





PARALLEL PROCESSING SYSTEMS

Chapter 7: Sorting and Selection Networks

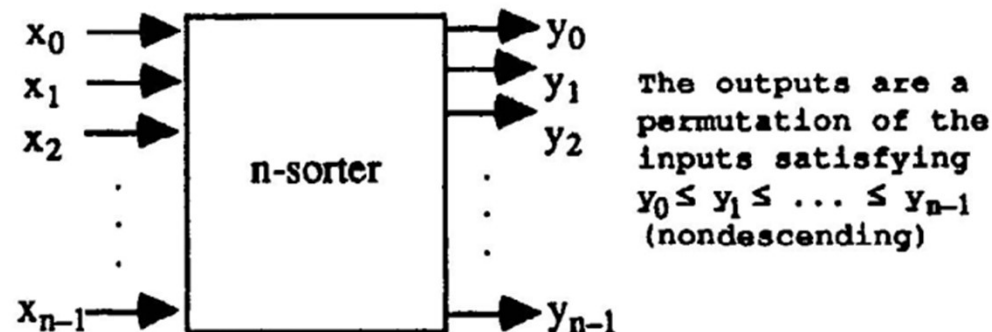


Chapter outline

- What is a sorting network?
 - Figures of merit for sorting networks
 - Design of sorting networks
 - Batcher sorting networks
 - Other classes of sorting networks
 - Selection networks
- 

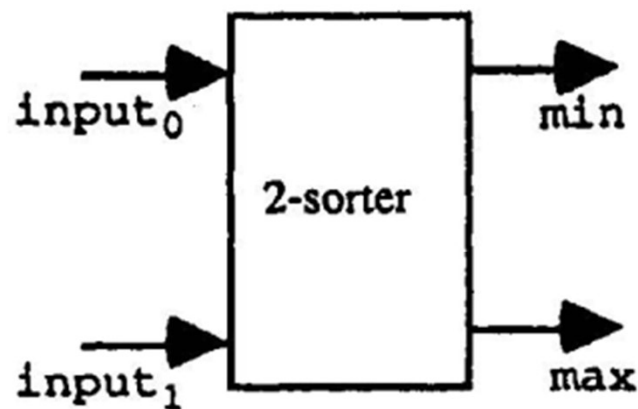
What is a sorting network

- A sorting network is a circuit that
 - receives n inputs, $x_0, x_1, x_2, \dots, x_{n-1}$
 - permutes them to produce n outputs, $y_0, y_1, y_2, \dots, y_{n-1}$
 - such that the outputs satisfy $y_0 \leq y_1 \leq y_2 \leq \dots \leq y_{n-1}$.
- For brevity
 - we often refer to such a sorting network as an n -sorter
- many sorting algorithms are based on comparing and exchanging pairs of keys
 - we can build an n -sorter out of 2-sorter building blocks

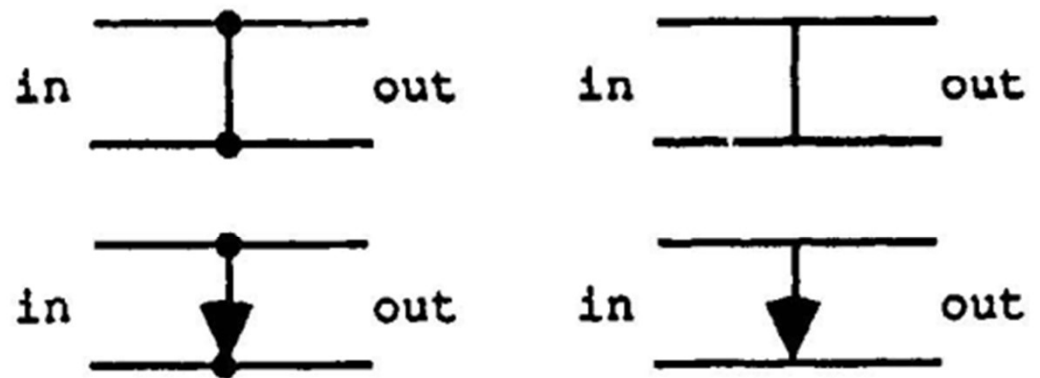


What is a sorting network

- 2-sorter
 - compares its two inputs
 - orders them at the output, by the smaller value before the larger value



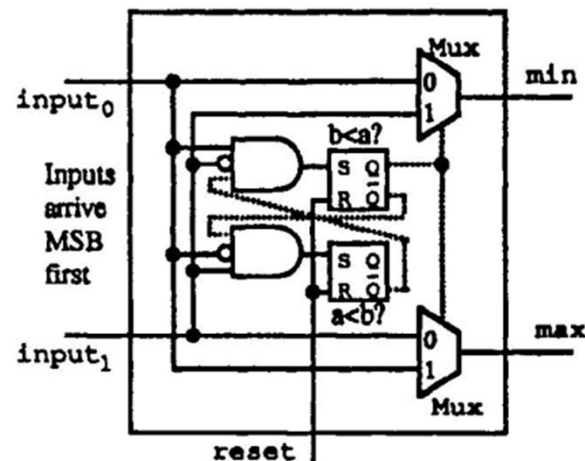
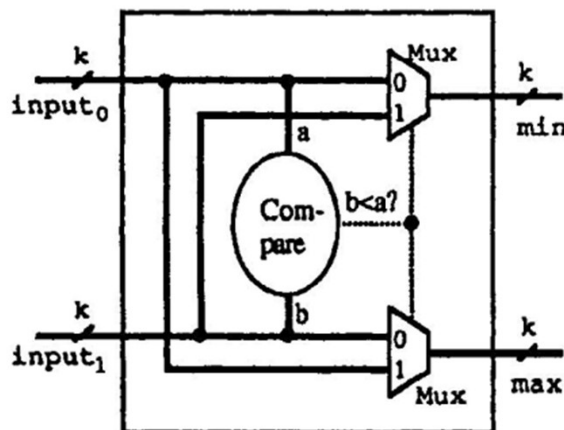
Block Diagram



Alternate Representations

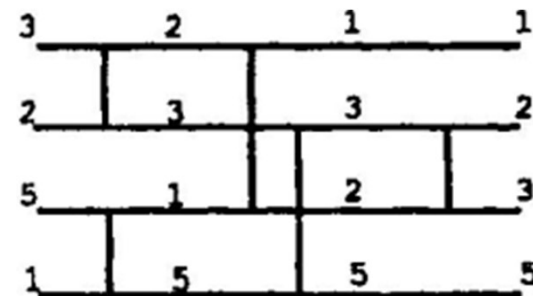
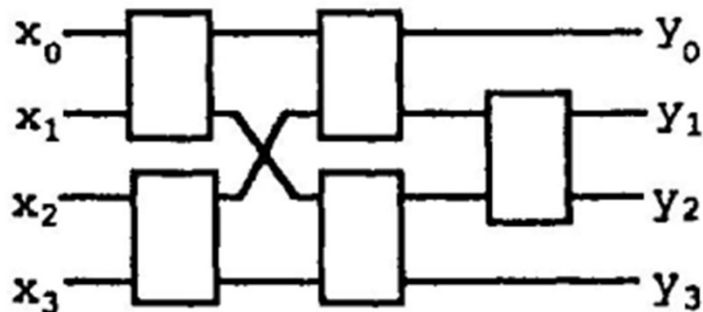
What is a sorting network

- 2-sorter hardware realization
 - If we view inputs as unsigned integers in bit-parallel form
 - can be implemented using
 - a comparator
 - and two 2-to-1 multiplexers
 - If bit-parallel input are impractical
 - the keys are long
 - or we have pin limitation on a single VLSI chip
 - Can be implemented using
 - two state flip-flops.
 - The flip-flops state 00 represents the two inputs being equal
 - The state of 01 means the upper input is less
 - The state of 10 means the lower input is less
 - in the state 00 or 01
 - the inputs are passed to the outputs straight through
 - in the state 10
 - the inputs are interchanged



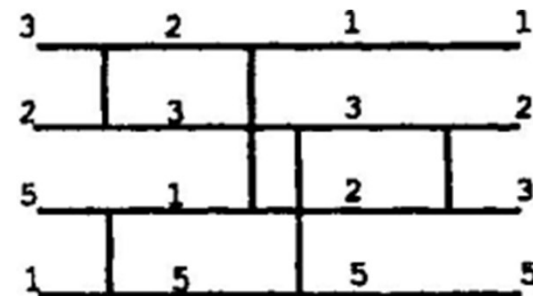
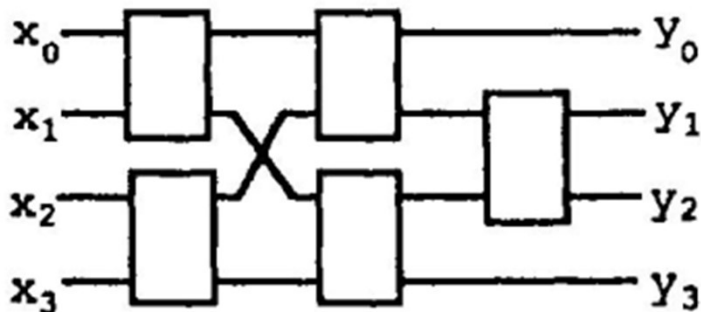
What is a sorting network

- 4-sorter built of 2-sorter building blocks
 - How do we verify the circuit?
 - easy in this case
 - After the first two circuit levels
 - the top line carries the smallest
 - the bottom line carries the largest
 - The final 2-sorter orders the middle two values
 - More generally, we need to verify through
 - Tedious formal proofs
 - or by time-consuming exhaustive testing
 - Neither approach is attractive
 - We can use the zero-one principle



What is a sorting network

- The Zero–One Principle
 - An n -sorter is valid if it correctly sorts all 0/1 sequences of length n .
 - Example
 - The network
 - clearly sorts 0000 and 1111.
 - sorts all sequences with a single 0
 - the 0 “bubbles up” to the top line.
 - sorts all sequences with a single 1
 - the 1 “sink down” to the bottom line
 - sorts the all the sequences 0011, 0101, 0110, 1001, 1010, 1100
 - to the correct output 0011



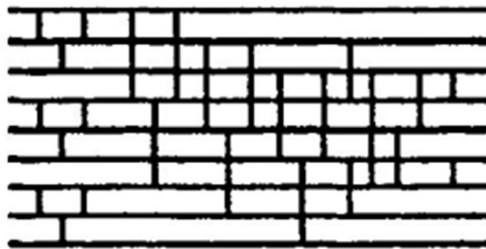


Figures of merit for sorting networks

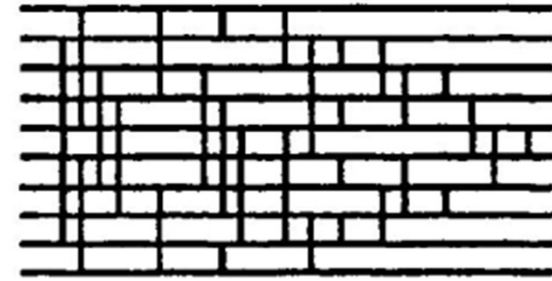
- Two figures of merit for “the best n-sorter”
 - Cost
 - the total number of 2-sorter blocks used
 - Delay
 - the number of 2-sorters on the critical path
- We can also use composite figures of merit
 - minimizing cost \times delay
 - if we expect linear speed-up from more investment in the circuit
 - E.g., redesigning a network to 10% faster but only 5% more complex is deemed to be cost-effective
 - the resulting circuit is said to be time-cost-efficient

Figures of merit for sorting networks

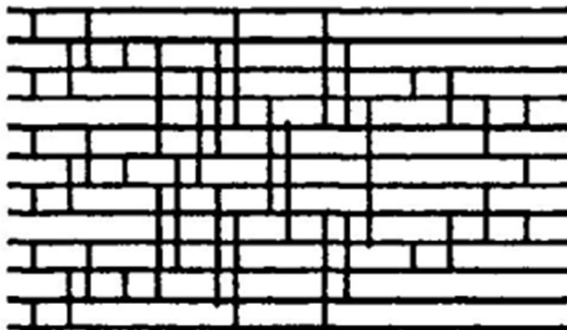
- examples of low-cost sorting networks
 - lowest-cost designs are known only for small n
 - no general method for systematically deriving low-cost designs



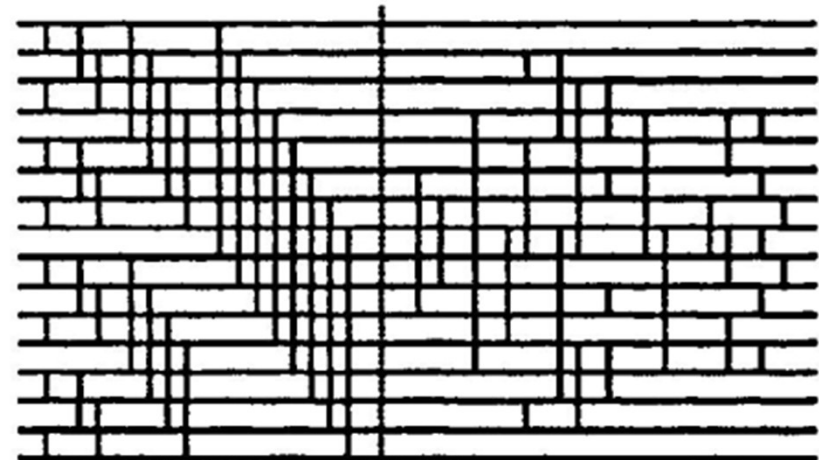
$n = 9, 25$ modules, 9 levels



$n = 10, 29$ modules, 9 levels



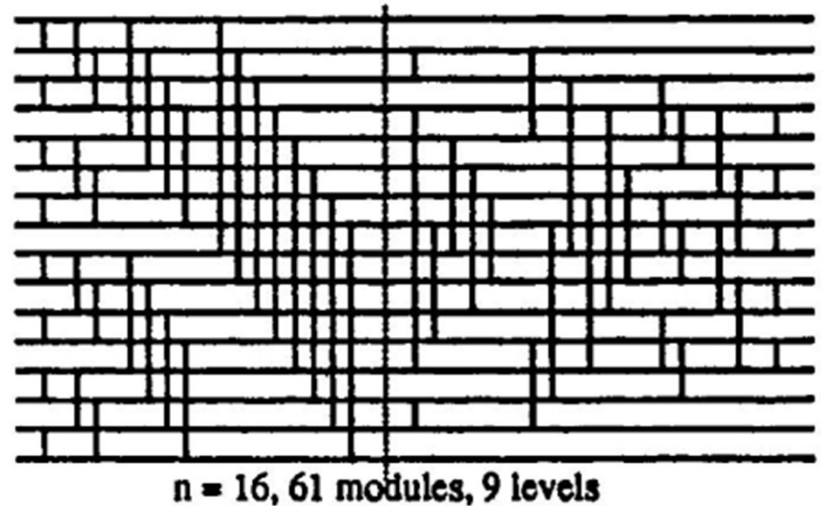
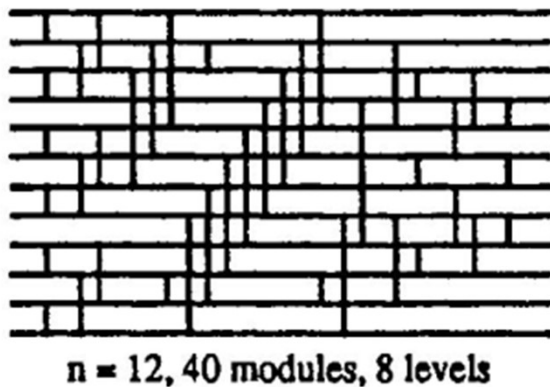
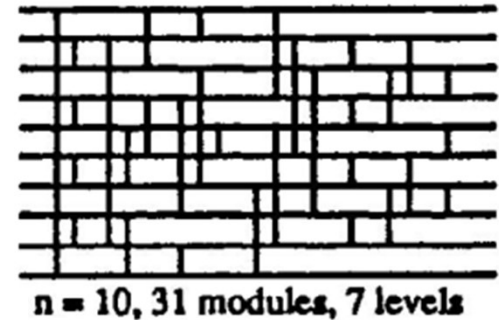
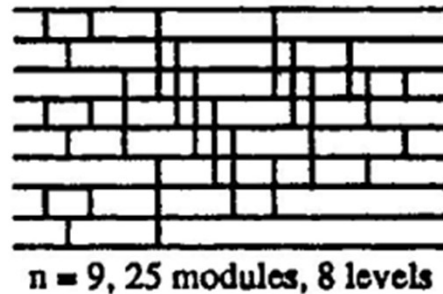
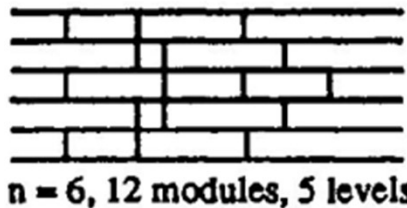
$n = 12, 39$ modules, 9 levels



$n = 16, 60$ modules, 10 levels

Figures of merit for sorting networks

- examples of fast sorting networks
 - possible designs are also known only for small n



Figures of merit for sorting networks

- Time-cost-efficient sorting networks are even harder to come by
 - For the 10-input examples
 - 29 modules, 9 delay units $\text{cost} \times \text{delay} = 261$
 - 31 modules, 7 delay units $\text{cost} \times \text{delay} = 217$
 - in general, the most time-cost-efficient design
 - may be neither the fastest nor the least complex n-sorter.

Design of sorting networks

- many ways to design sorting networks
 - leading to different results
- a 6-sorter based on the odd–even transposition
 - discussed in sorting on a linear array in Section 2.3
 - quite inefficient
 - it uses $n \lfloor n/2 \rfloor$ modules
 - has n units of delay.
 - Its cost \times delay product is $\Theta(n^3)$.

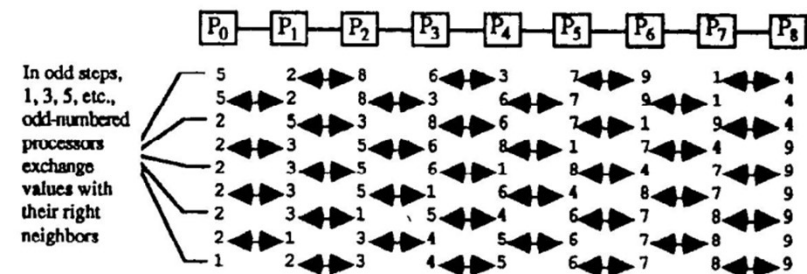
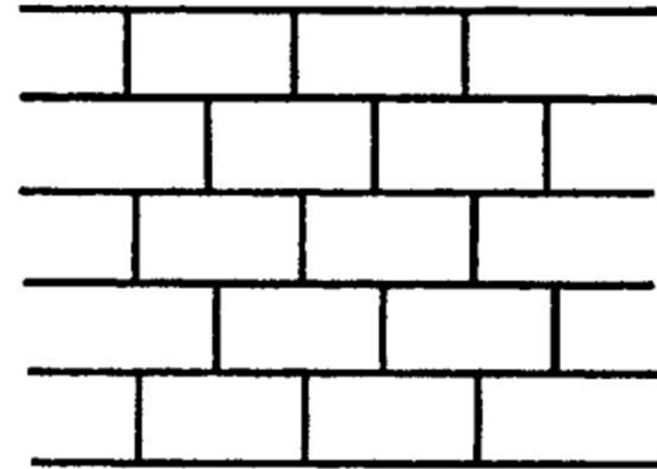
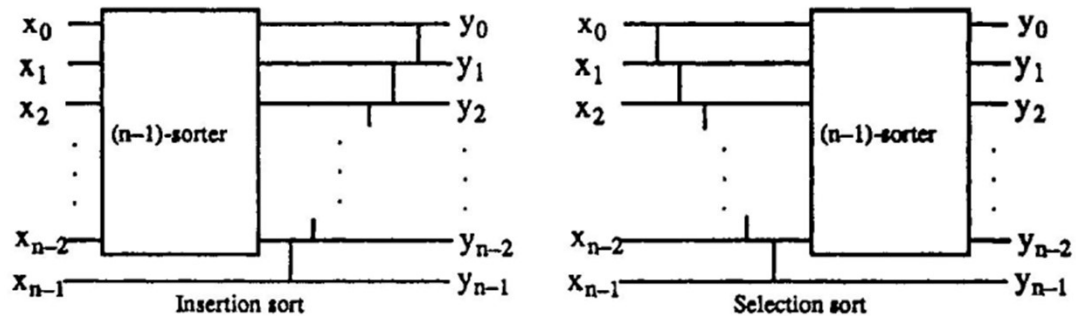


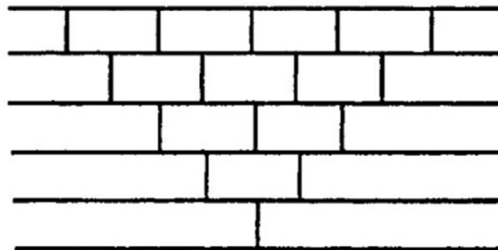
Figure 2.10. Odd–even transposition sort on a linear array.

Design of sorting networks

- One way to sort n inputs
 - sort the first $n - 1$ inputs
 - then insert the last input in its proper place
- Another way
 - select the largest value among the n inputs
 - output it on the bottom line
 - then sort the remaining $n - 1$ values
- both lead to the same design
 - which is in effect based on the parallel version of bubblesort



Parallel insertion sort = Parallel selection sort = Parallel bubble sort!



Design of sorting networks

- One way to sort n inputs
 - sort the first $n - 1$ inputs
 - then insert the last input in its proper place
- Another way
 - select the largest value among the n inputs
 - output it on the bottom line
 - then sort the remaining $n - 1$ values

$$C(n) = C(n - 1) + n - 1 = (n - 1) + (n - 2) + \dots + 2 + 1 = n(n - 1)/2$$

$$D(n) = D(n - 1) + 2 = 2 + 2 + \dots + 2 + 1 = 2(n - 2) + 1 = 2n - 3$$

$$\text{Cost} \times \text{Delay} = n(n - 1)(2n - 3)/2 = \Theta(n^3)$$

- Both are quite inefficient
 - Lower bounds are:
 - Cost : $\Omega(n \log n)$
 - Delay: $\Omega(\log n)$



Design of sorting networks

- Can we achieve those lower bounds?
 - if both bounds are achieved simultaneously
 - $\text{cost} \times \text{delay}$ product will be $\Theta(n \log^2 n)$
 - which is more than the sequential lower bound on work
 - but this is the best we can hope for
 - AKS sorting network
 - $O(n \log n)$ -cost, $O(\log n)$ -delay
 - is of theoretical interest only
 - as the asymptotic notation hides huge four-digit constants




Design of sorting networks

- Can we achieve those lower bounds?
 - researchers have not given up hope
 - But work has diversified on other fronts
 - One is the design of efficient sorting networks with special inputs or outputs, for example
 - when inputs are only 0s and 1s
 - or they are already partially sorted
 - or we require only partially sorted outputs
 - Another is the networks that
 - sort the input sequence with high probability
 - but do not guarantee sorted order for all possible inputs
- Practical sorting networks are based on designs by Batcher and others
 - have $O(n \log^2 n)$ cost and $O(\log^2 n)$ delay.
 - are a factor of $\log n$ away from being asymptotically optimal
 - but $\log_2 n$ is only 20 when n is as large as 1 million




Batcher sorting networks

- Batcher's ingenious constructions
 - date back to the early 1960s
 - constitute some of the earliest examples of parallel algorithms.
 - in more than three decades
 - only small improvements have been made
- 



Batcher sorting networks

- One Batcher network
 - is based on the idea of an (m, m') -merger
 - uses a technique known as even–odd merge or odd–even merge
 - An (m, m') -merger is a circuit
 - merges two sorted sequences of lengths m and m'
 - produce a single sorted sequence of length $m + m'$
- 

Batcher sorting networks

- Let the two sorted sequences be

$$x_0 \leq x_1 \leq \dots \leq x_{m-1}$$

$$y_0 \leq y_1 \leq \dots \leq y_{m'-1}$$

- If $m = 0$ or $m' = 0$
 - then nothing needs to be done
- For $m = m' = 1$
 - a single comparator can do the merging
- we assume $mm' > 1$ in what follows

Batcher sorting networks

- The odd–even merge

- Is done by merging the even- and odd-indexed elements of the two lists

$x_0, x_2, \dots, x_{2\lceil m/2 \rceil - 2}$ and

$y_0, y_2, \dots, y_{2\lceil m'/2 \rceil - 2}$ are merged to get

$v_0, v_1, \dots, v_{\lceil m/2 \rceil + \lceil m'/2 \rceil - 1}$

$x_1, x_3, \dots, x_{2\lfloor m/2 \rfloor - 1}$ and

$y_1, y_3, \dots, y_{2\lfloor m'/2 \rfloor - 1}$ are merged to get

$w_0, w_1, \dots, w_{\lfloor m/2 \rfloor + \lfloor m'/2 \rfloor - 1}$

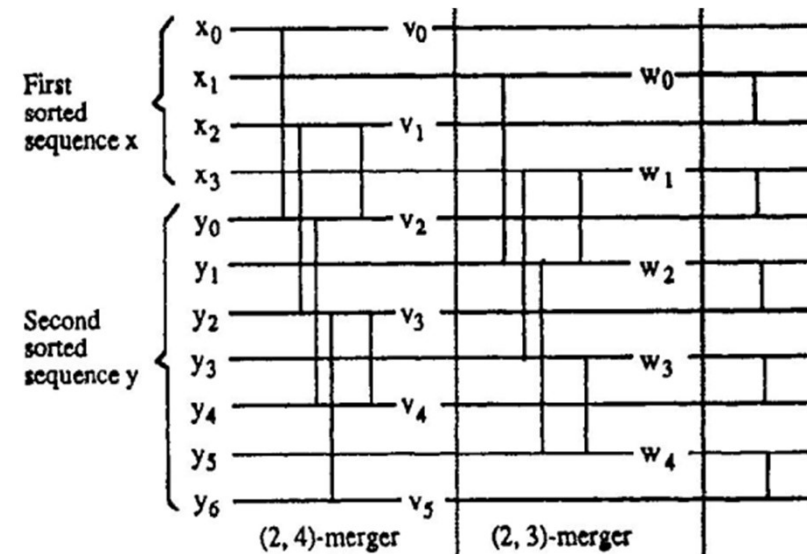
- If we now compare–exchange the pairs of

$w_0:v_1, w_1:v_2, w_2:v_3, \dots,$

- the resulting $v_0 w_0 v_1 w_1 v_2 w_2 \dots$ sequence will be completely sorted.

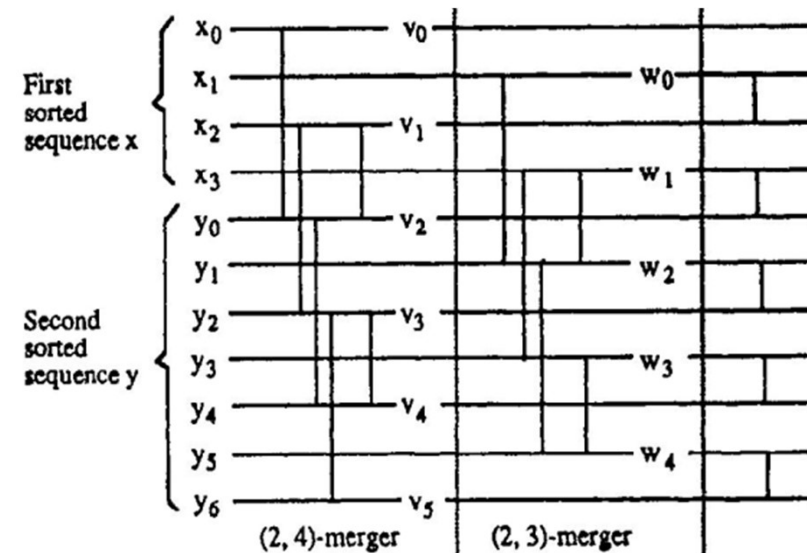
Batcher sorting networks

- example
 - merging two sorted lists of sizes 4 and 7
 - The three circuit segments correspond to
 - a (2, 4)-merger for even-indexed inputs
 - a (2, 3)-merger for odd-indexed inputs
 - and the final parallel compare–exchange operations



Batcher sorting networks

- Each of the smaller mergers can be designed recursively
 - a (2, 4)-merger consists of
 - two (1, 2)-mergers for even- and odd-indexed inputs
 - followed by two parallel compare-exchange operations
 - a (1, 2)-merger is built from
 - a (1, 1)-merger or a single comparator
 - for the even-indexed inputs
 - followed by a single compare-exchange operation.



Batcher sorting networks

- delay and cost for (m, m) even-odd merger
 - m is a power of 2

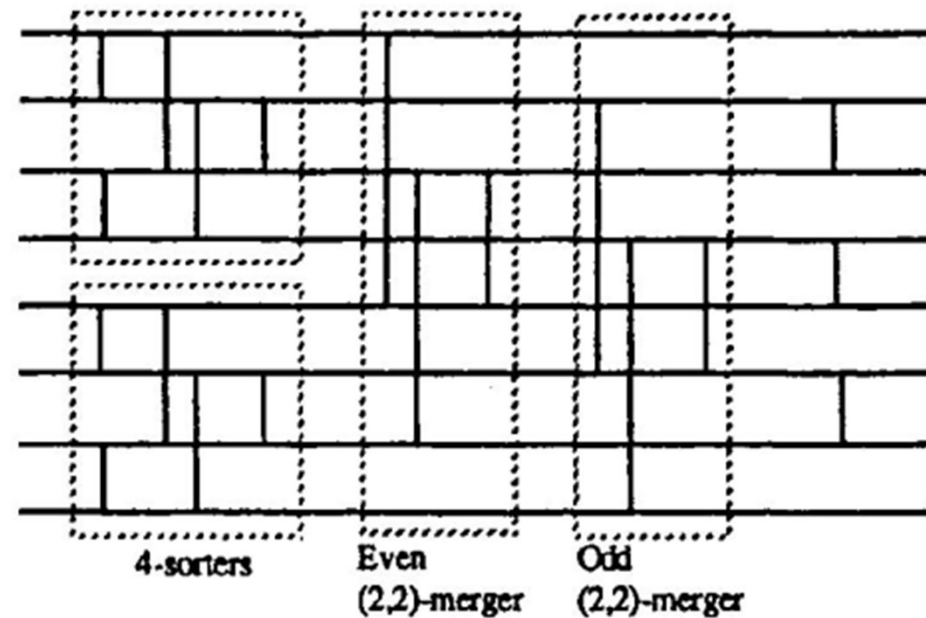
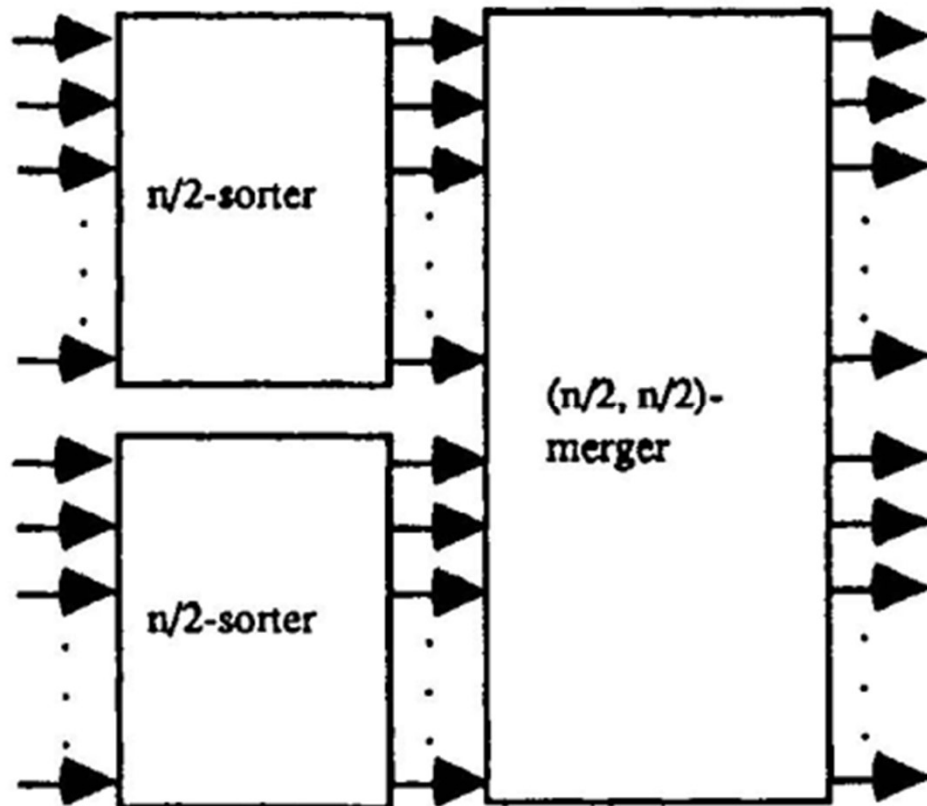
$$C(m) = 2C(m/2) + m - 1 = (m - 1) + 2(m/2 - 1) + 4(m/4 - 1) + \dots = m \log_2 m + 1$$

$$D(m) = D(m/2) + 1 = \log_2 m + 1$$

$$\text{Cost} \times \text{Delay} = \Theta(m \log^2 m)$$

Batcher sorting networks

- n -sorter using two $n/2$ -sorters and an $(n/2, n/2)$ -merger



Batcher sorting networks

- delay and cost for Batcher sorting networks
 - based on the even–odd merge technique

$$C(n) = 2C(n/2) + (n/2)(\log_2(n/2)) + 1 \approx n(\log_2 n)^2/2$$

$$D(n) = D(n/2) + \log_2(n/2) + 1 = D(n/2) + \log_2 n = \log_2 n (\log_2 n + 1)/2$$

$$\text{Cost} \times \text{Delay} = \Theta(n \log^4 n)$$

Batcher sorting networks

- Batcher network based on the notion of bitonic sequences
 - A bitonic sequence is defined as one that
 - “rises then falls” $(x_0 \leq x_1 \leq \dots \leq x_i \geq x_{i+1} \geq x_{i+2} \geq \dots \geq x_{n-1})$
 - “falls then rises” $(x_0 \geq x_1 \geq \dots \geq x_i \leq x_{i+1} \leq x_{i+2} \leq \dots \leq x_{n-1})$
 - or is obtained from the first two through cyclic shifts or rotations

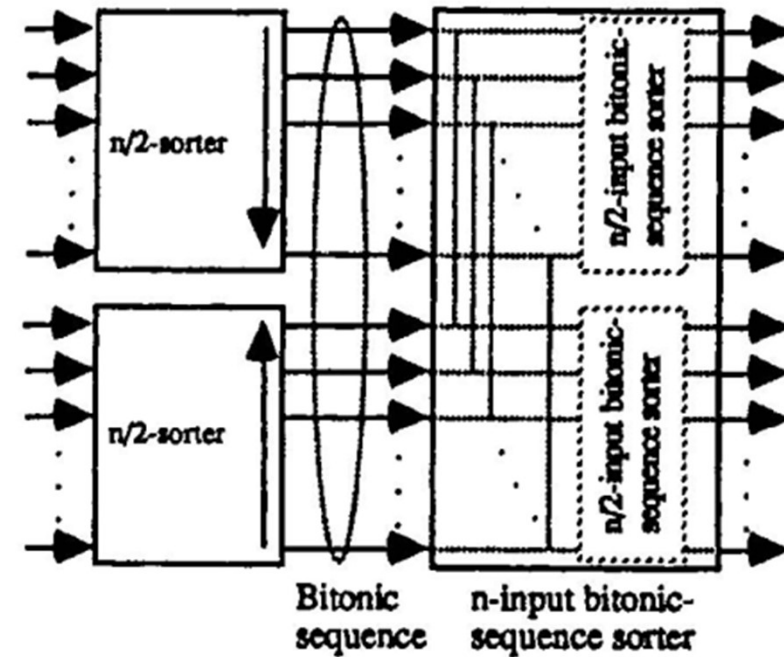
1 3 3 4 6 6 6 2 2 1 0 0 Rises then falls

8 7 7 6 6 6 5 4 6 8 8 9 Falls then rises

8 9 8 7 7 6 6 6 5 4 6 8 The previous sequence, right-rotated by 2

Batcher sorting networks

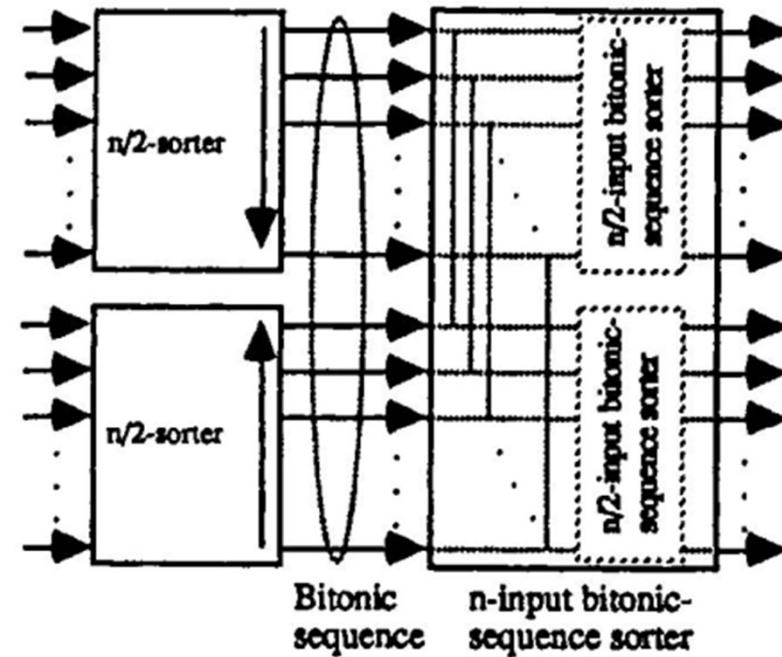
- bitonic Batcher sorting network
 - if we sort the first half and second half in opposite directions
 - the resulting sequence will be bitonic
 - can thus be sorted by a special bitonic-sequence sorter



Batcher sorting networks

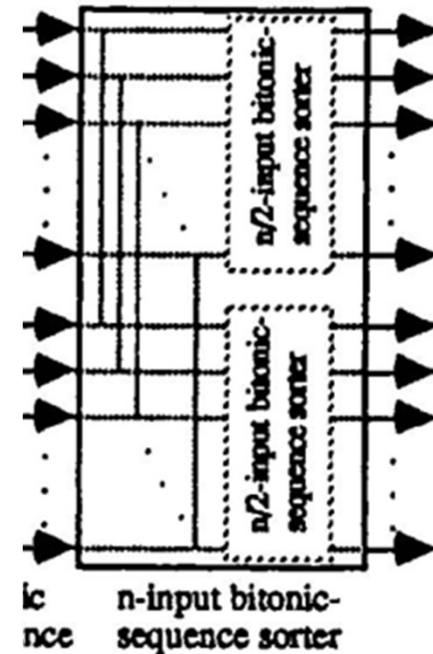
- bitonic Batcher sorting network

- A bitonic-sequence sorter with n
 - has the same delay and cost as an even-odd $(n/2, n/2)$ -merger.
- bitonic sorters have the same delay and cost as those based on even-odd merging



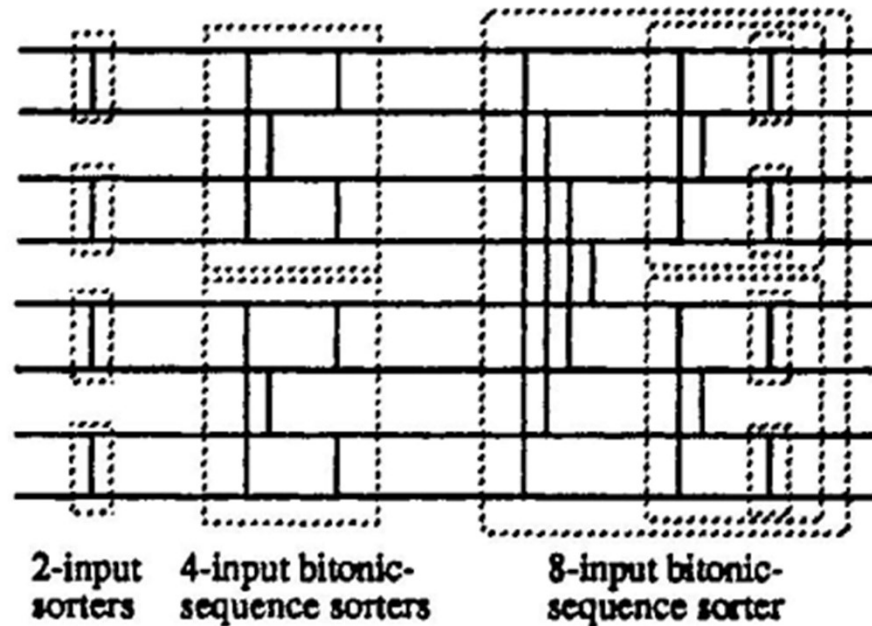
Batcher sorting networks

- bitonic Batcher sorting network
 - A bitonic-sequence sorter design
 - if we compare–exchange the elements in the first half with those in the second half
 - indicated by the dotted comparators
 - each half of the resulting sequence will be a bitonic sequence
 - each element in the first half will be no larger than any element in the second half
 - the two halves can be independently sorted
 - by smaller bitonic-sequence sorters



Batcher sorting networks


- 8 input bitonic Batcher sorting network



- Batcher sorting networks are quite efficient
 - when n is large
 - Only marginal improvements are obtained

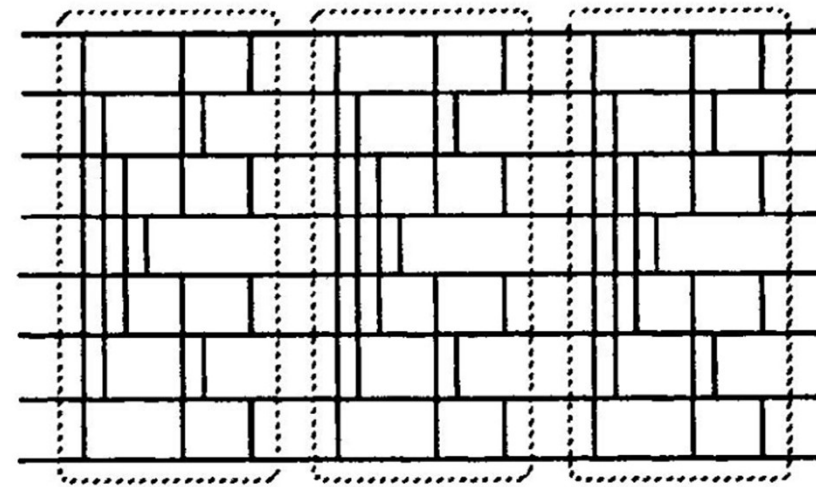


Other classes of sorting networks

- Periodic balanced sorting networks
 - possess the same asymptotic delay and cost as Batcher
 - consists of $\log_2 n$ identical stages
 - each is a $(\log_2 n)$ -stage n -input bitonic-sequence sorter
 - the delay and cost are $(\log_2 n)^2$ and $n (\log_2 n)^2/2$
- 


Other classes of sorting networks

- Periodic balanced sorting networks
 - larger delay (9 versus 6)
 - higher cost (36 versus 19)
 - offer some advantages
 - The structure is regular and modular
 - easier VLSI layout
 - Slower, but more economical
 - implementations are possible by reusing the blocks
 - Using an extra block provides tolerance to some faults
 - missed exchanges
 - Using two extra blocks provides tolerance to any single fault
 - a missed or incorrect exchange
 - Multiple passes through a faulty network can lead to correct sorting
 - graceful degradation
 - Single-block design can be made fault-tolerant by adding an extra stage to the block





Selection networks

- If we need the k th smallest value
 - using a sorting network would be an overkill
 - n -sorter does more than what is required
 - three selection problems
 - I. Select the k smallest values and present them on k outputs in sorted order.
 - the hardest
 - II. Select the k th smallest value and present it on one of the outputs.
 - III. Select the k smallest values and present them on k outputs in any order
 - the easiest
- 

Selection networks

- a type III (8, 4)-selector
 - pairs of integers denote the possible minimum and maximum rank

[0,7]	[0,6]	[0,4]	[0,4]	[0,3]	} Outputs
[0,7]	[1,7]	[1,6]	[1,5]	[1,3]	
[0,7]	[0,6]	[1,6]	[2,6]	[1,3]	
[0,7]	[1,7]	[3,7]	[3,7]	[0,3]	
[0,7]	[0,6]	[0,4]	[0,4]	[4,7]	
[0,7]	[1,7]	[1,6]	[1,5]	[4,6]	
[0,7]	[0,6]	[1,6]	[2,6]	[4,6]	
[0,7]	[1,7]	[3,7]	[3,7]	[4,7]	