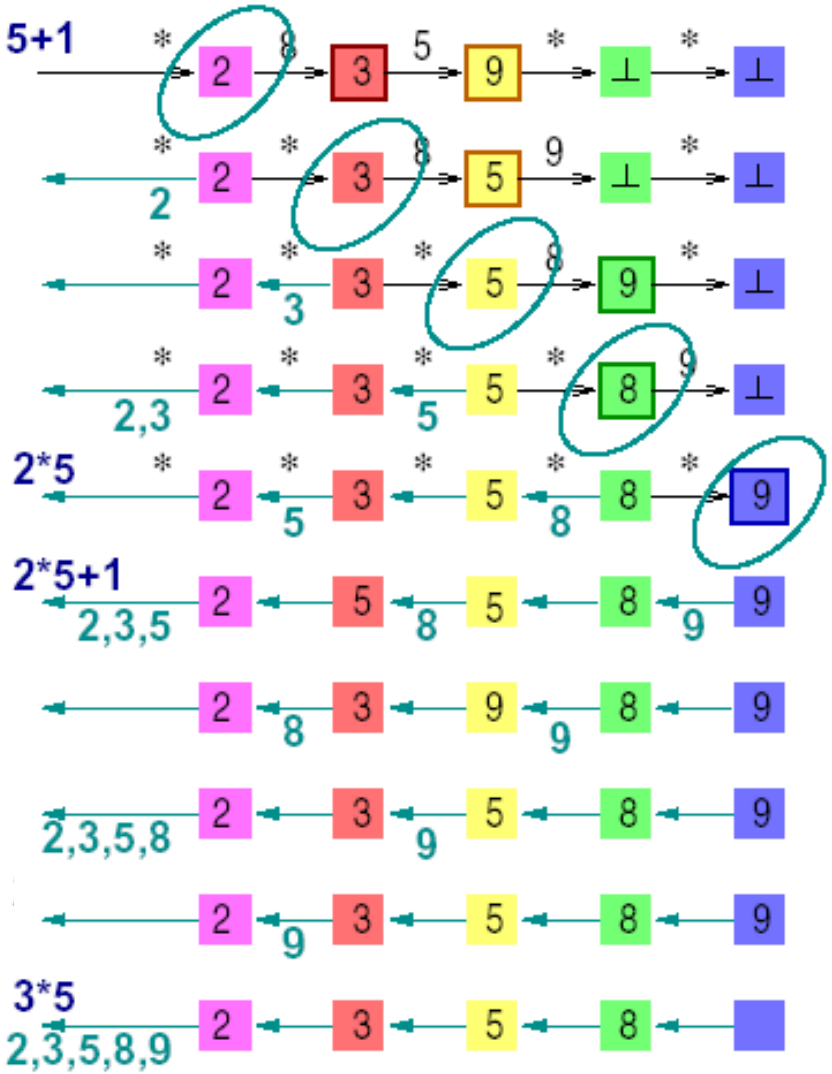
Sorting Example



Result Output

- The output process begins when a processor receives a non-∗, followed by a “∗”
- Each processor forwards its storage to its left neighbor and subsequent data it receives from right neighbors
- How many steps does it take to sort N numbers?
- What is the speedup and efficiency?

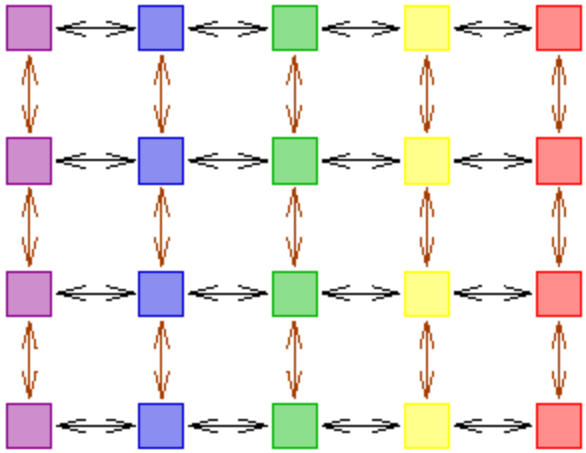
Output Example



Bit Model

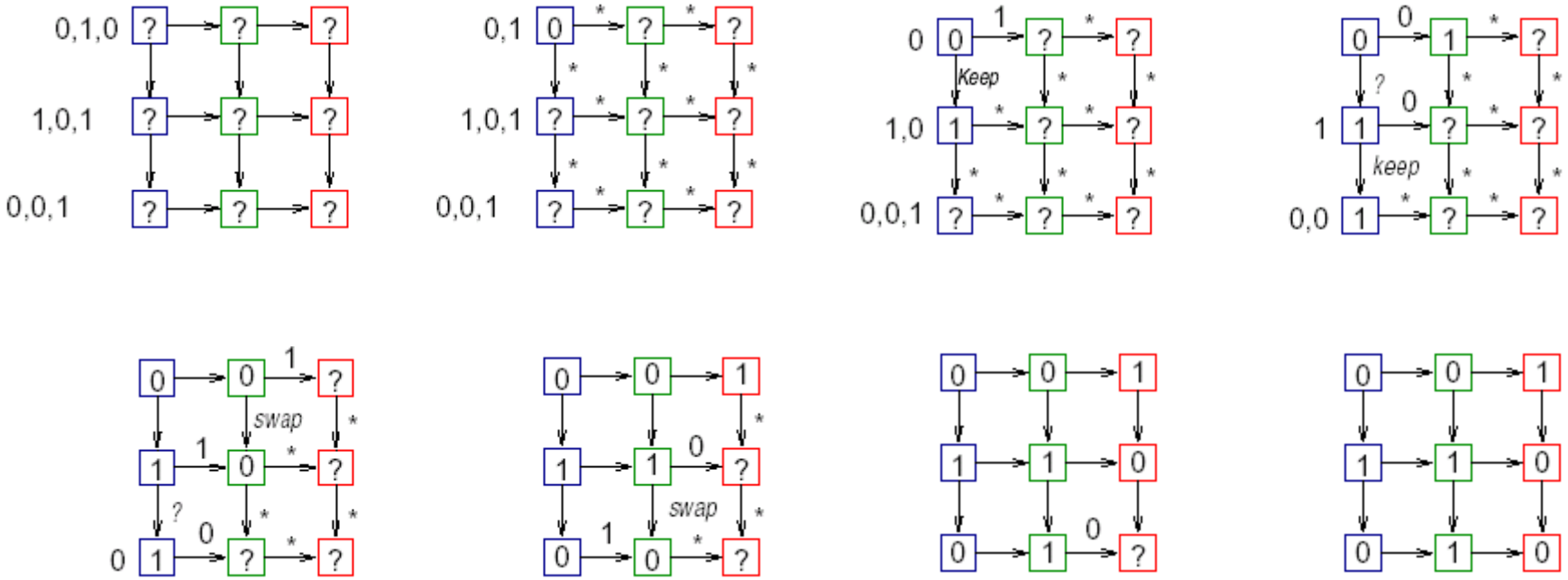
- The bit model affords a more precise measure of complexity – we will now assume that each processor can only operate on a bit at a time

- To compare N k-bit words, you may now need an N x k 2-d array of bit processors



Pipelined Comparison

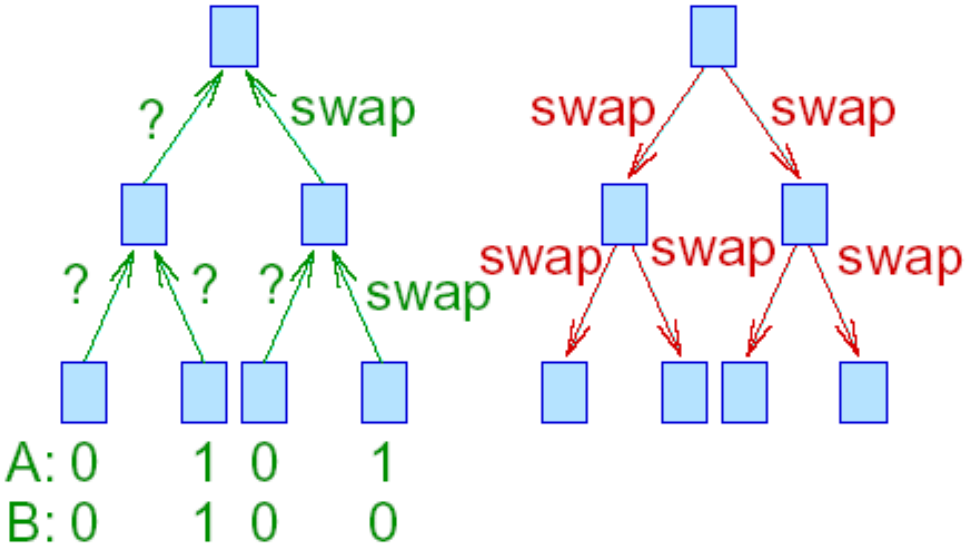
Input numbers: 3 4 2
0 1 0
1 0 1
1 0 0



Comparison Strategies

- Strategy 1: Bits travel horizontally, keep/swap signals travel vertically; if inputs arrive from the left, the array is sorted in 2N + k steps

- Strategy 2: Use a tree to communicate information on which number is greater – can set up a pipeline so the sorting happens in 2N + logk steps

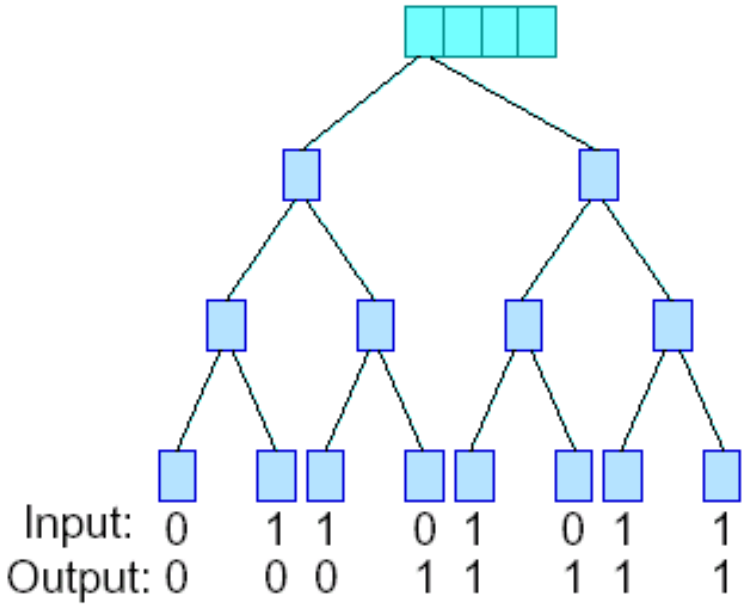


Lower Bounds

- Input/Output bandwidth: Nk bits are being input/output with k pins – requires $\Omega(N)$ time
- Diameter: the comparison at processor (1,1) influences the value of the bit stored at processor (N,k) – for example, N-1 numbers are 011..1 and the last number is either 00…0 or 10…0 – it takes at least N+k-2 steps for information to travel across the diameter
- Bisection width: if processors in one half require the results computed by the other half, the bisection bandwidth imposes a minimum completion time

Counter Example

- N 1-bit numbers that need to be sorted with a binary tree
- Since bisection bandwidth is 2 and each number may be in the wrong half, will any algorithm take at least N/2 steps?



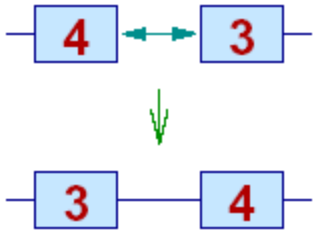
Counting Algorithm

- It takes O(logN) time for each intermediate node to add the contents in the subtree and forward the result to the parent, one bit at a time
- After the root has computed the number of 1's, this number is communicated to the leaves – the leaves accordingly set their output to 0 or 1
- Each half only needs to know the number of 1's in the other half (logN-1 bits) – therefore, the algorithm takes Ω(logN) time
- Careful when estimating lower bounds!

Sorting with Comparison Exchange

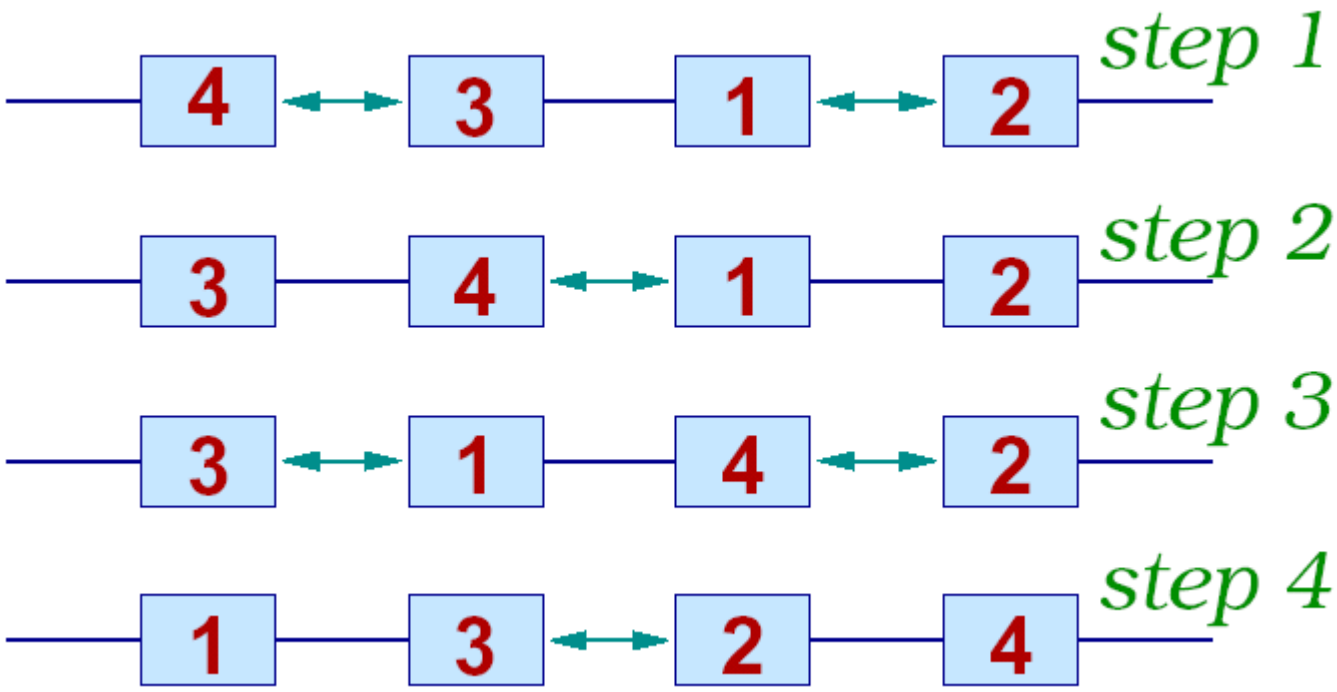
- Earlier sort implementations assumed processors that could compare inputs and local storage, and generate an output in a single time step

- The next algorithm assumes comparison-exchange processors: two neighboring processors I and J (I < J) show their numbers to each other and I keeps the smaller number and J the larger



Odd-Even Sort

- N numbers can be sorted on an N-cell linear array in O(N) time: the processors alternate operations with their neighbors



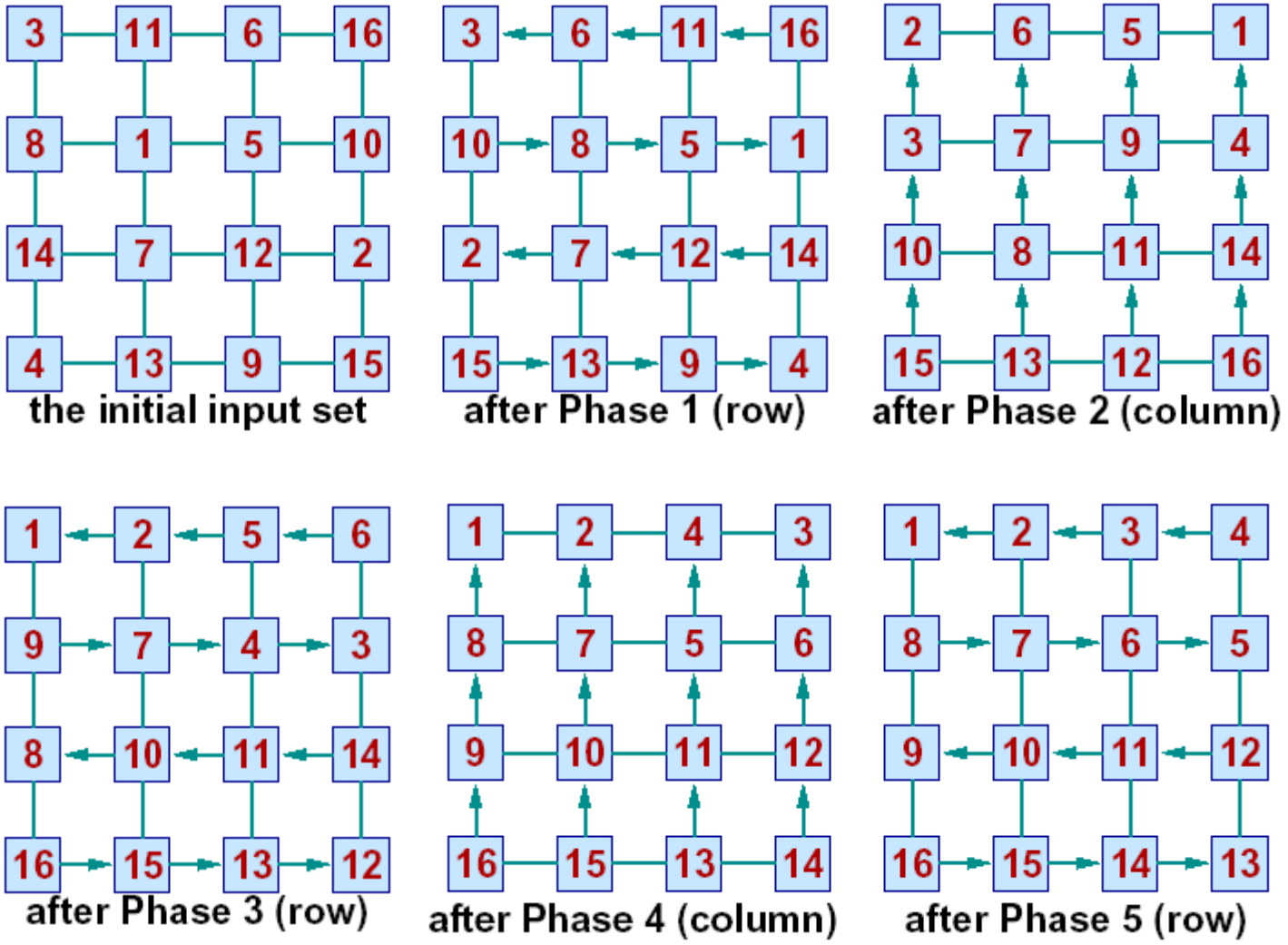
Shearsort

- A sorting algorithm on an N-cell square matrix that improves execution time to O(sqrt(N) logN)

- Algorithm steps:
  Odd phase: sort each row with odd-even sort (all odd rows are sorted left to right and all even rows are sorted right to left)
  Even phase: sort each column with odd-even sort
  Repeat

- Each odd and even phase takes O(sqrt(N)) steps – the input is guaranteed to be sorted in O(logN) steps

Example


3 11 6 16
8 1 5 10
14 7 12 2
4 13 9 15
the initial input set
3 6 11 16
10 8 5 1
2 7 12 14
15 13 9 4
after Phase 1 (row)
2 6 5 1
3 7 9 4
10 8 11 14
15 13 12 16
after Phase 2 (column)
1 2 5 6
9 7 4 3
8 10 11 14
16 15 13 12
after Phase 3 (row)
1 2 4 3
8 7 5 6
9 10 11 12
16 15 13 14
after Phase 4 (column)
1 2 3 4
8 7 6 5
9 10 11 12
16 15 14 13
after Phase 5 (row)

The 0-1 Sorting Lemma

If a comparison-exchange algorithm sorts input sets consisting solely of 0's and 1's, then it sorts all input sets of arbitrary values

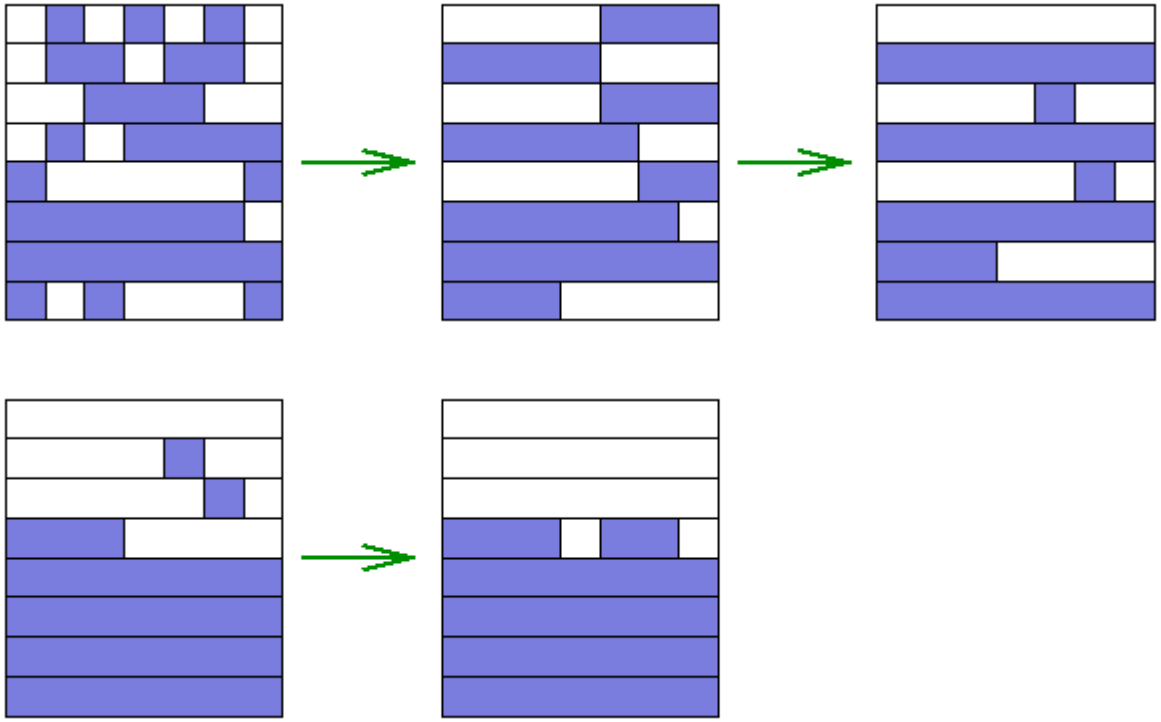
Proof Let an o.c.e. algorithm $\mathcal{A}$ for input size N be given. Suppose that $\mathcal{A}$ works on all 0-1 inputs. Assume that $\mathcal{A}$ fails on an input $[x_1, \ldots, x_N]$ with $[y_1, \ldots, y_N]$ be the output. Then $y_1 \leq \cdots \leq y_m > y_{m+1}$ for some m. Define mapping F by $F(x) = 0$ if $x < y_m$ and 1 otherwise. Since F preserves the order $\leq$, the output of $\mathcal{A}$ on $[F(x_1), \ldots, F(x_N)]$ is $[F(y_1), \ldots, F(y_N)]$, and is of the form $[\ldots, 1, 0, \ldots]$ because of $y_m > y_{m+1}$. This is a contradiction.

Complexity Proof

- How do we prove that the algorithm completes in O(logN) phases? (each phase takes O(sqrt(N)) steps)
- Assume input set of 0s and 1s
- There are three types of rows: all 0s, all 1s, and mixed entries – we will show that after every phase, the number of mixed entry rows reduces by half
- The column sort phase is broken into the smaller steps below: move 0 rows to the top and 1 rows to the bottom; the mixed rows are paired up and sorted within pairs; repeat these small steps until the column is sorted

Example

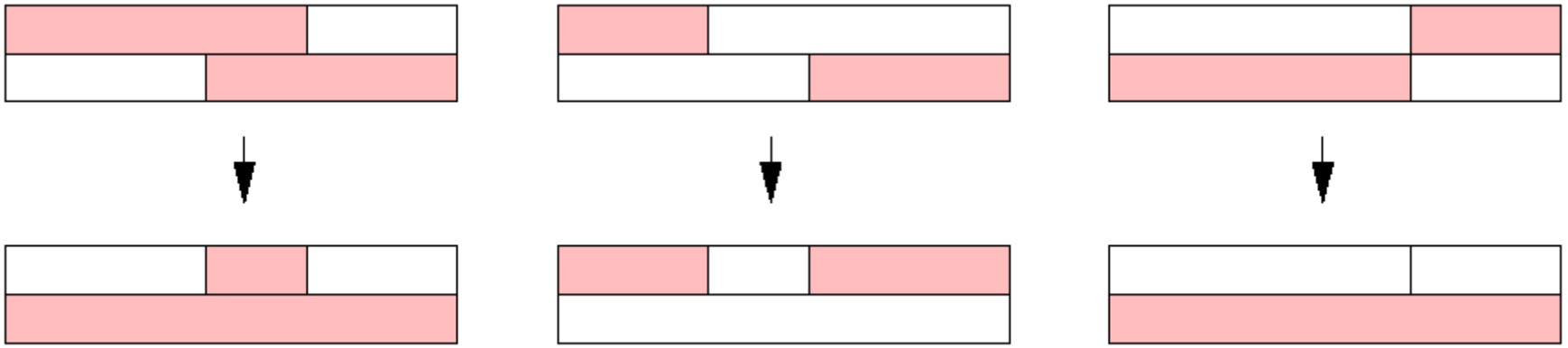
- The modified algorithm will behave as shown below:
  white depicts 0s and blue depicts 1s



Proof

- If there are N mixed rows, we are guaranteed to have fewer than N/2 mixed rows after the first step of the column sort (subsequent steps of the column sort may not produce fewer mixed rows as the rows are not sorted)

Each pair of mixed rows produces at least one pure row when sorted



Matrix Algorithms

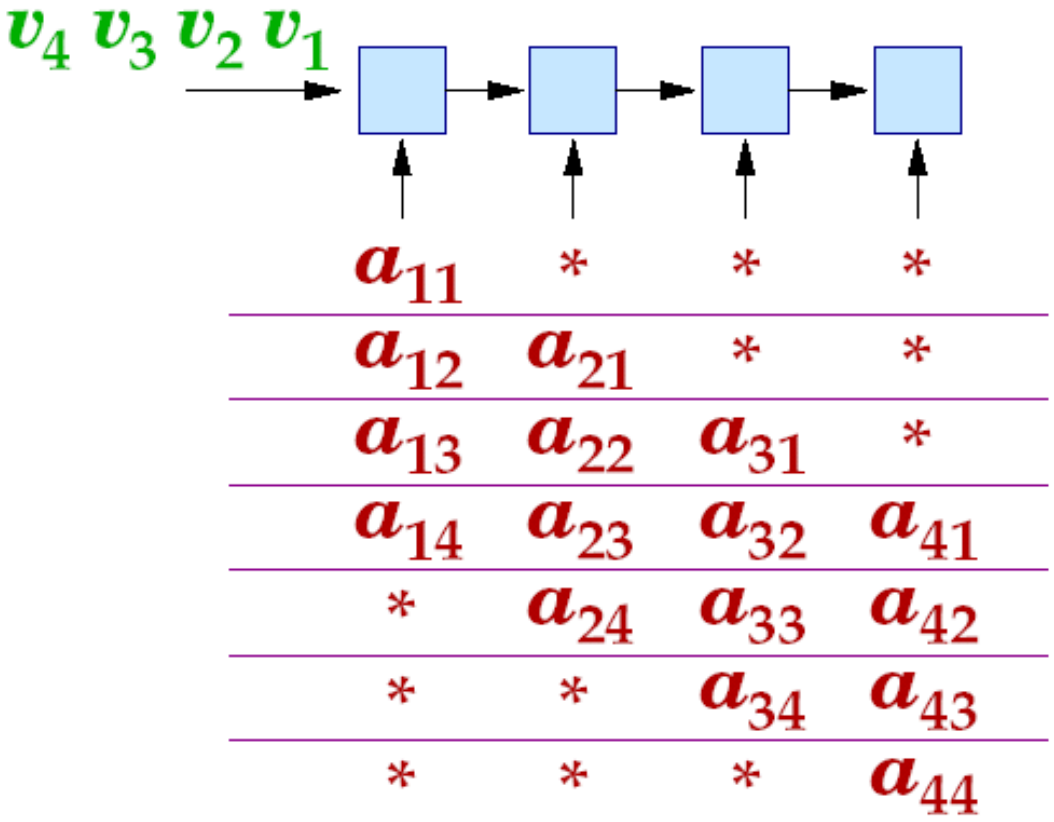
- Consider matrix-vector multiplication:

$$y_i = \Sigma_j \, a_{ij} x_j$$

- The sequential algorithm takes $2N^2 - N$ operations

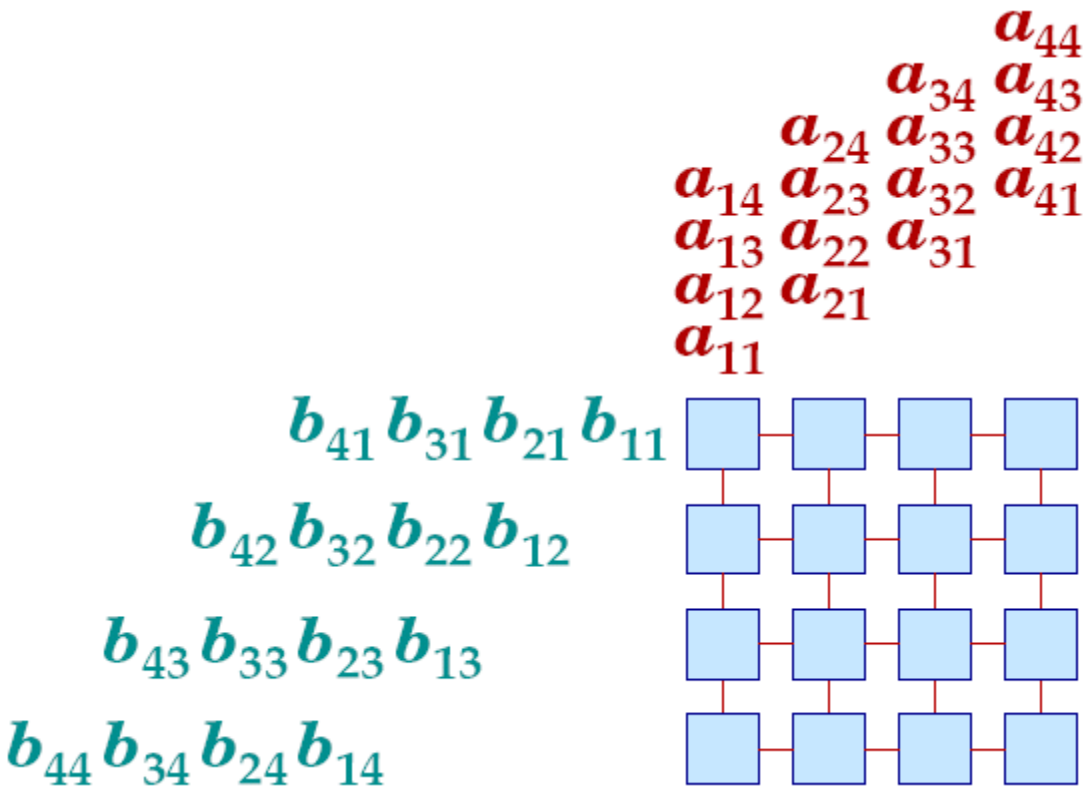
- With an N-cell linear array, can we implement matrix-vector multiplication in O(N) time?

Matrix Vector Multiplication



Number of steps = 2N – 1

Matrix-Matrix Multiplication

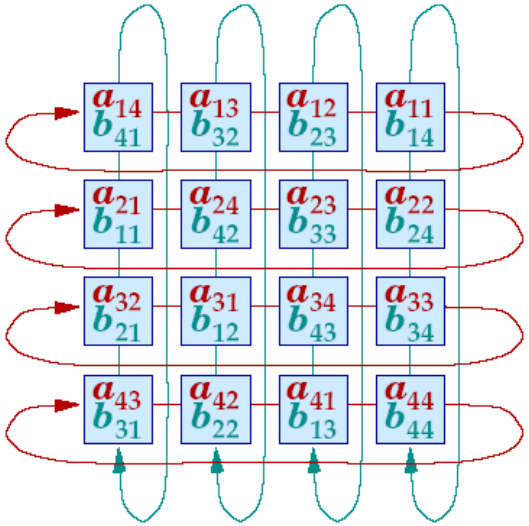


Number of time steps = 3N – 2

Complexity

- The algorithm implementations on the linear arrays have speedups that are linear in the number of processors – an efficiency of O(1)

- It is possible to improve these algorithms by a constant factor, for example, by inputting values directly to each processor in the first step and providing wraparound edges (N time steps)



References

- “Introduction to Parallel Algorithms and Architectures,” Leighton
- Figure credits: Mitsu Ogihara