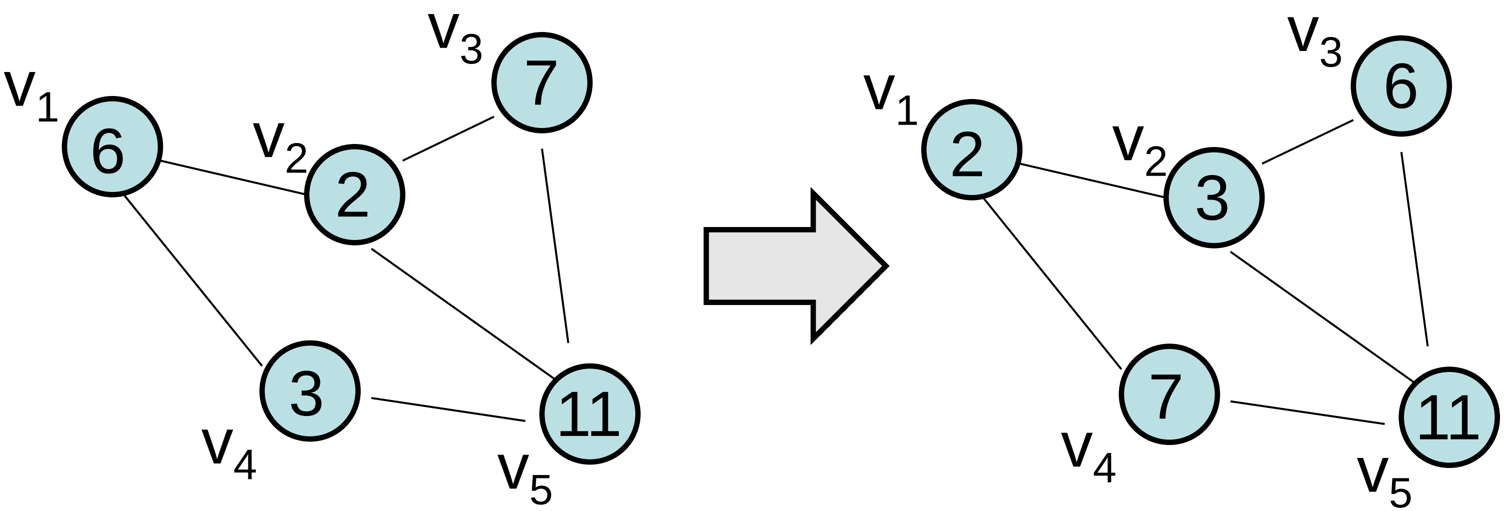
Network Algorithms

Distributed Sorting

Distributed Sorting

Graph with n nodes $\{v_1,\ldots,v_n\}$ and n values.
Goal: node vi should store i-th smallest value.



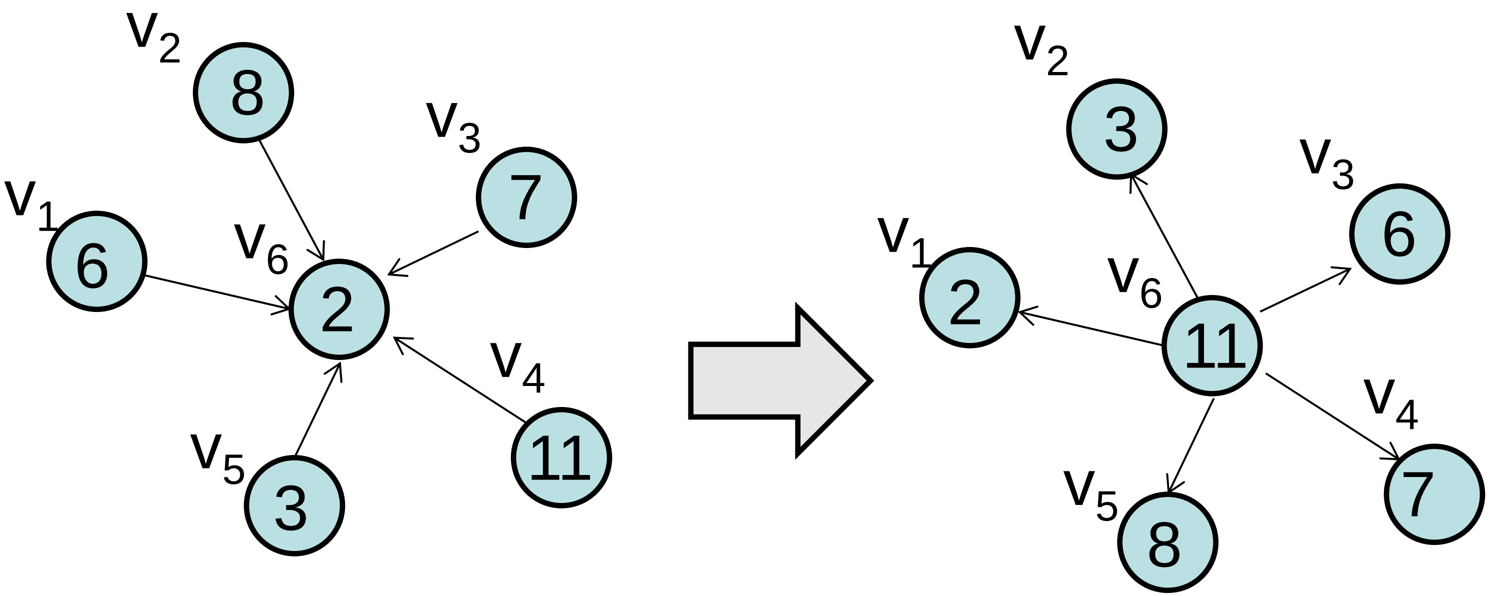
Simple solution?

Simple Solution

Simple Algo

Send to some node v, sorts it locally, redistributes!

Example on Star Graph:

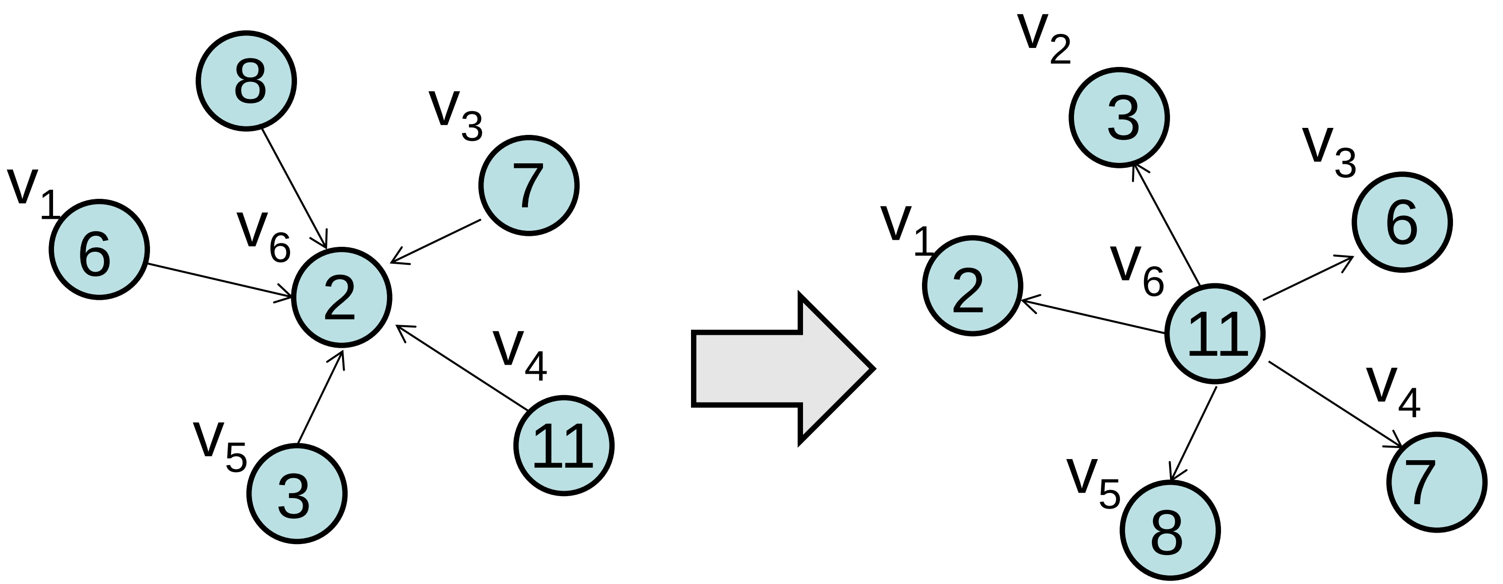


O(1) time, O(n) messages ☺ Problem?

Contention

Node Contention

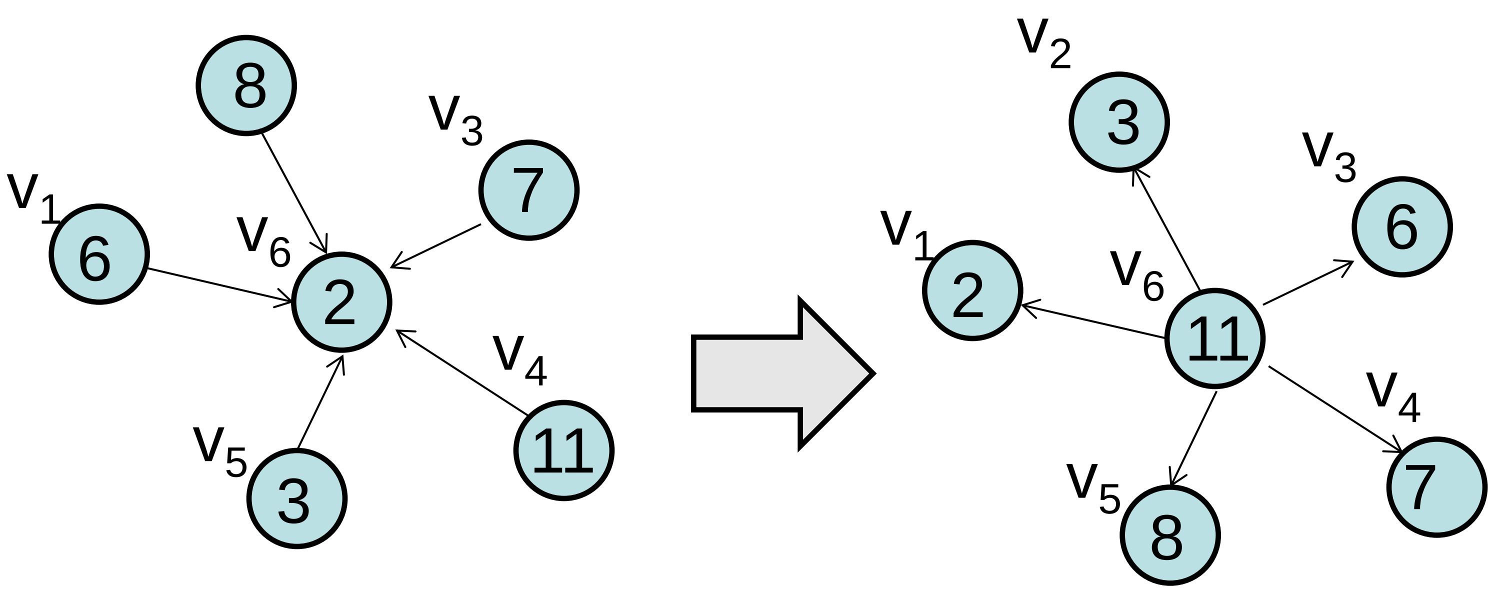
Nodes can only send and receive O(1) messages containing O(1) identifiers per node and round, independently of node degree!



Complexity to sort star graph?

Node Contention

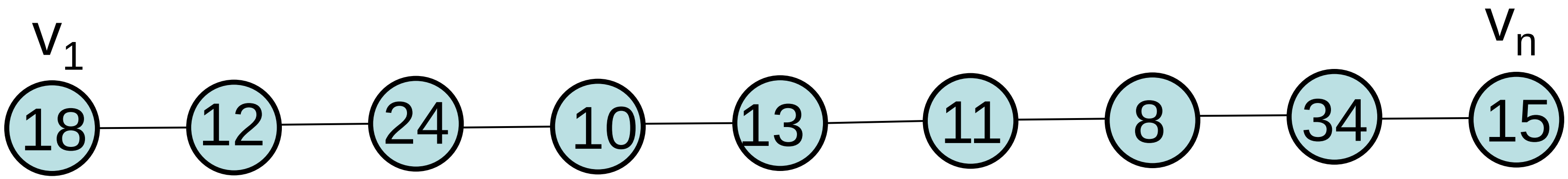
Nodes can only send and receive O(1) messages containing O(1) identifiers per node and round, independently of node degree!



Complexity to sort star graph? Ω(n) time! How to do it faster?

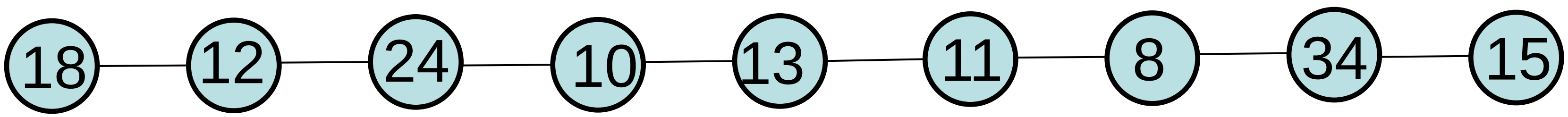
Array

How to sort in an array?



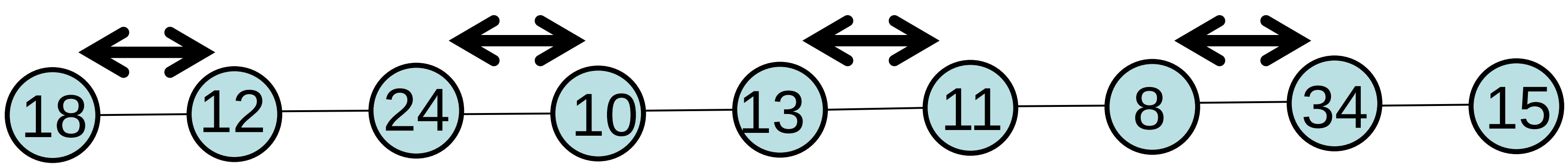
Array

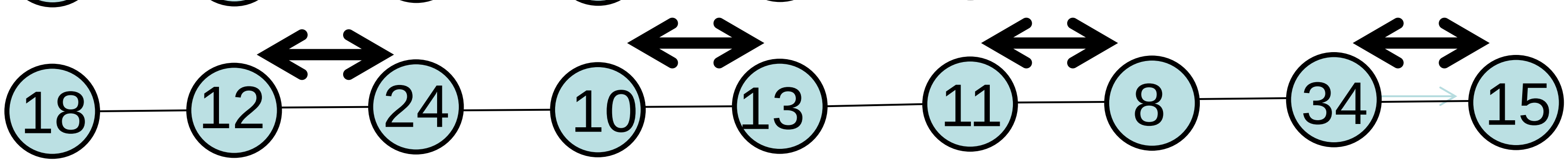
How to sort in an array?



Odd/Even Sort

1. Exchange values at node i and i+1, i odd
2. Exchange values at nodes i and i+1, i even
3. Loop until no exchanges needed anymore





Why correct? Congestion okay?

Largest value will eventually arrive on right,
second largest value will….
Congestion also okay.

Better proof: 0-1 Sorting Lemma

Remember it?

0-1 Sorting Lemma

01-Sorting Lemma

If an oblivious comparison-exchange algorithm sorts all inputs of 0s and 1s, then it sorts arbitrary inputs.

Oblivious = whether two elements are exchanged only depends on relative order, nothing else.

Proof (1): Equivalent: “If ALG does not sort some, then does not sort some 01 either!”

$$A \rightarrow B \iff \mathrm{not}(B) \rightarrow \mathrm{not}(A)$$

$$\Updownarrow \qquad\qquad\qquad\qquad \Updownarrow$$

not(A) or B   not(not(B)) or not(A)

0-1 Sorting Lemma

01-Sorting Lemma

If an oblivious comparison-exchange algorithm sorts all inputs of 0s and 1s, then it sorts arbitrary inputs.

Oblivious = whether two elements are exchanged only depends on relative order, nothing else.

Proof (2):

- Assume: $x_1,\ldots,x_n$ not sorted correctly by ALG.
- After wrong sorting, find smallest value k at some node v_k such that $k > x_k$. (Smallest value at a wrong node.)
- Define a binary input: $x^*_i=0$ if $x_i \leq x_k$, $x^*_i=1$ else.
- When oblivious ALG exchanges
  - <0,0> or <1,1>: does not matter
  - Exchange $x^*_i = 0$, $x^*_j = 1$ implies that $x_i \leq x_k < x_j$ (ALG oblivious)
- So x and x* are sorted the same way!

■

01-Sorting Lemma

If an oblivious comparison-exchange algorithm sorts all inputs of 0s and 1s, then it sorts arbitrary inputs.

Oblivious = whether two elements are exchanged only depends on relative order, nothing else.

Proof (2):

- Assume: $x_1,\ldots,x_n$ not sorted correctly by ALG.
- After wrong sorting, find smallest value k at some node v_k such that $k > x_k$. (Smallest value at a wrong node.)
- Define a binary input: $x^*_i=0$ if $x_i \leq x_k$, $x^*_i=1$ else.
- When oblivious ALG exchanges
  - <0,0> or <1,1>: does not matter
  - Exchange $x^*_i = 0$, $x^*_j = 1$ implies that $x_i \leq x_k < x_j$ (ALG oblivious)
- So x and x* are sorted the same way!

■

Runtime also the same.

0-1 Sorting Lemma

Array Sort

Odd/Even Sort sort is correct. Runtime: n steps.

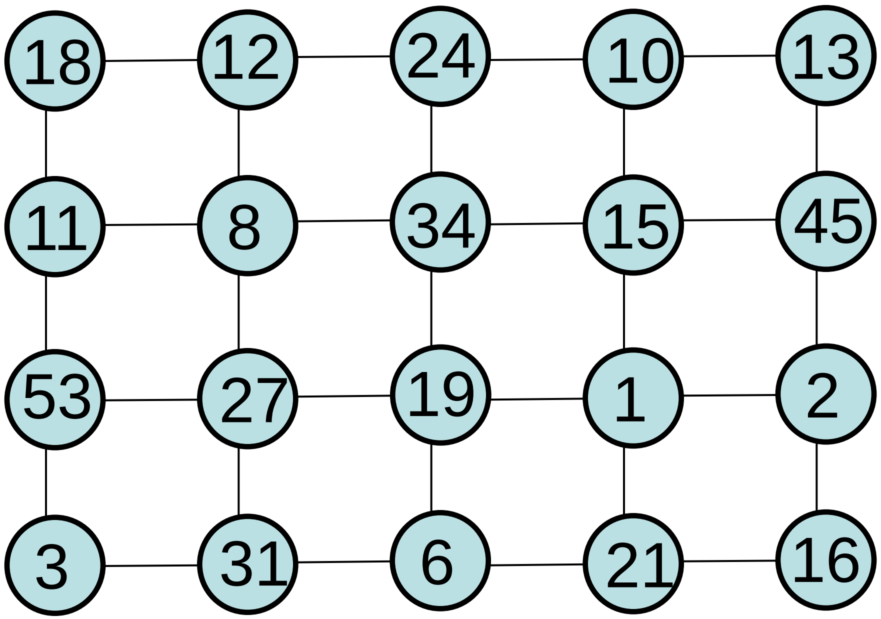
Proof: Can focus on 01-inputs only!

- Let j_1 be the index of the node with the rightmost “1”.
- Either j_1 index will grow in odd or even step for first time.
- And from then on always, until v_n reached.
- Also index of k-th most “1” is increasing in each step: by induction.

■

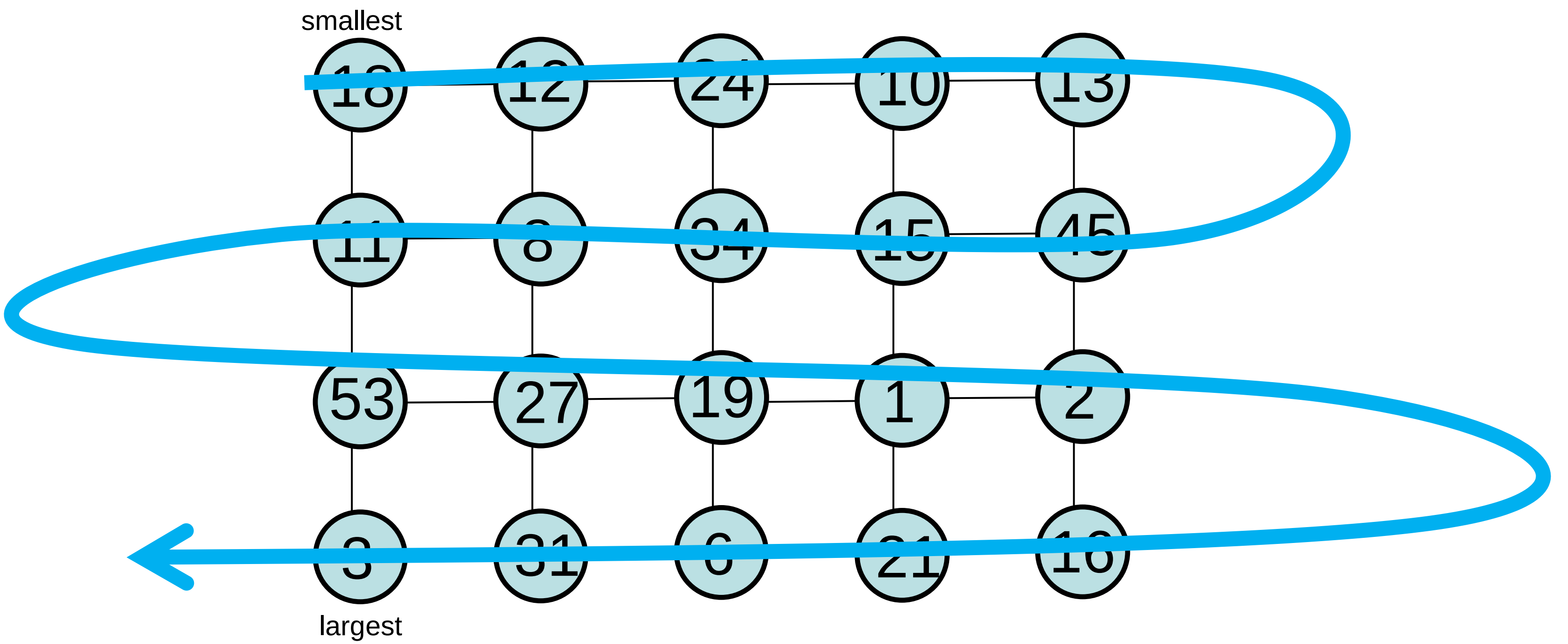
Mesh

How to sort in a mesh (aka grid)?



Mesh

How to sort in a mesh (aka grid)?



Shearsort

For mxm grid with n nodes, assume m even

In phases (of m rounds each), **Odd/Even-Sort** on columns or rows

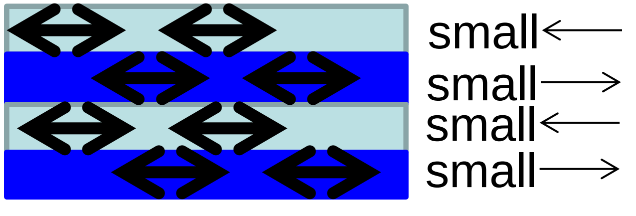
Repeat:

In odd phase: sort rows, in even phase: sort column, as follows:

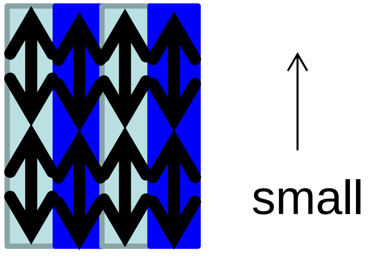
- Odd rows: sort s.t. small values move left
- Even rows: sort s.t. small values move right
- Sort column: sort s.t. small values move up

Until done

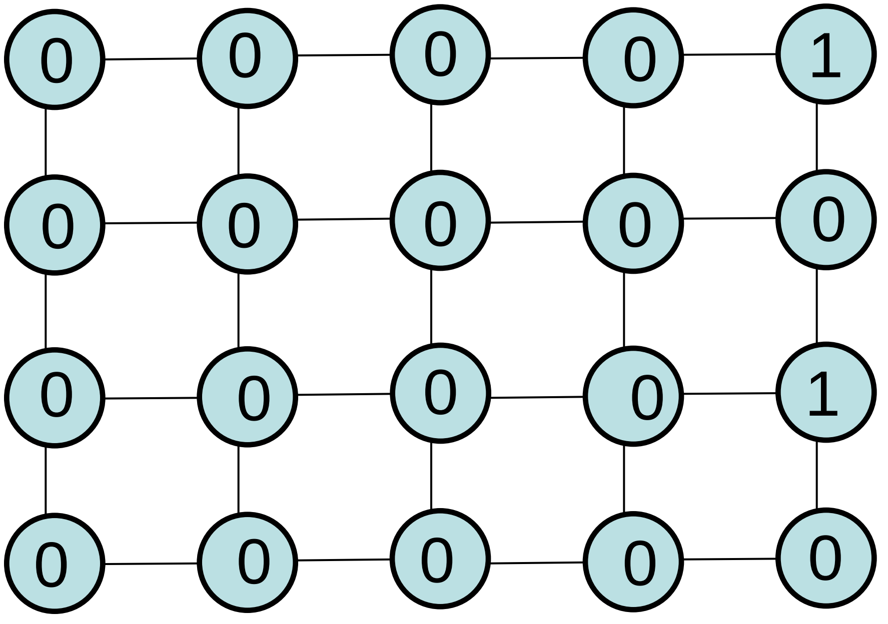
Phase 1



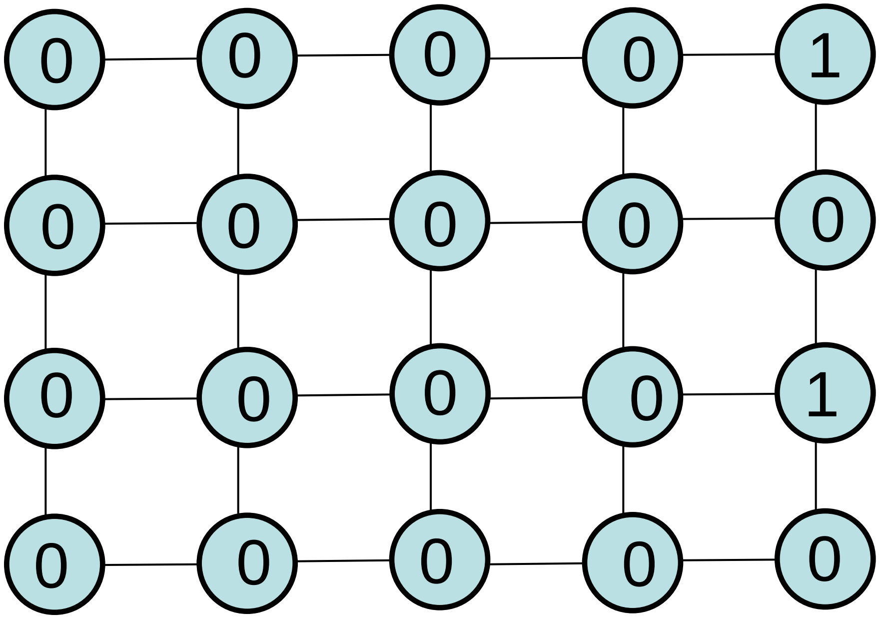
Phase 2



1. Row sort

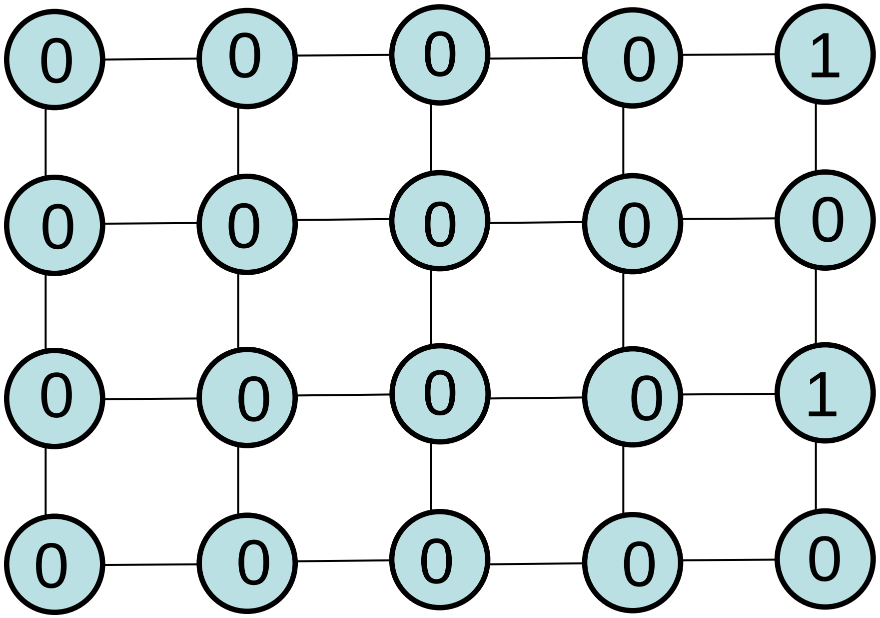


1. Row sort: nothing to do!

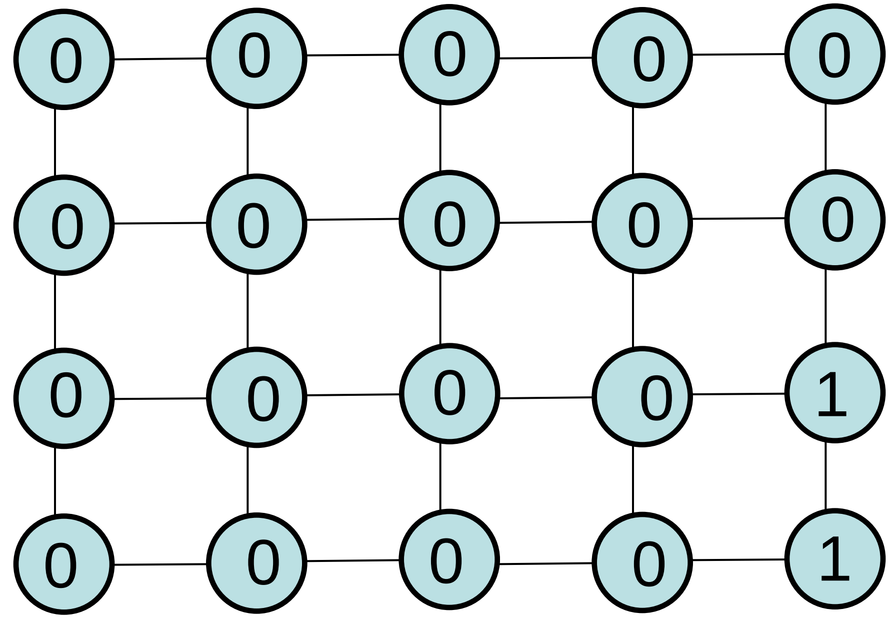


2. Column sort

1. Row sort: nothing to do!

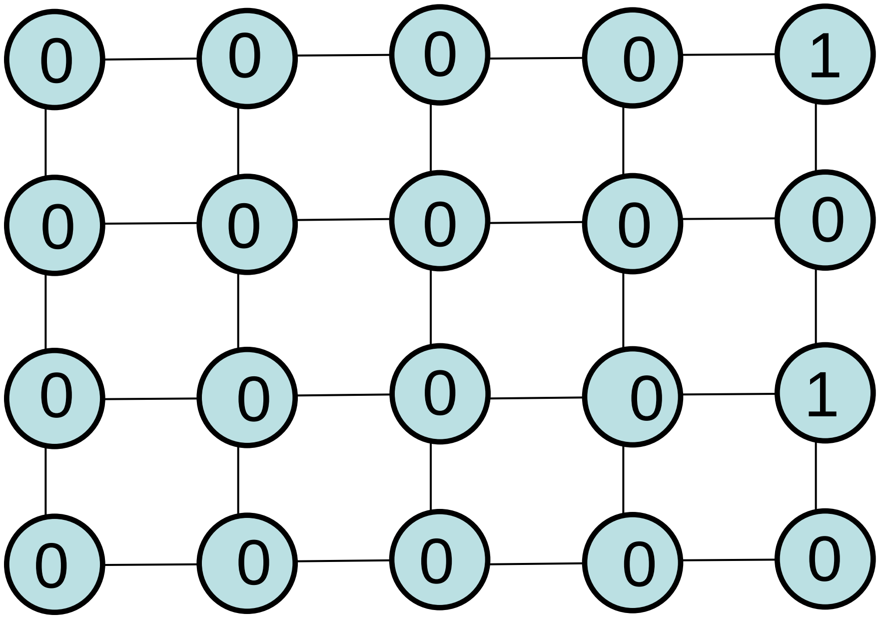


2. Column sort

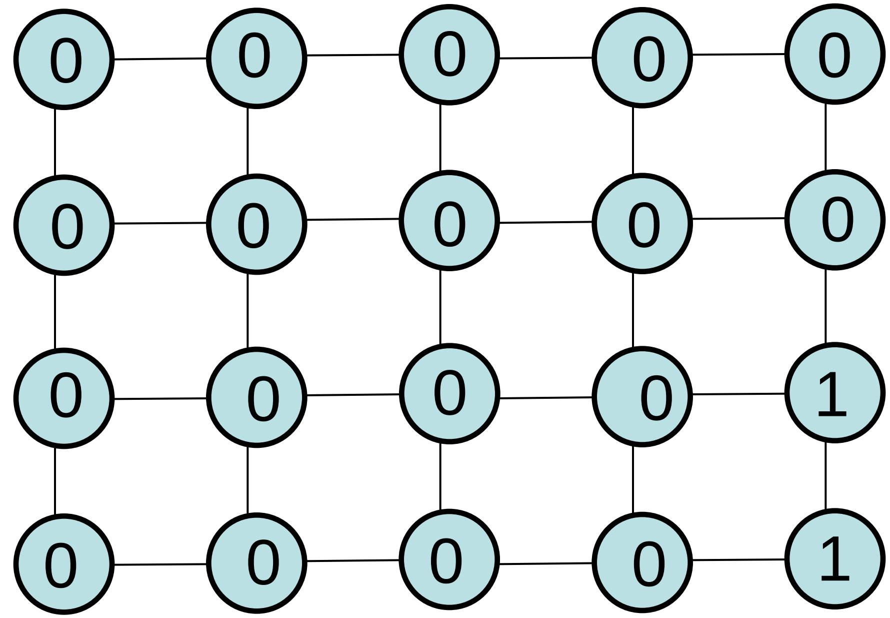


3. Row sort

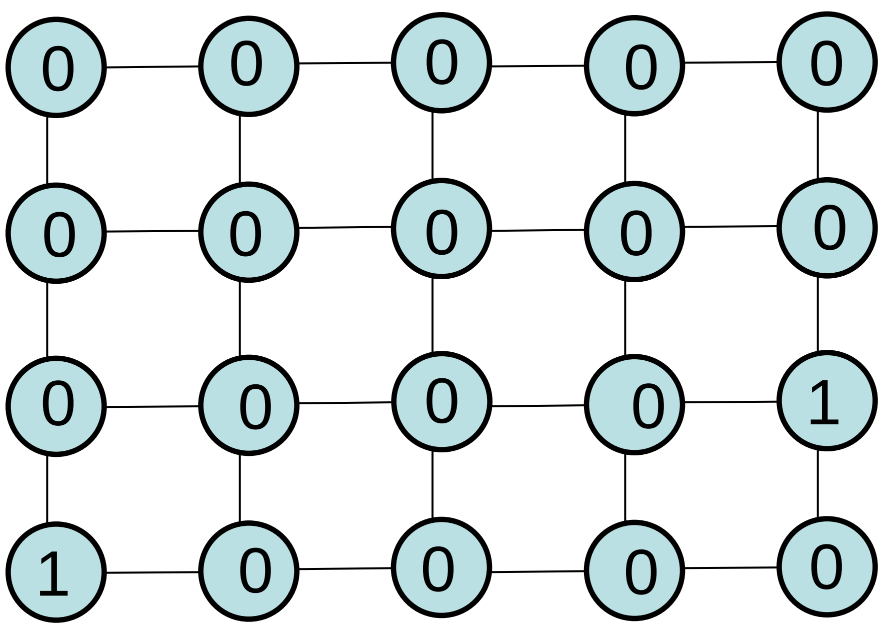
1. Row sort: nothing to do!



2. Column sort



3. Row sort



Shearsort

Shearsort

For mxm grid with n nodes, assume m even

In phases (of m rounds each), **Odd/Even-Sort** on columns or rows

Repeat:

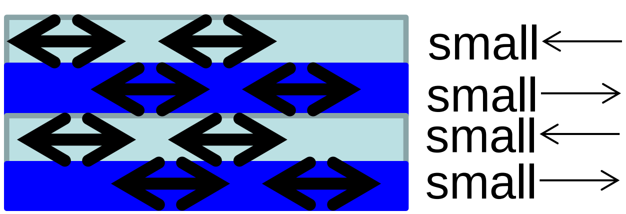
In odd phase: sort rows, in even phase: sort column,

as follows:

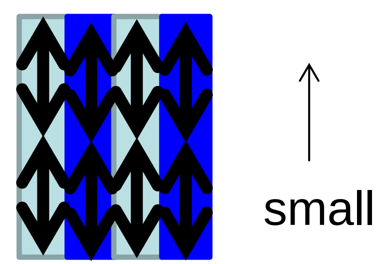
- Odd rows: sort s.t. small values move left
- Even rows: sort s.t. small values move right
- Sort column: sort s.t. small values move up

Until done

Phase 1

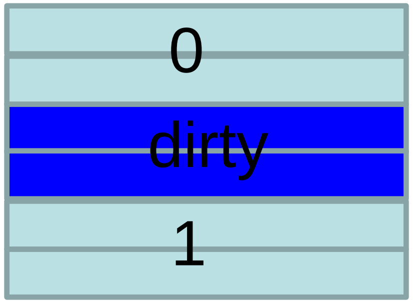


Phase 2



Runtime?

Analysis

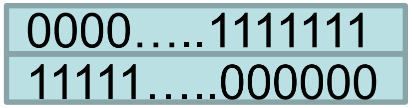


Shearsort

Sorts in time $\sqrt{n}(\log n+1)$.

Proof: Can focus on 01-inputs only!

- Idea: After a row and a column phase, half of previously unsorted rows will be sorted. So log n many phases until all are sorted, and one row/column takes time $\sqrt{n}$.
- Clean row/column: only “0” or only “1”; otherwise dirty
- At any stage, rows fall in three regions: north = clean-0, south = clean-1, middle dirty
- Initially maybe all dirty! And Shearsort does not touch clean rows.
- Consider two consecutive dirty rows, so they look as follows:



- Pair can be in three states: (A) more 0 than 1, (B) more 1 than 0, (C) same
- If (A) or (B), column sorting will give us at least one clean row, (C) gives two
- Clean row will move up or down (column sorter), and left with half the dirty rows!
- Last single row will be sorted in end.

■

Comments

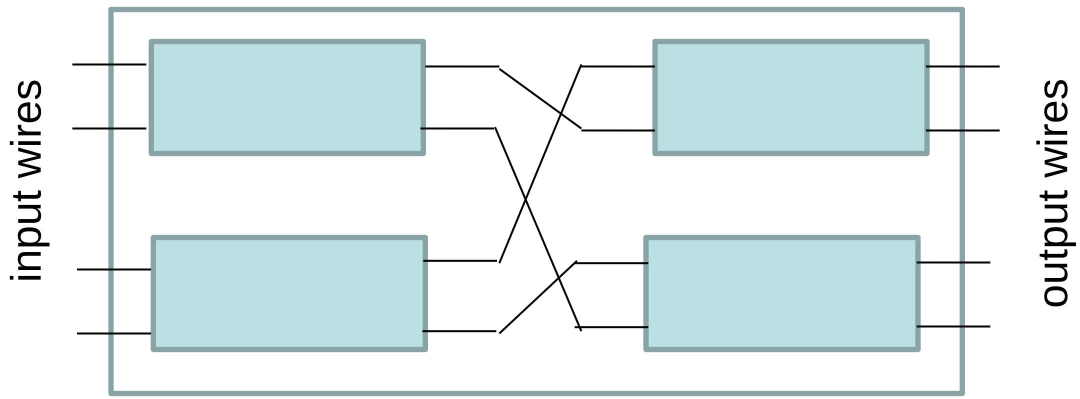
- O(m) algorithms exist, which is optimal on grid
- Anyhow $\sqrt{n}$ is nice, faster than classic sorting!
- But Heapsort & Co. have O(n log n), so maybe we can achieve even O(log n) in n-node parallel network?

Sorting Networks

Comparator

x'=min(x,y), y'=max(x,y)

Comparator Network



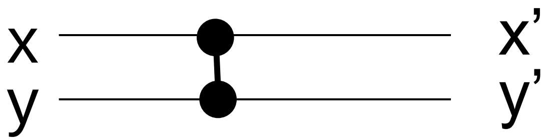
n input values sorted at output

Width

Number of wires in network.

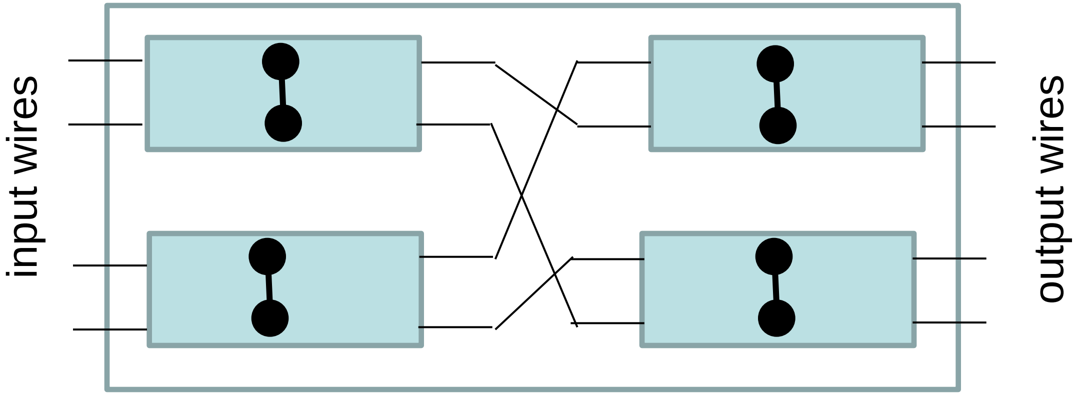
Sorting Networks

Comparator



$x' = \min(x,y)$, $y' = \max(x,y)$

Comparator Network



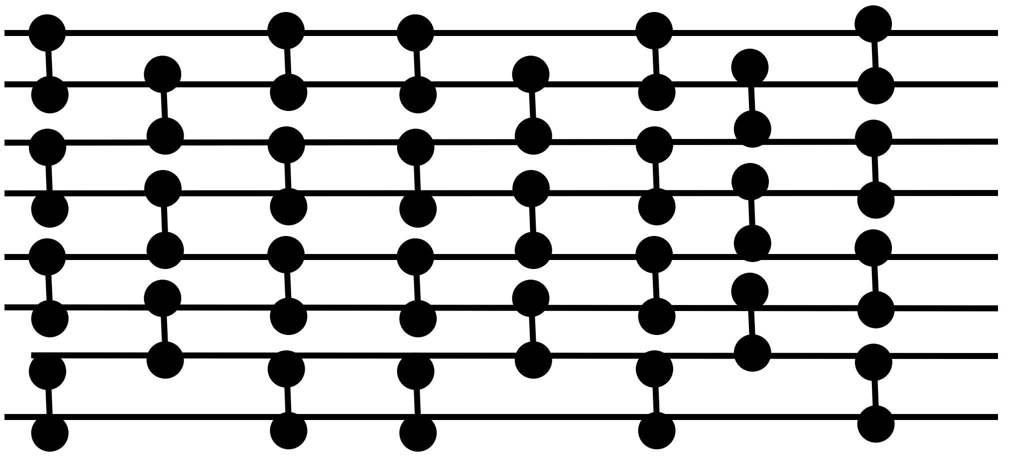
n input values sorted at output

Width

Number of wires in network.

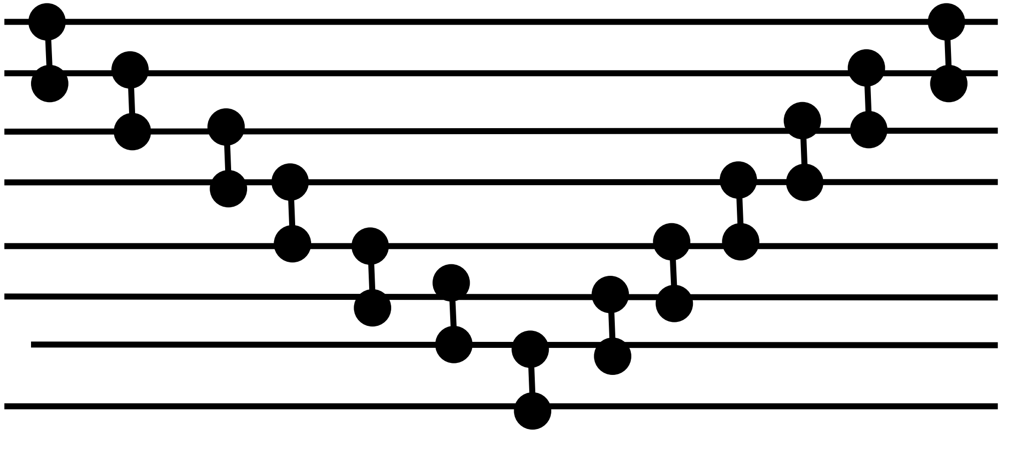
Idea how to build sorting network from comparator?

Idea how to build sorting network from comparator?

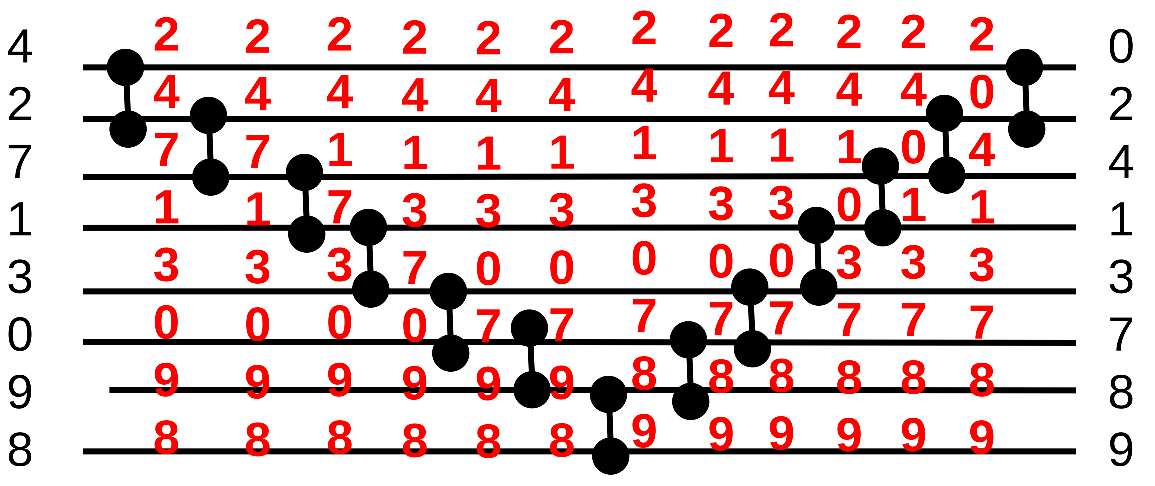


Idea 1: Odd/Even Sort

Idea 2:



Idea how to build sorting network from comparator?



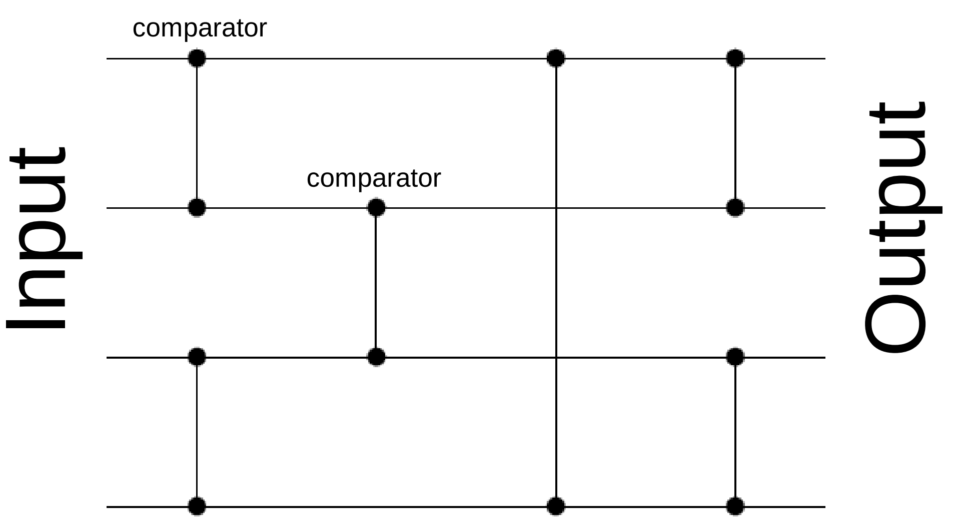
Hmm… ☺

Definitions

Sorting network is oblivious, so 01-Lemma applies

Depth

depth(input wire) = 0
depth(comparator) = max of its input wires + 1
depth(output wire) = depth of comparator
depth(comparison network) = max depth (of wires)



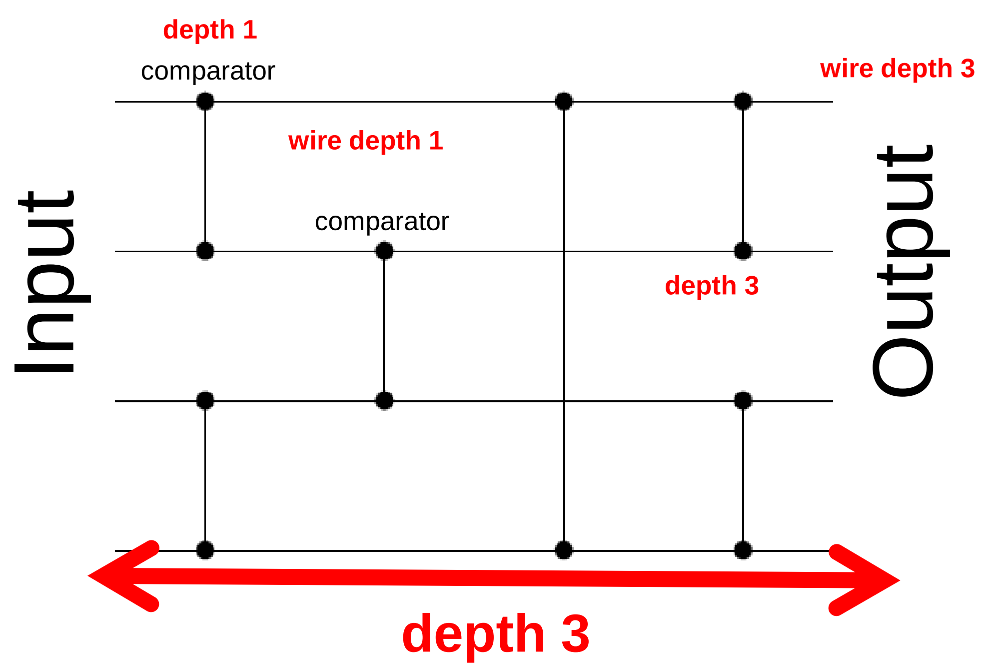
Depths?

Definitions

Sorting network is oblivious, so 01-Lemma applies

Depth

depth(input wire) = 0
depth(comparator) = max of its input wires + 1
depth(output wire) = depth of comparator
depth(comparison network) = max depth (of wires)

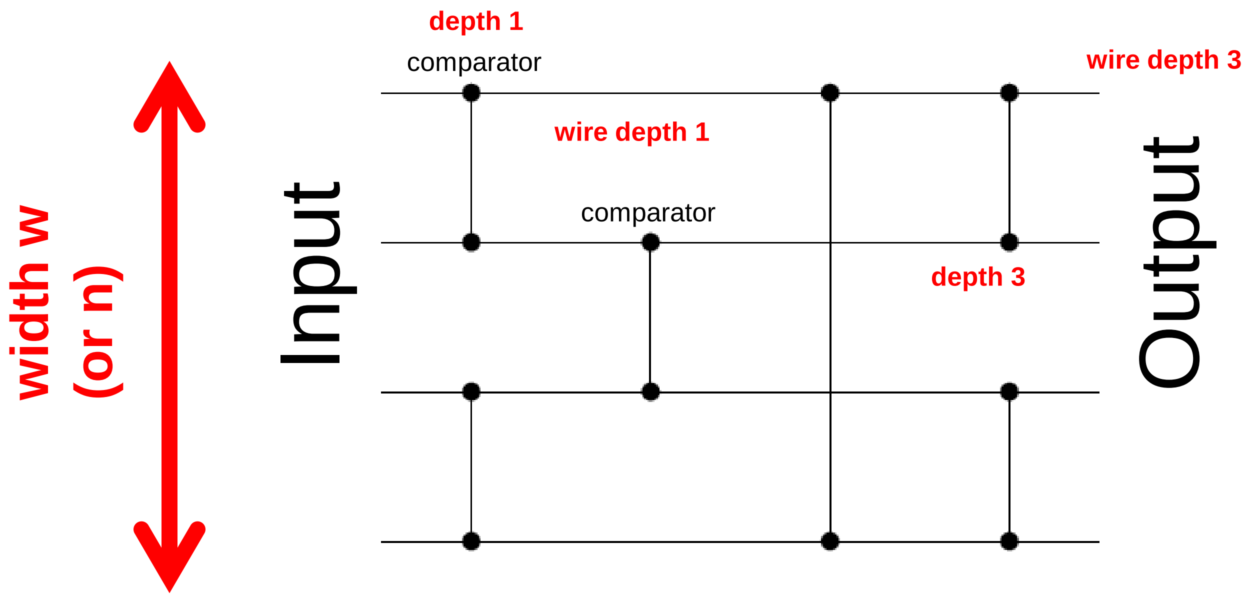


Definitions

Sorting network is oblivious, so 01-Lemma applies

Depth

depth(input wire) = 0
depth(comparator) = max of its input wires + 1
depth(output wire) = depth of comparator
depth(comparison network) = max depth (of wires)



Bitonic Sequence

Sequence of numbers which first monotonically increase, then monotonically decrease; or vice versa.

Bitonic sequence?

<1,4,6,8,3,2>, <5,3,2,1,4,8>,<9,6,2,3,5,4>,<7,4,2,5,9,8>

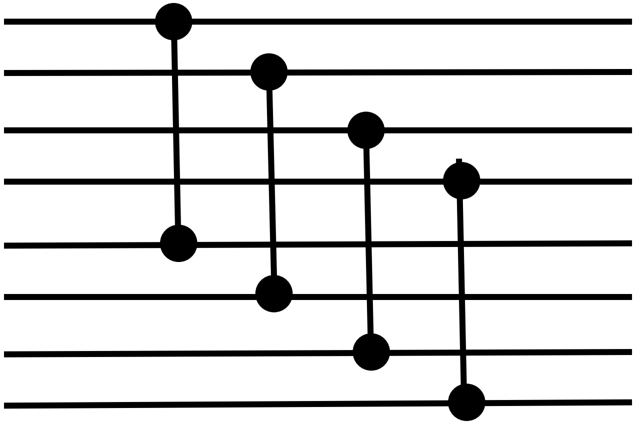
Binary bitonic sequences? $0^i1^j0^k$ or $1^i0^j1^k$

Half Cleaner

Sorting network is oblivious, so 01-Lemma applies

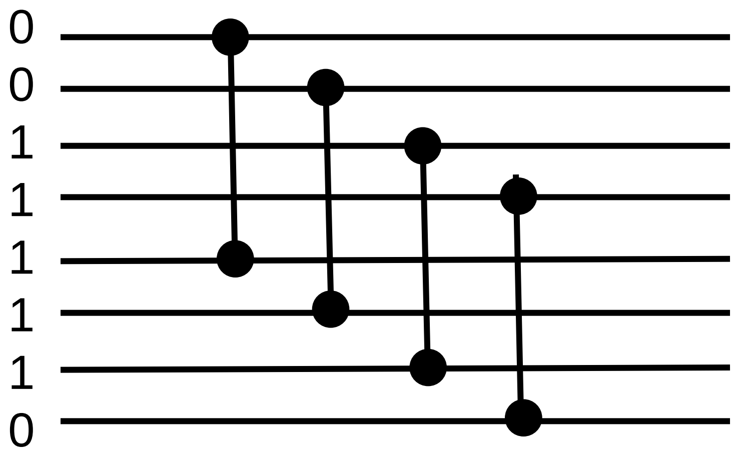
Half Cleaner (HC)

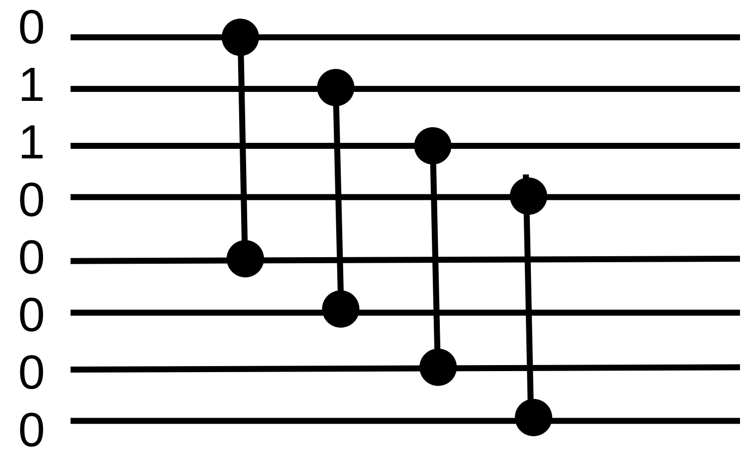
Comparison network of depth 1, where wire i is compared with wire i+n/2 (for i=1,…,n/2).

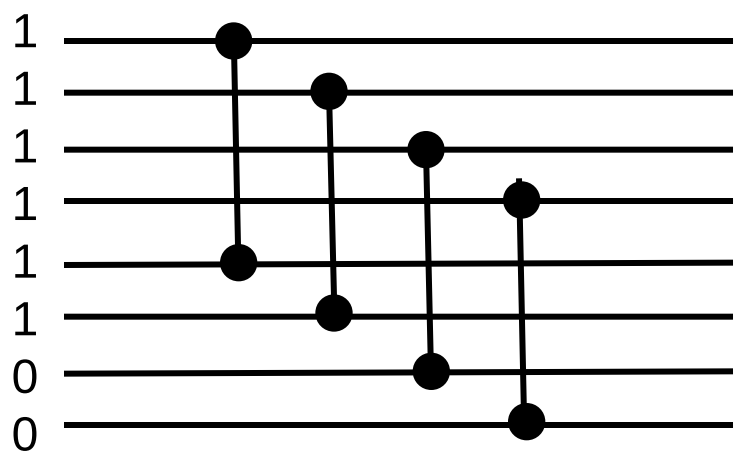


What does it do?

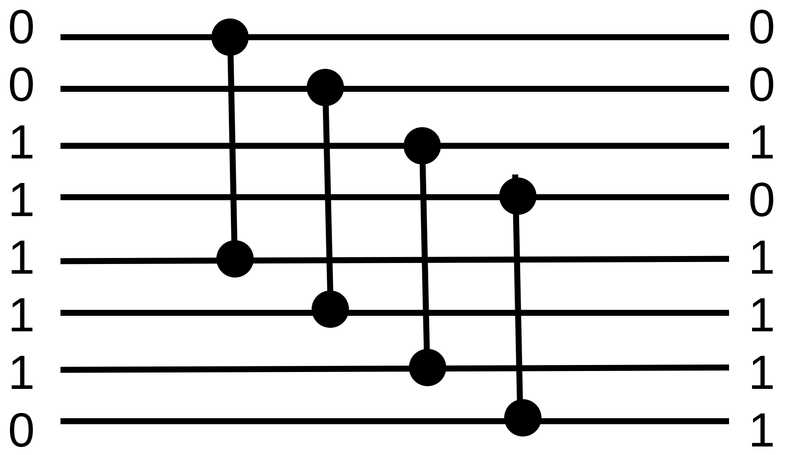
Example

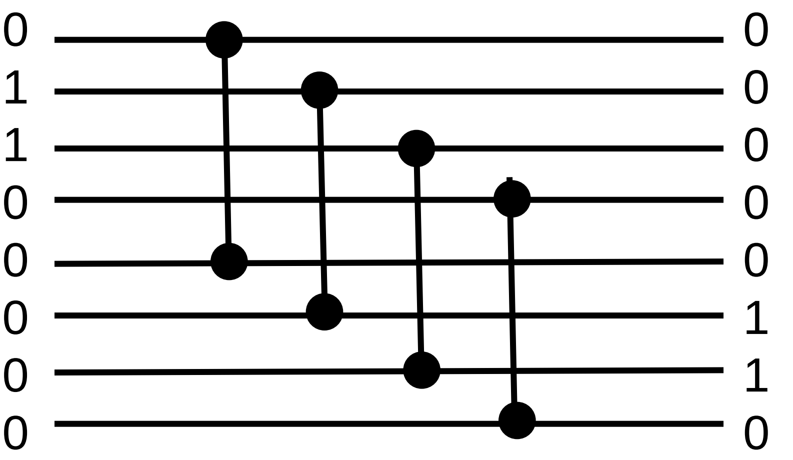


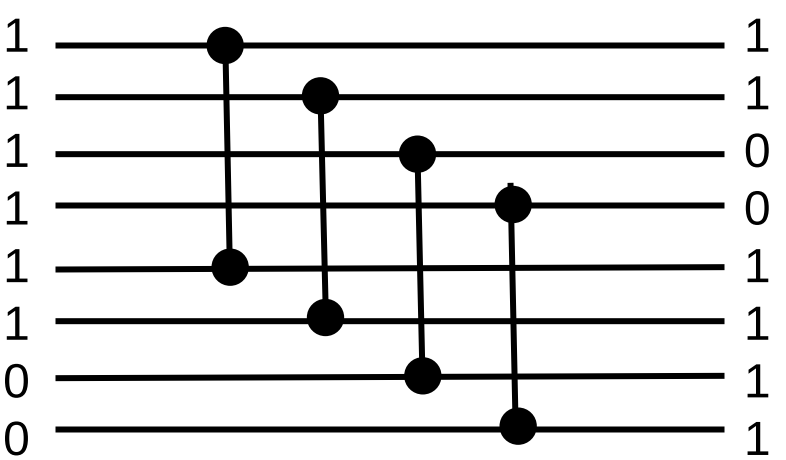




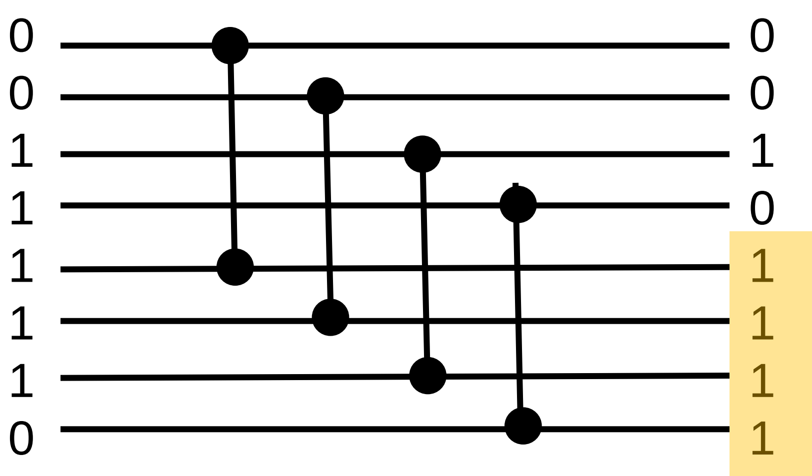
Example

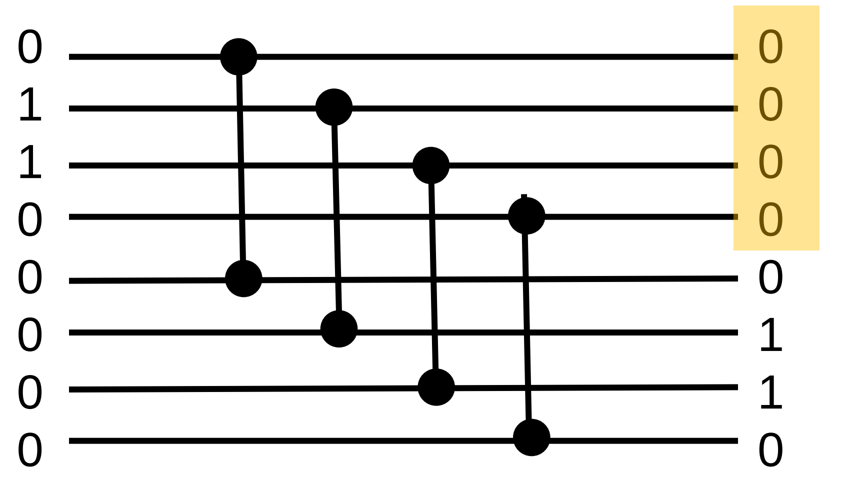


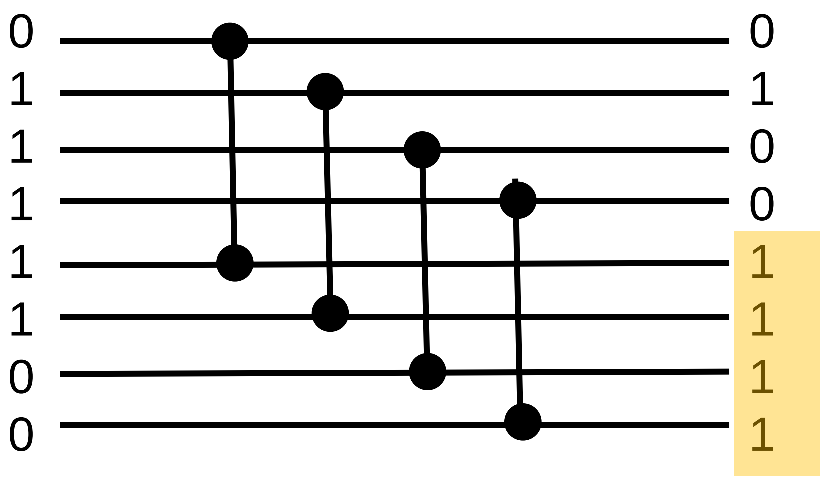




Example



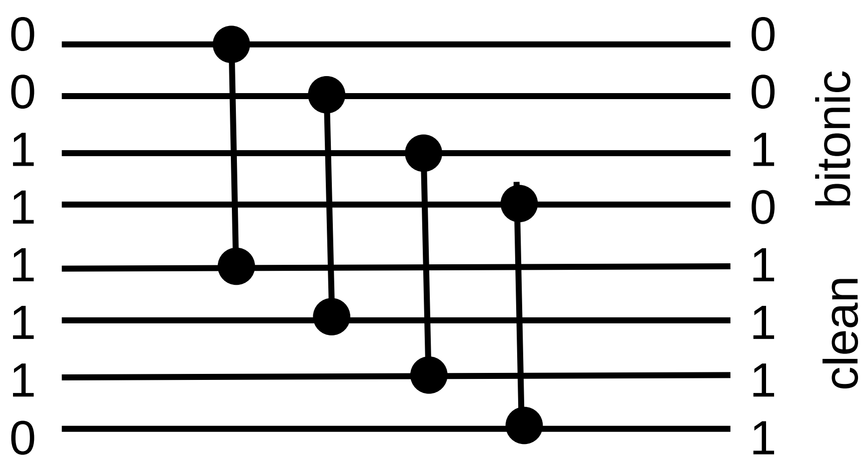




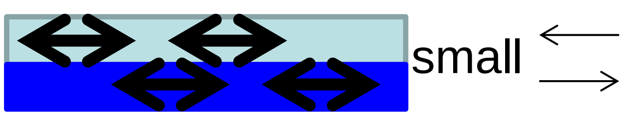
Bitonic Sorter (BS)

Given a bitonic sequence, a Half Cleaner cleans the upper or lower half of the n wires. The other half is bitonic.

Proof: Without loss of generality, assume input is $0^i 1^j 0^k$

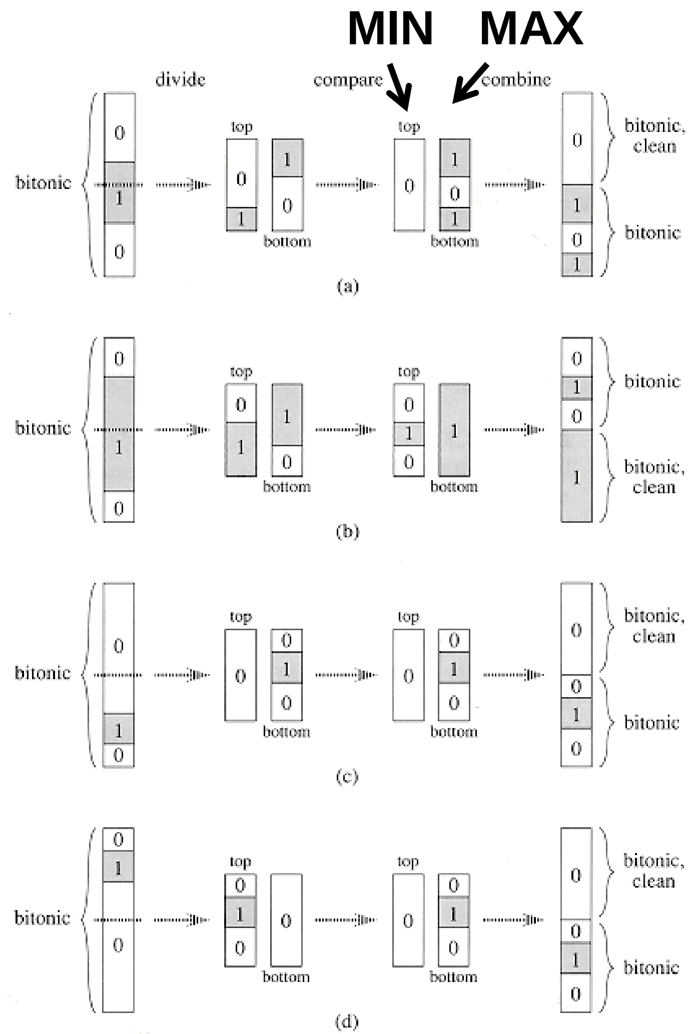


- If midpoint of bitonic sequence is in 0s, half is 0s only => will stay so
- If midpoint is in 1s, bitonic sorter is like Shearsort with two adjacent rows! See proof there.

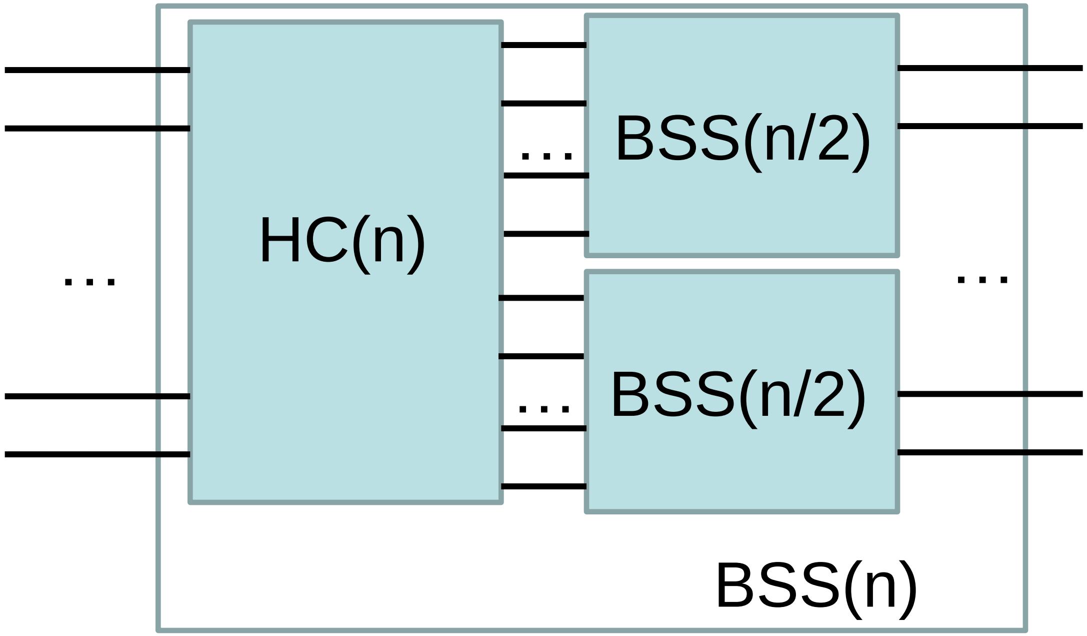


■

Proof by Case Distinction



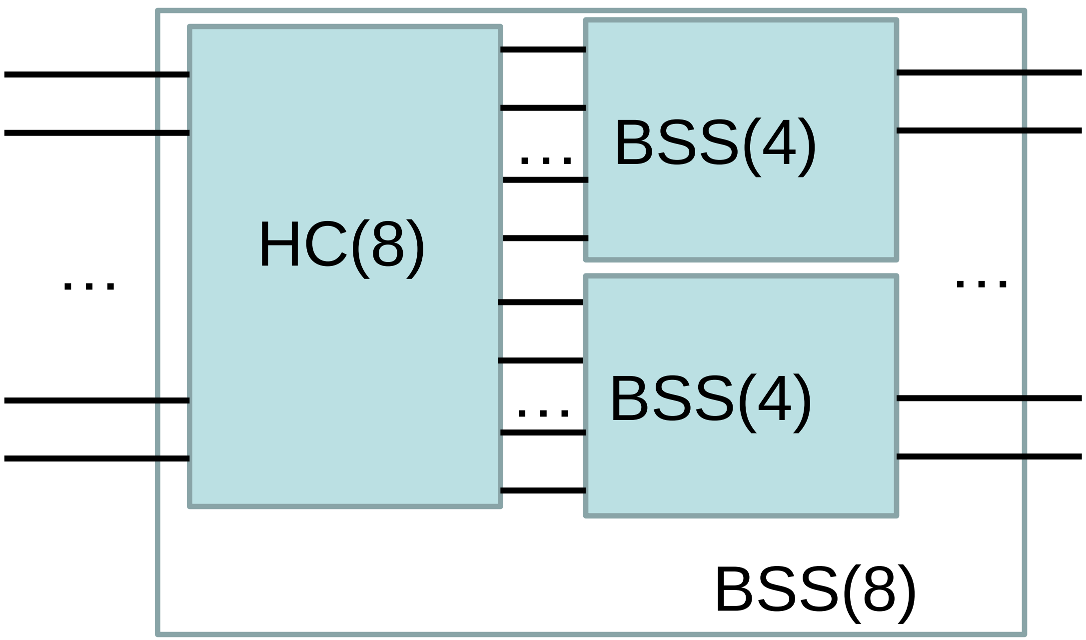
Bitonic Sequence Sorter (BSS)



BSS(n) consists of a n-port Half Sorter and 2 BSS(n/2).
BSS(1) is empty.
Recursively defined, so depth? Logarithmic!

Example: BSS(8)?

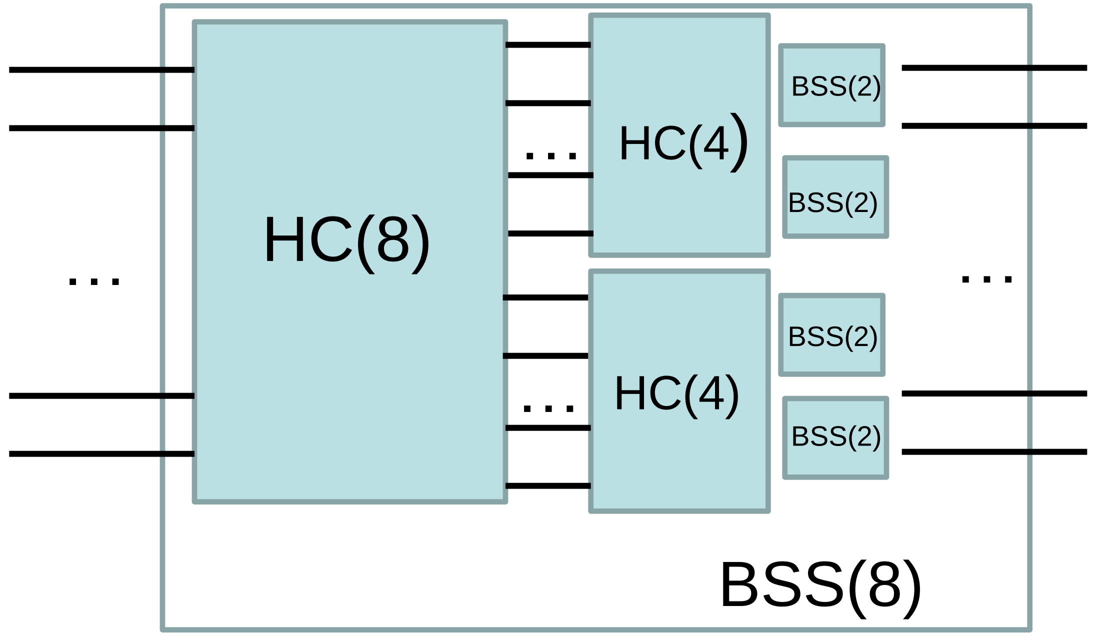
Recursion 1:



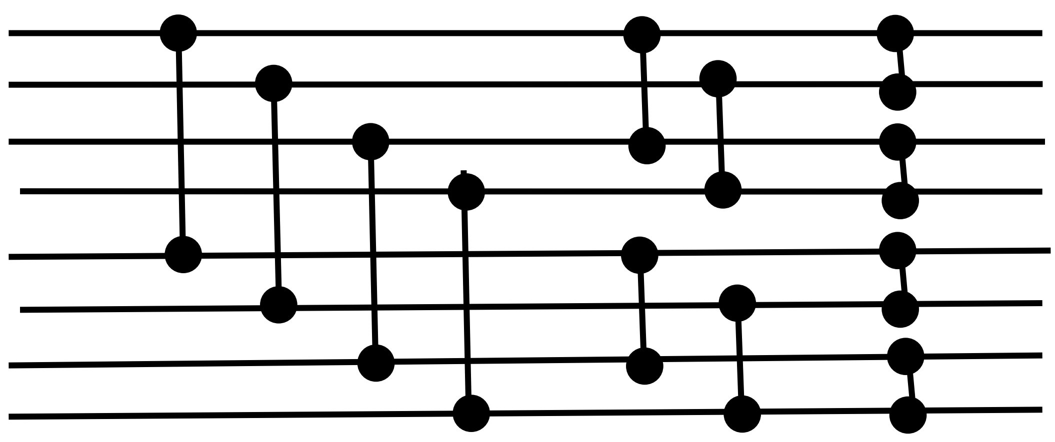
Draw BSS(8)!

Example: BSS(8)?

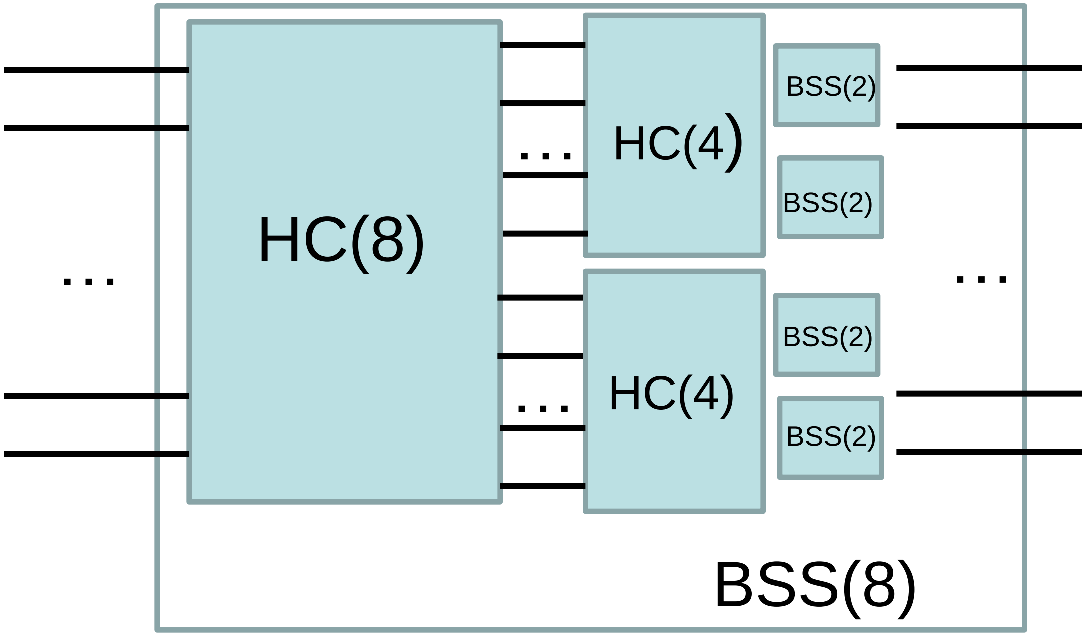
Recursion 2:

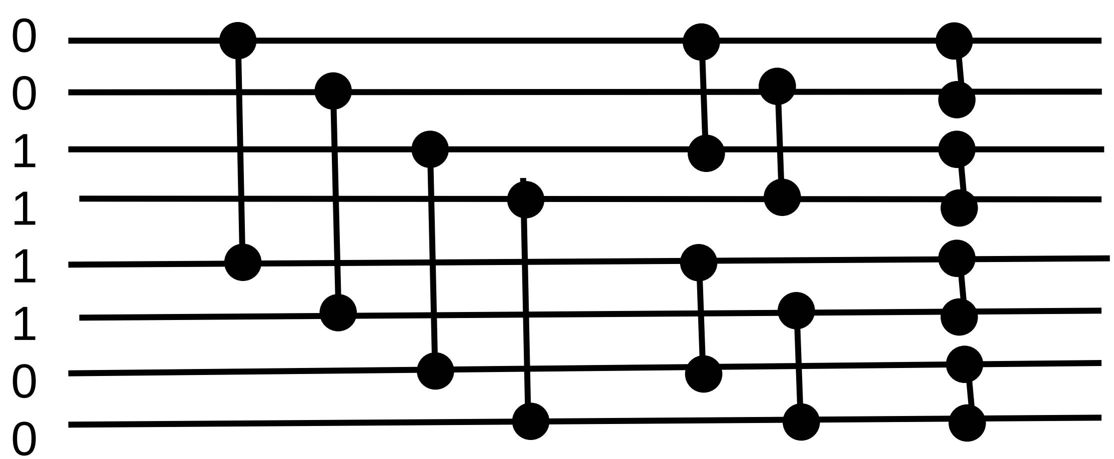


Sequence of Half-Cleaners!

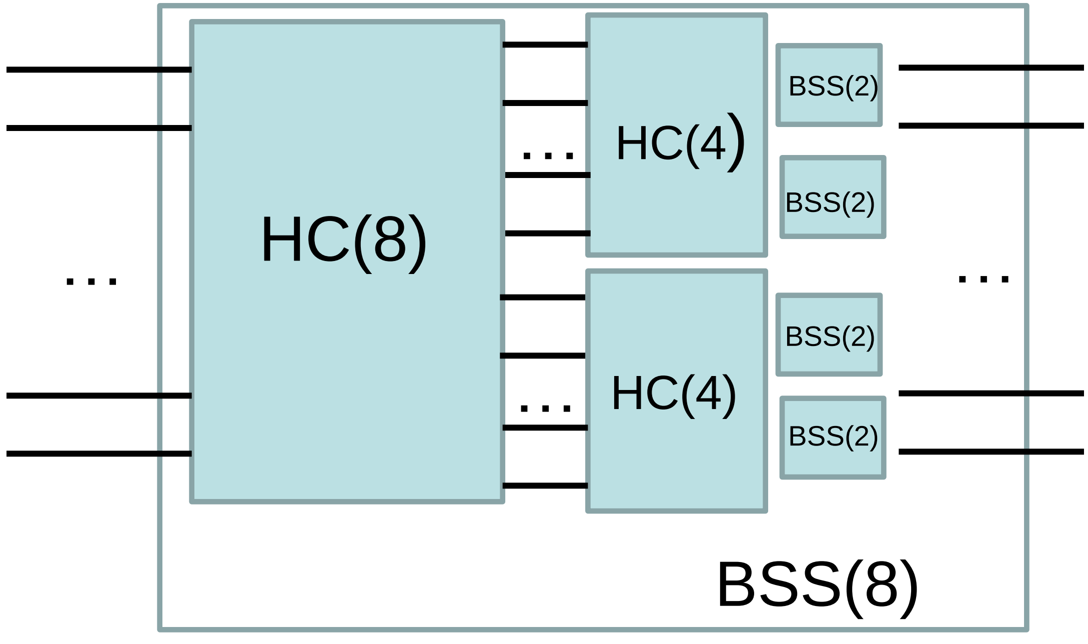


What does it do??

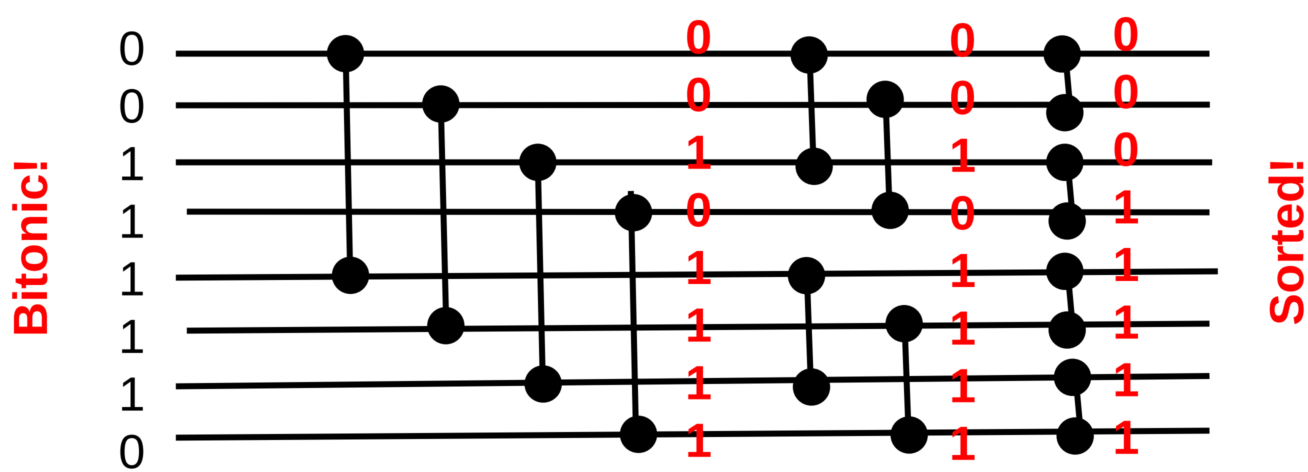




What does it do??



Why does it work?



Bitonic Sequence Sorter

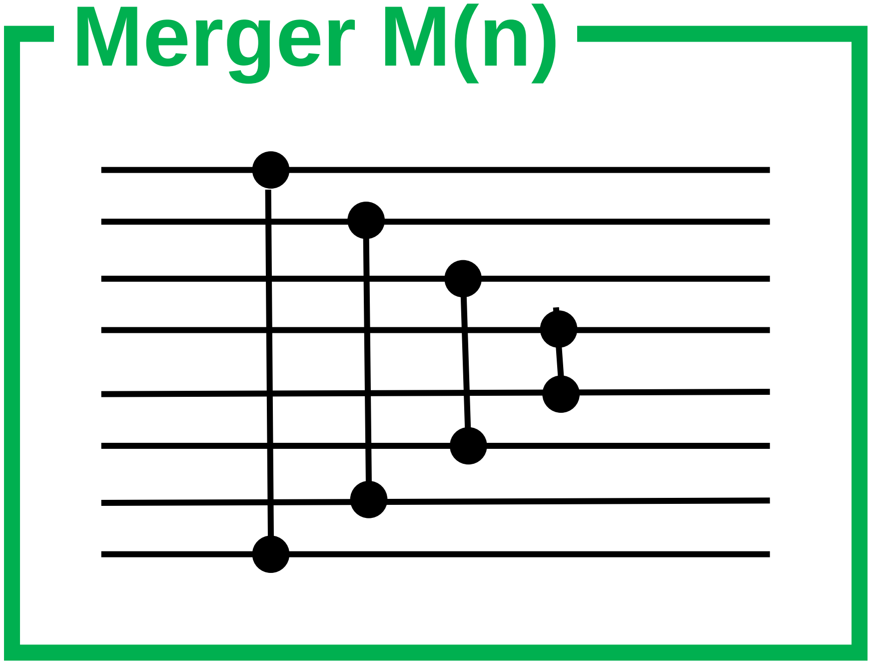
BSS(n)

BSS(n) sorts bitonic sequence in time log(n).

Proof: Follows directly from BSS(n) algorithm and property that size of bitonic half is divided in two in each step. ■

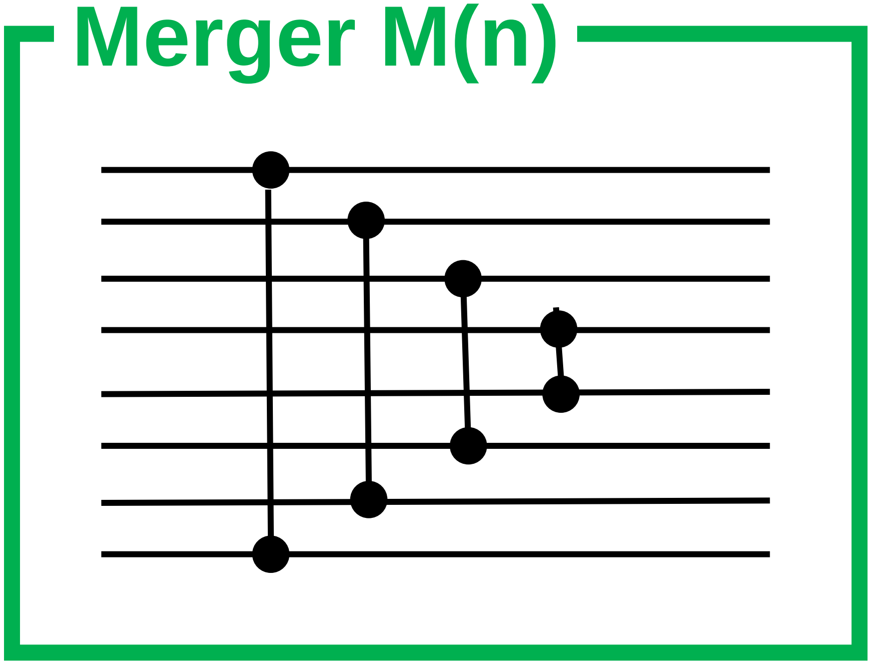
But we want to sort arbitrary sequences, not only bitonic ones! How? Need Merging Networks (MN)!

Merger



Depth-1 network where wire i is compared to n-i+1.

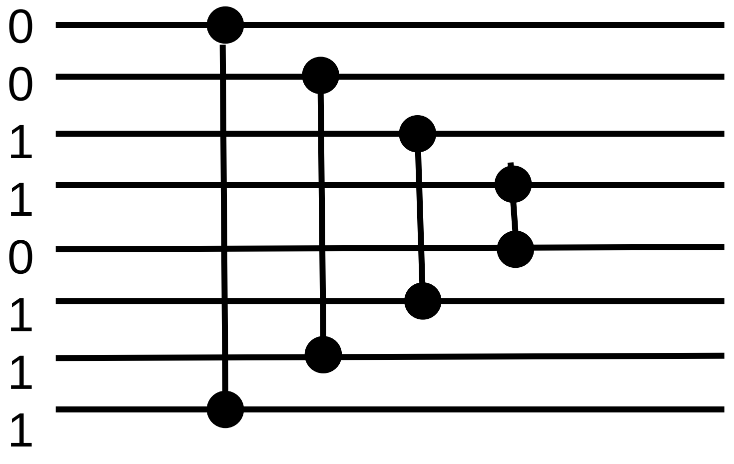
Merger

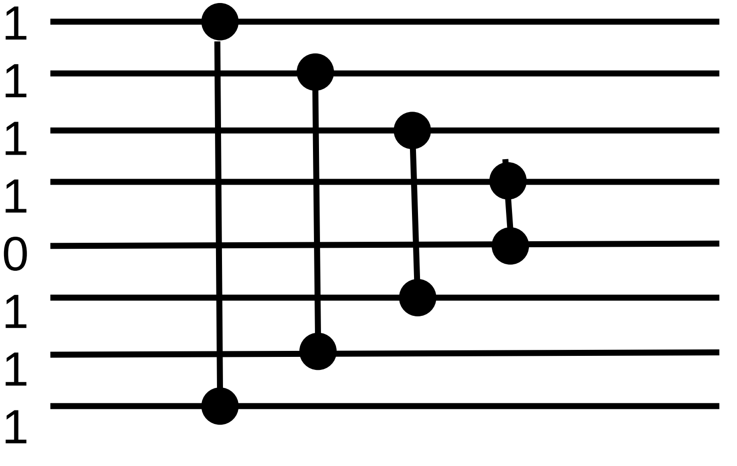


Depth-1 network where wire i is compared to n-i+1.

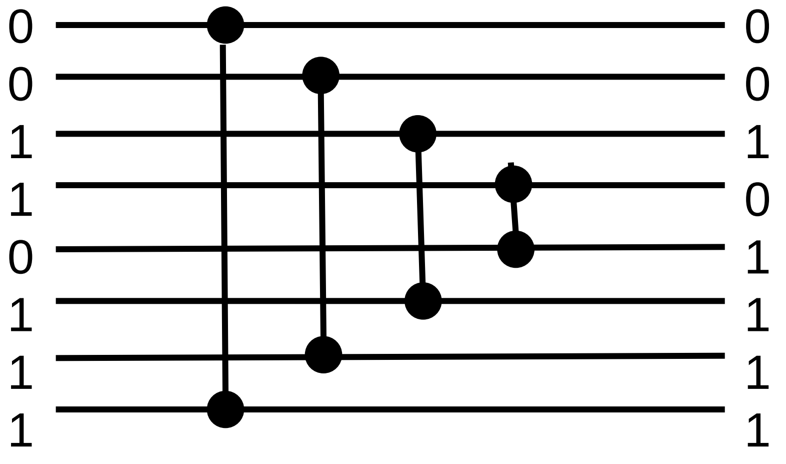
What does it do?!

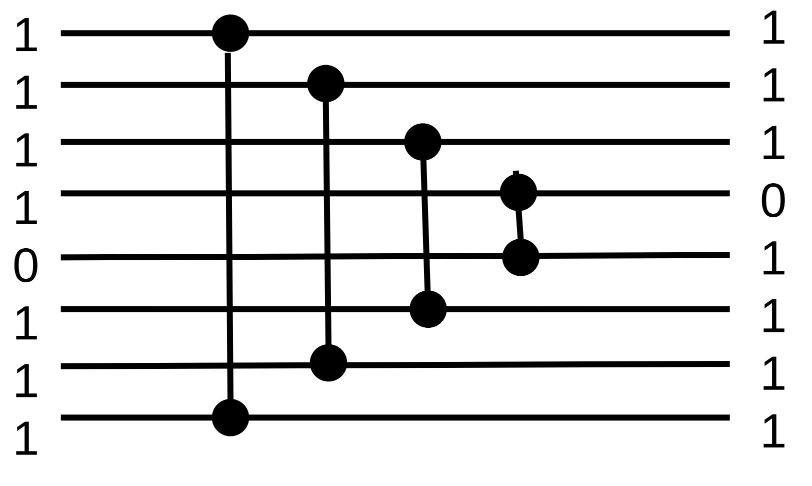
Merger



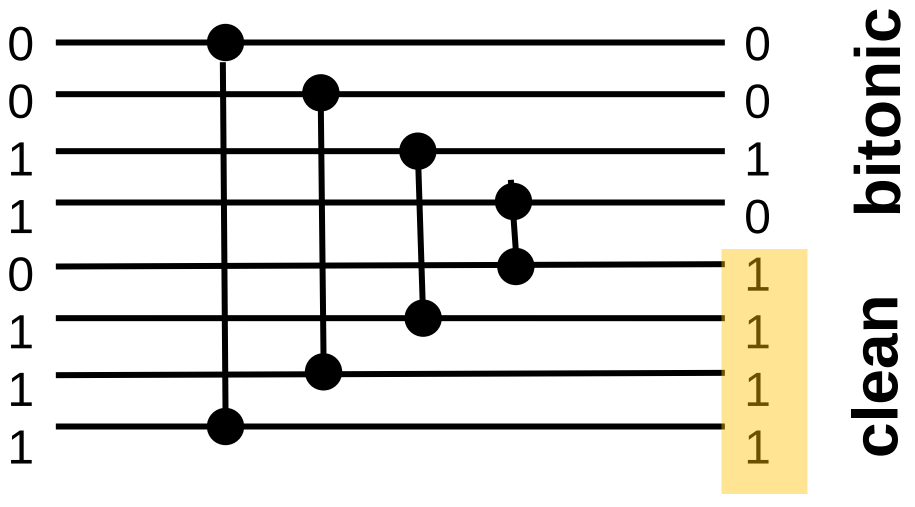


Merger

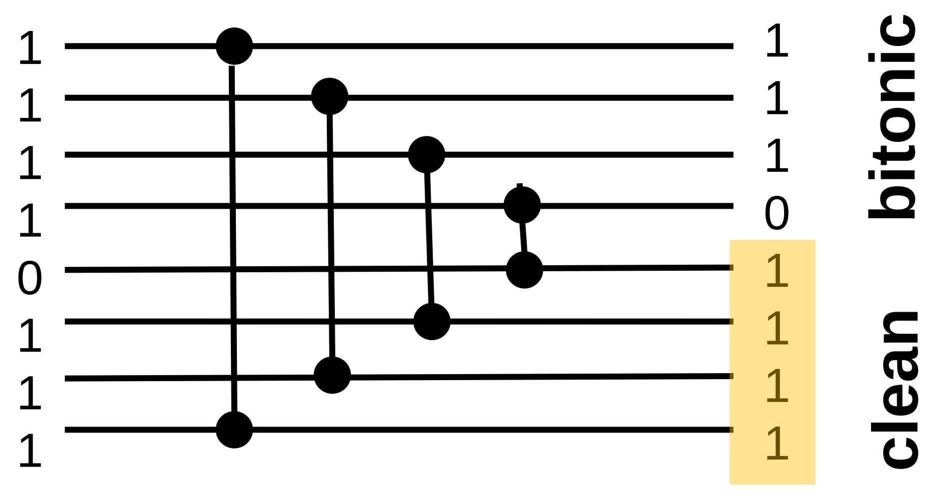




Merger



0
0
1
1
0
1
1
1
0
0
1
0
1
1
1
1
bitonic
clean

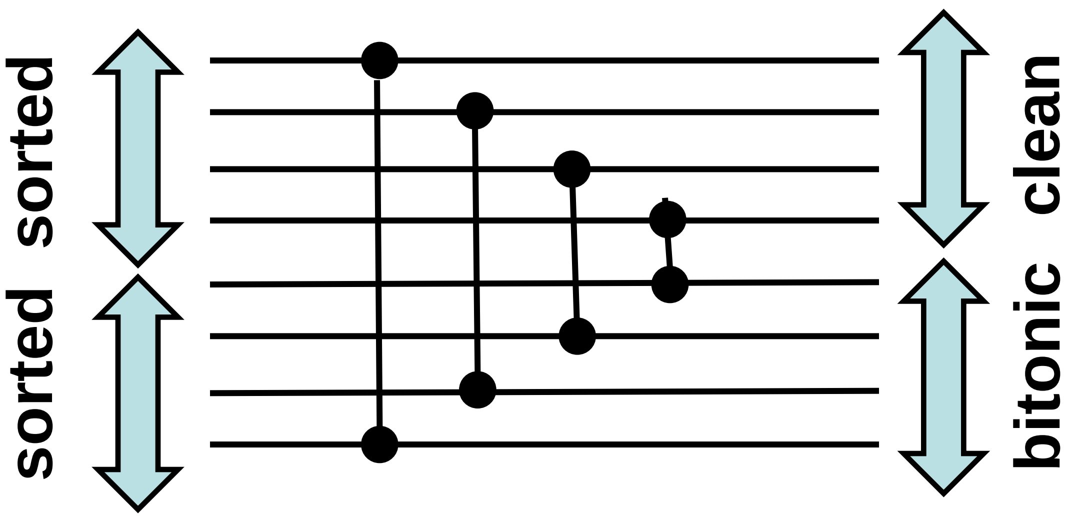


1
1
1
1
0
1
1
1
1
1
1
0
1
1
1
1
bitonic
clean

Merger

If two sorted sub-sequences are input to Merger,
then output two sub-sequences: one clean, other bitonic.

Proof: Merger for sorted parts is like Half-Cleaner for bitonic:
After the merger step, either the upper or lower half is clean, the other bitonic.

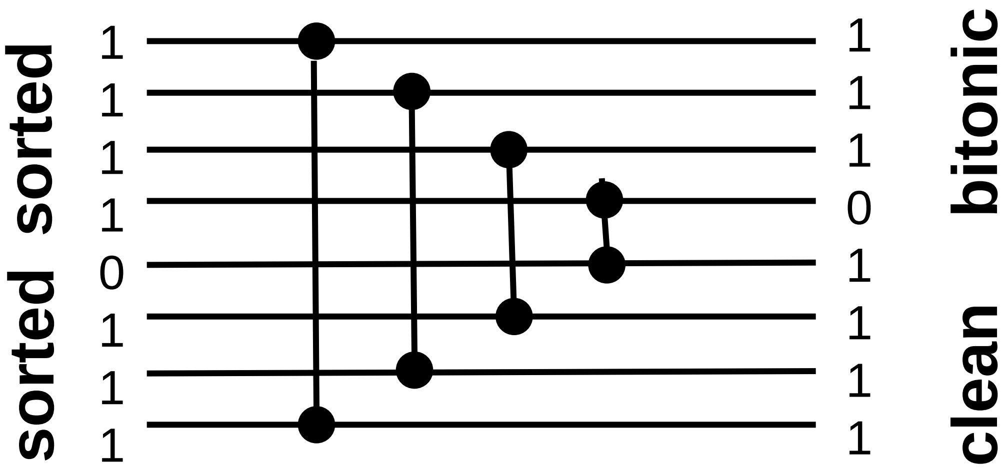


■

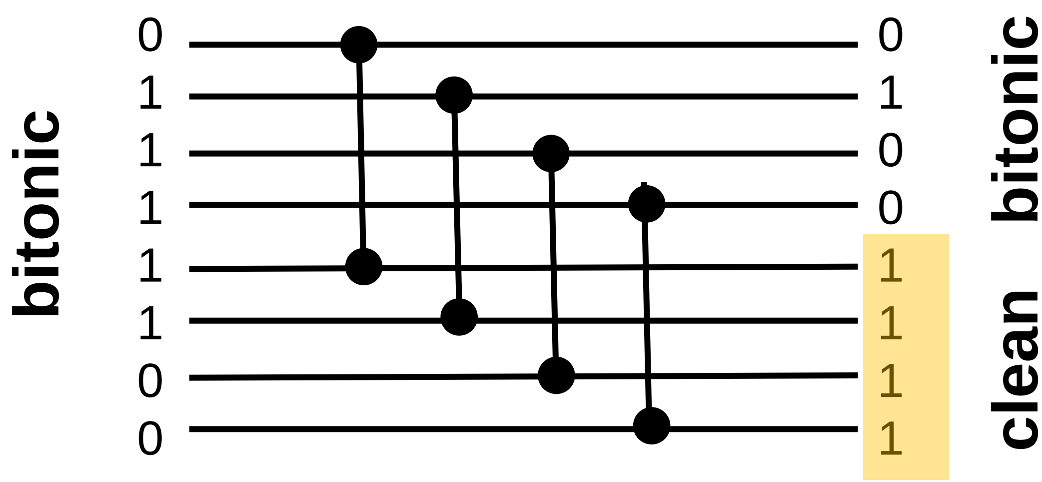
Or vice versa ☺

Perfect Output for HC

Merger:

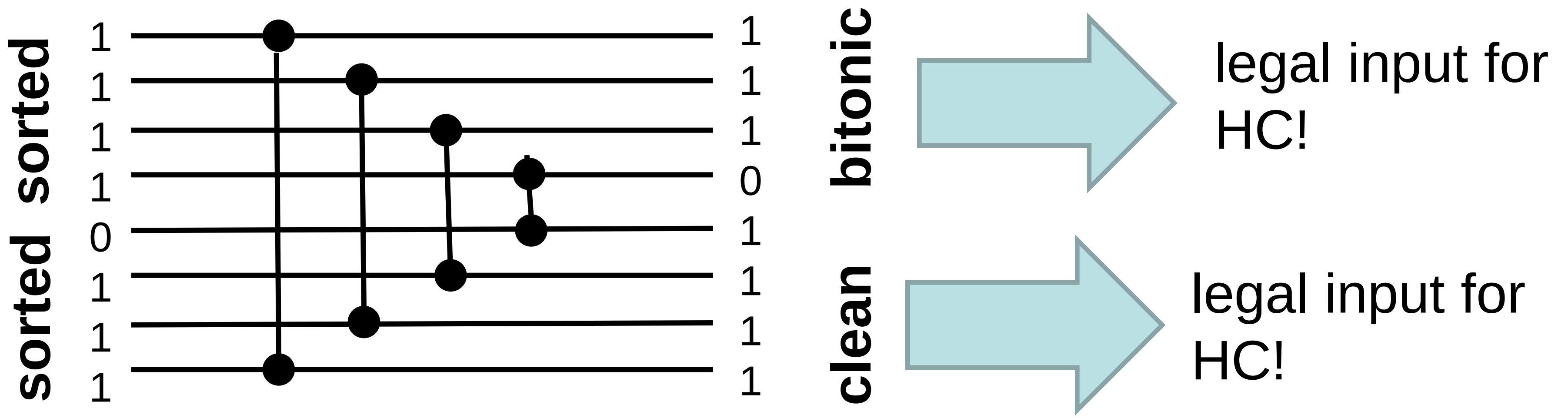


Half Cleaner:

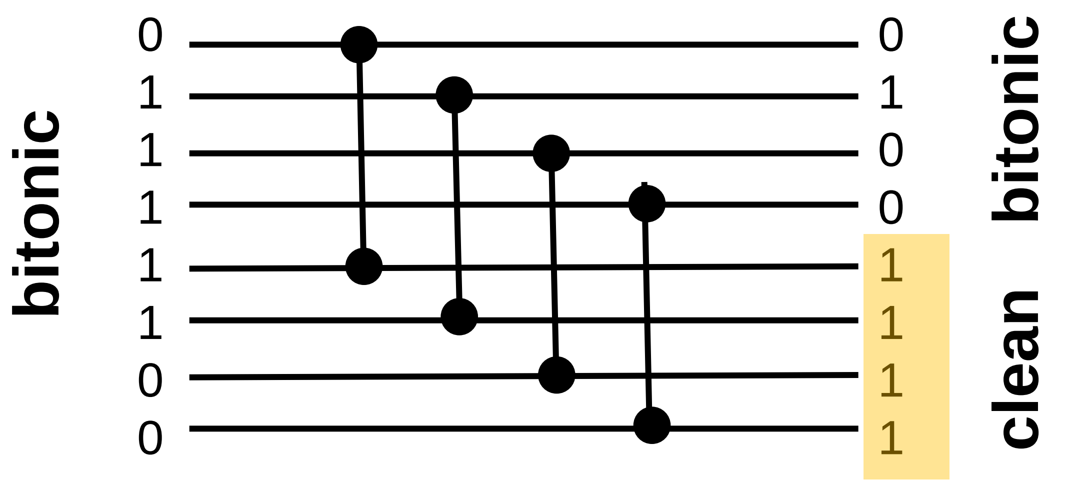


Perfect Output for HC

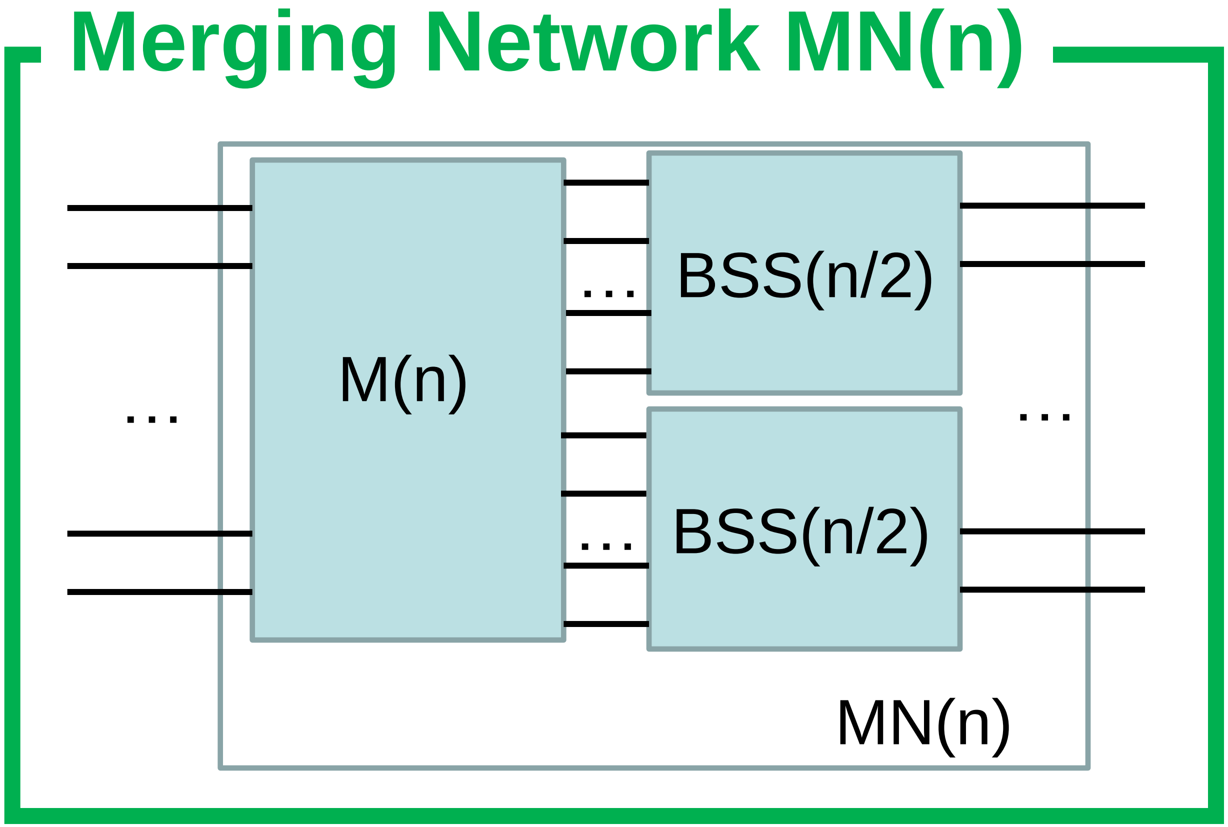
Merger:



Half Cleaner:

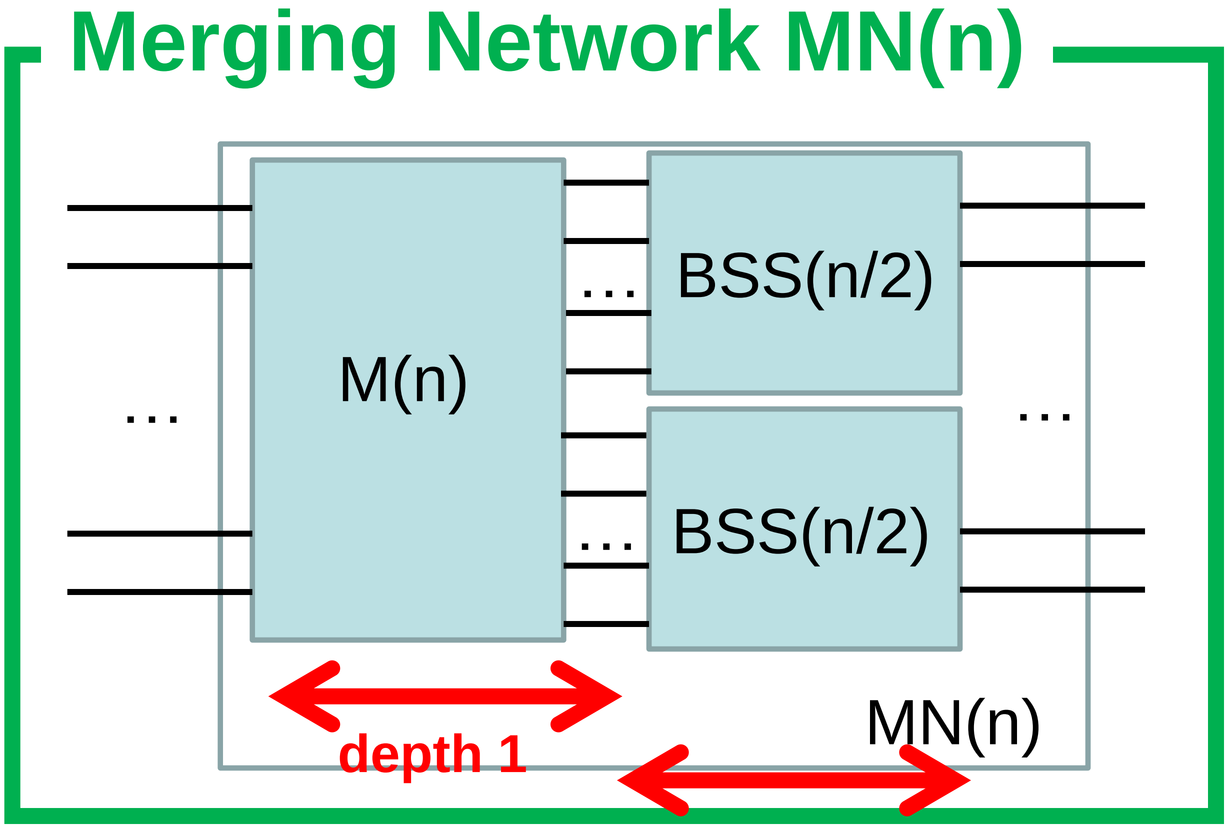


Merging Network



Merge then half-clean it!
Merger M(n) followed by two BSS(n/2).
What is depth?

Merging Network

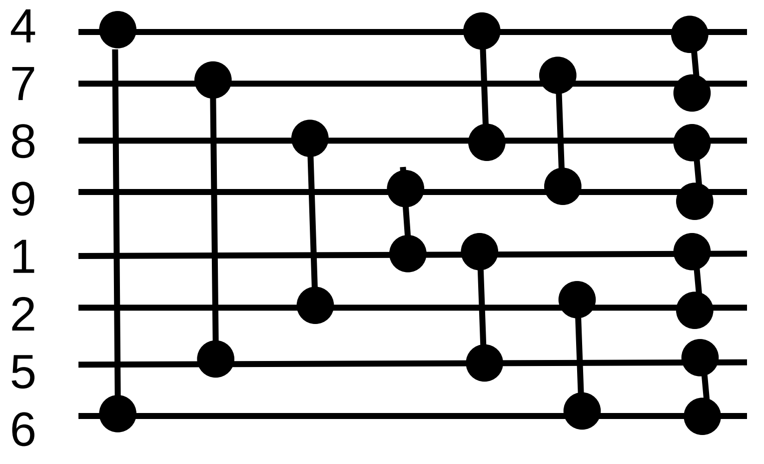


recursive so log n

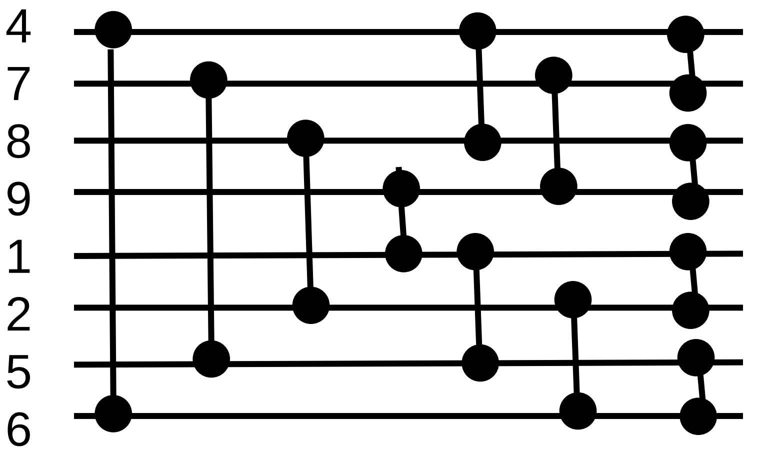
Merge then half-clean it!
Merger M(n) followed by two BSS(n/2).
What is depth?

How does MN(8) look like?

How does MN(8) look like?

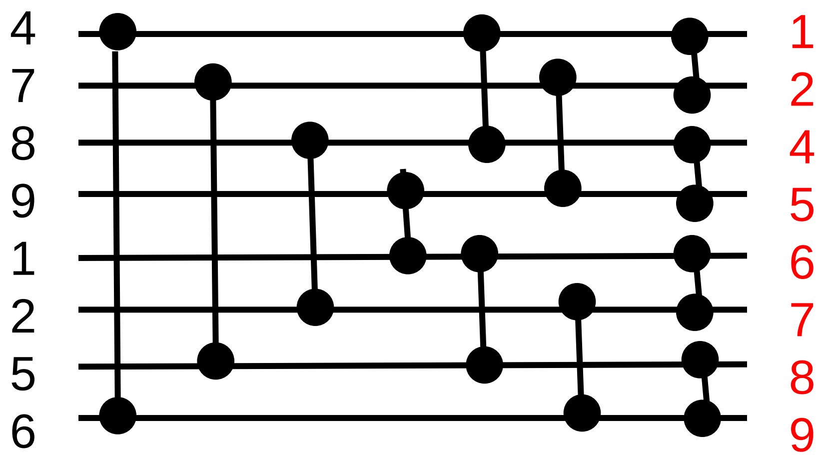


What does MN(8) do?



What does MN(8) do?

If both halfs of input sequences sorted, sorted in end!

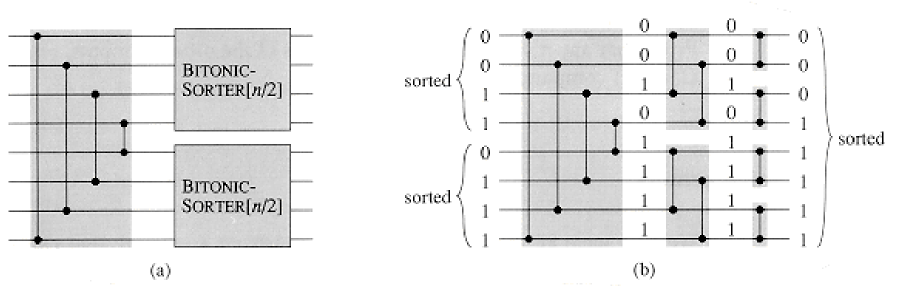


Merging Network (MN)

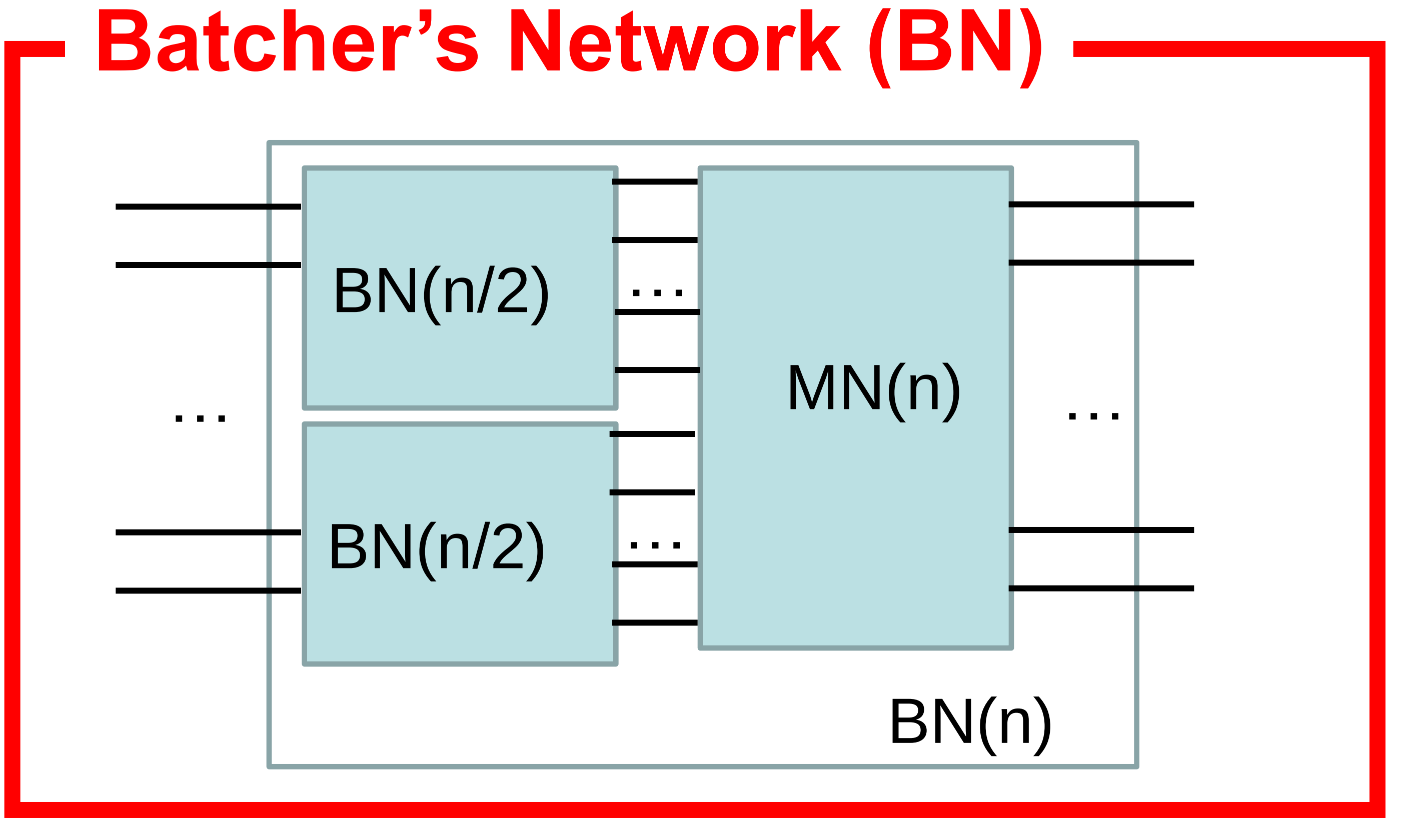
Merges two sorted input sequences of length n/2 into one sorted sequence of length n.

Proof: After the merger step, either the upper or lower half is clean, the other bitonic. BSS sequence sorters take care of complete sorting.

∎

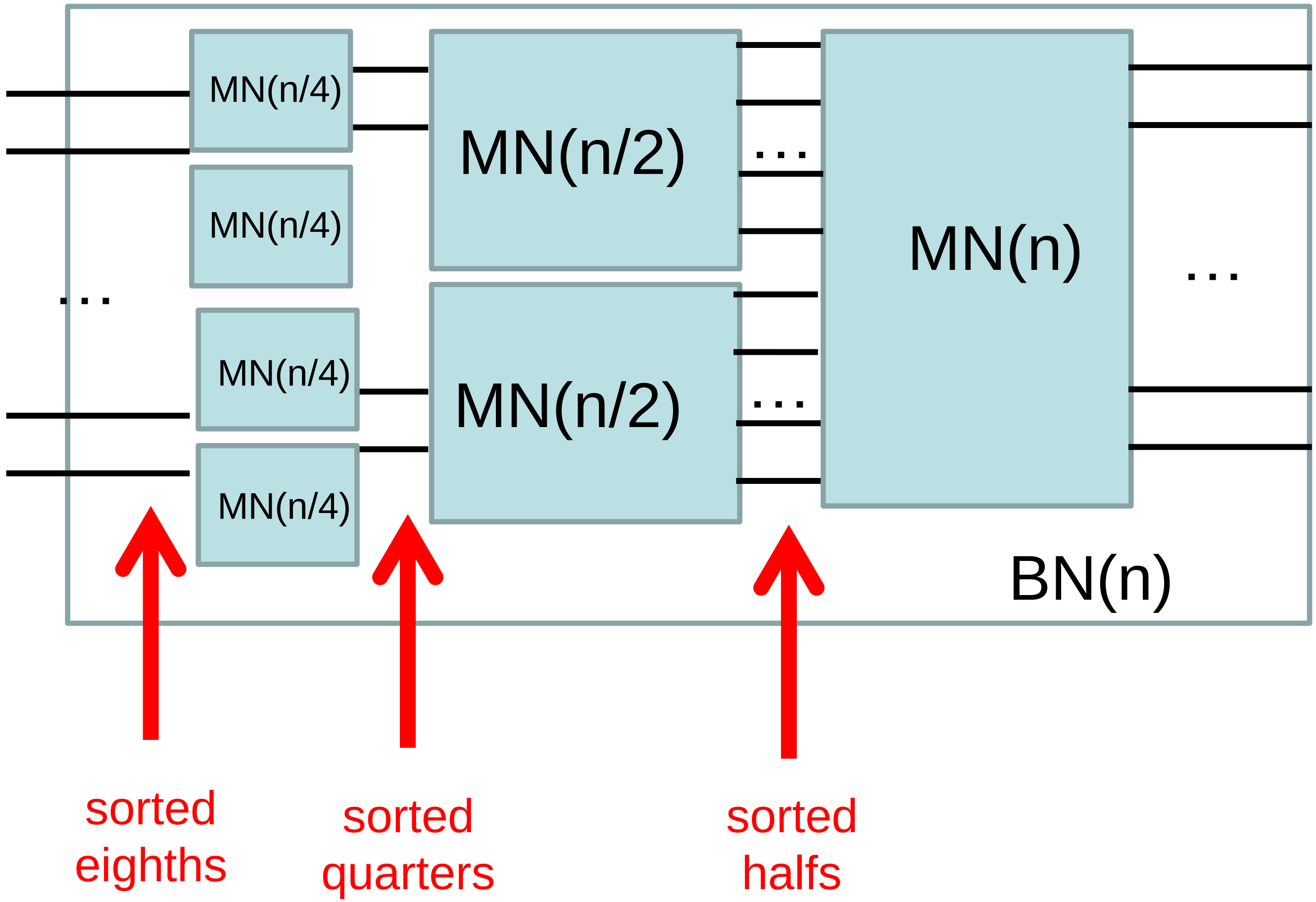


So how to sort n values? Can merge two halves: do it recursively!

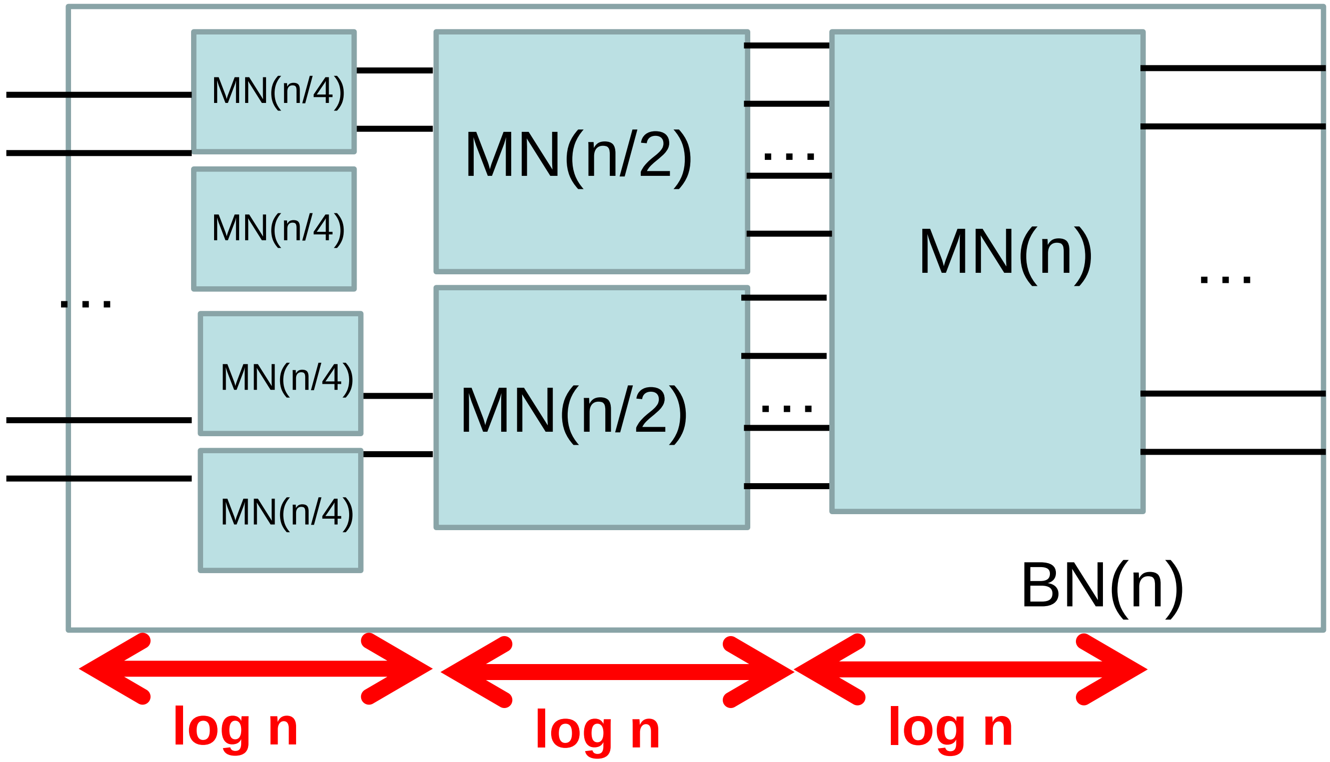


Like Merge-Sort: Sort larger and larger subsequences!
Example: BN(4)?
Sorting time / depth?

Batcher's

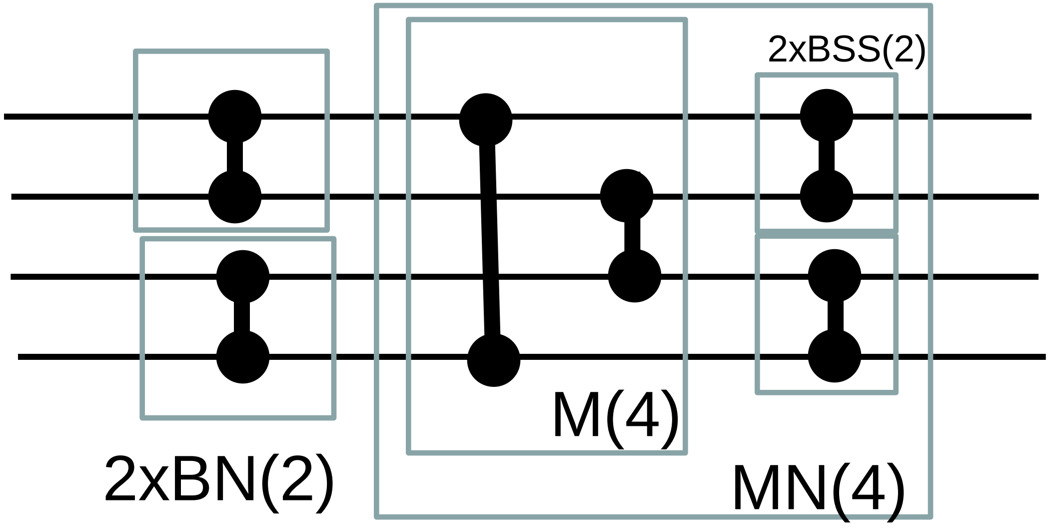

MN(n/4)
MN(n/4)
MN(n/4)
MN(n/4)
MN(n/2)
MN(n/2)
MN(n)
BN(n)
sorted eighths
sorted quarters
sorted halfs

Batcher's



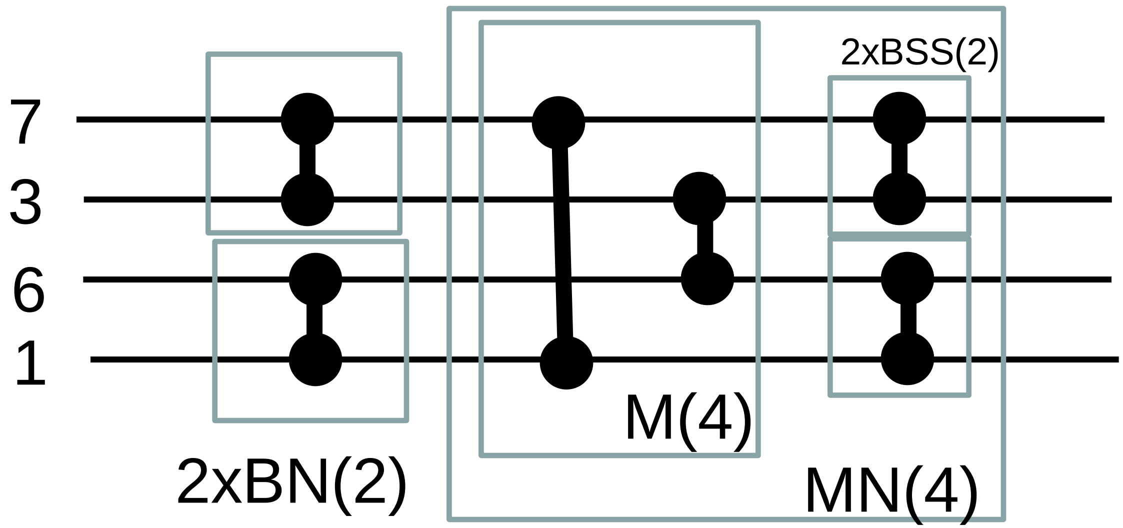
Example

Batcher Network BN(4), i.e. w=4:



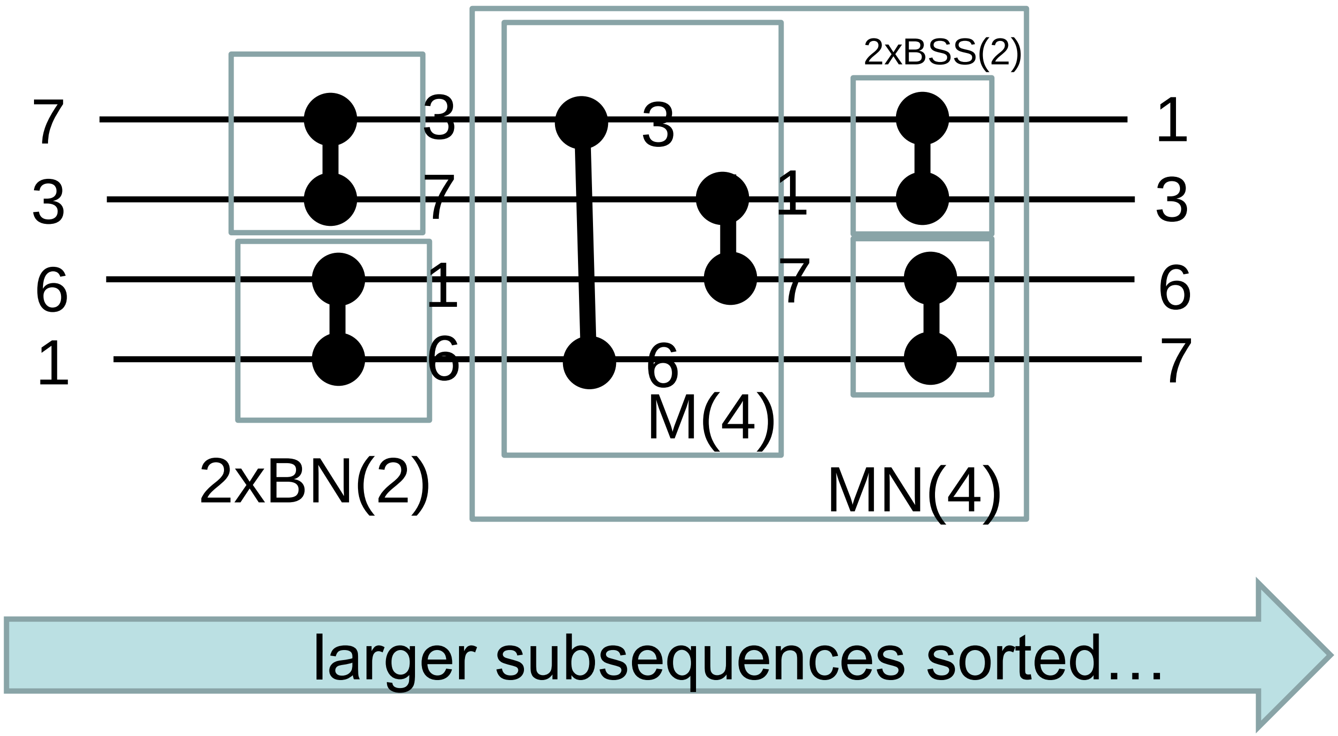
Example

Batcher Network BN(4), i.e. w=4:



Example

Batcher Network BN(4), i.e. w=4:



Batcher’s Sort

Batcher’s network sorts in $O(\log^2 n)$ time.

Proof:

Correctness: It’s like merge sort! At recursive stage k (for k=1,…, log n), we merge 2^k sorted sequences into 2^{k-1} sorted sequences.

Depth: Merging network has log n depth, and we have log n many.

■

Can we do better? Yes, but not in this lecture…

Remark:
- $O(\log^2 n)$ also possible in hypercubic networks / butterflies

End of Lecture