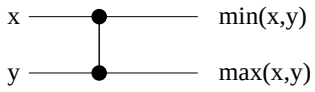
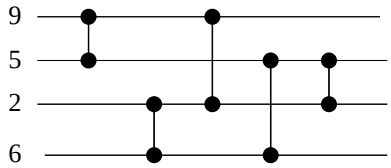
Lecture 10: Sorting networks

A parallel model of computation where comparisons can be made simultaneously.

Comparator:



Draw the following network, insert data and propagate them. The sequence will be sorted.

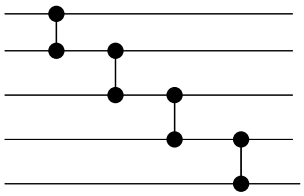


Argue that four arbitrary numbers are sorted by the network.

Running time = **depth** = longest path of comparators (3 in the example).

Note: depth is not always equal to the maximum number of comparators connected to a line.

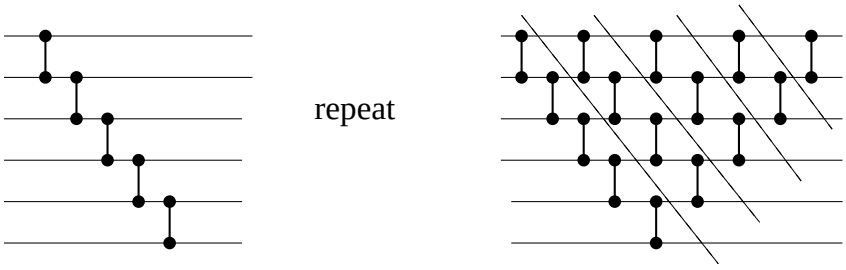
In the example, maximum number on a line is 3, which equals the depth. But consider:



It has has depth = 4, not 2. The depth of the network equals the time to produce all output values.

Size of network = number of comparators (5 in our example).

Example: Selection sort



Draw building block (left part), argue that it produces max of n inputs.

Sort by recursively finding max, the max of the rest, and so on. Add one building block at a time.

What is the depth of the network? Seems to be an arithmetic sum from 1 to n, i.e. $\Theta(n^2)$.

Pipelining: note that operations overlap. Depth is $(n-1)+(n-2) = 2n-3$, faster than any comparison based, sequential algorithm. Its size is $\Theta(n^2)$ though.

The same sorting network can be used to mimic insertion sort: draw diagonals the other way. The first comparator is then at the leftmost top.

How fast can we sort in parallel? Two arguments for an $\Omega(\log n)$ lower bound:

1. Simulating a sorting network on a sequential machine gives a comparison based algorithm, so $\Omega(n \log n)$ comparisons or comparators are needed. Since we can do at most $n/2$ comparisons at each depth in the network, the total depth is $\Omega(\log n)$.
2. Consider an arbitrary input element. After one level in the network the element can be on at most two lines, after two levels on at most four lines, after three levels on at most eight lines ... Hence need $\log n$ levels to move an element to its correct place, which can be any of n $(= 2^{\log n})$ outputs.

Rest of the lecture: Construction of an elegant, fast sorting network.

How to test a network for correctness? We could enter all $n!$ permutations and see if each gets sorted. The following says that we need only test 2^n input sequences, which is *much* less than $n!$

0-1 principle:
If a comparison network with n inputs sorts all 2^n sequences of 0's and 1's, then it sorts all sequences of arbitrary numbers

Proof: Let f be monotonically increasing: $x \leq y$ implies $f(x) \leq f(y)$.

Then min$(f(x), f(y)) = f(\min(x, y))$, and max$(f(x), f(y)) = f(\max(x, y))$.

If the network transforms input $\langle a_1, a_2, \ldots, a_n \rangle$ into output $\langle b_1, b_2, \ldots, b_n \rangle$, it transforms

$\langle f(a_1), f(a_2), \ldots, f(a_n) \rangle$ into $\langle f(b_1), f(b_2), \ldots, f(b_n) \rangle$

This can be shown by induction on the depth of the network, using the result above for min and max. We skip the proof.

Now suppose the 0-1 principle does not hold. Assume the network sorts all 0-1 sequences, but there is a sequence $\langle a_1, a_2, \ldots, a_n \rangle$ where $a_i < a_j$, but the network places a_j before a_i in the output.

Define the monotonically increasing function:

$$f_i(x) = \begin{cases} 0 & \text{if } x \leq a_i \\ 1 & \text{if } x > a_i \end{cases}$$

Then the network fails to sort $\langle f_i(a_1), \ldots, f_i(a_n) \rangle$, a 0-1 sequence, which contradicts the assumption that it sorts all 0-1 sequences.

We sketch a network with misordered outputs a_j and a_i and apply f_i to get misordered output bits.

It follows that we need only construct a 0-1 sorter to get a general sorter. □

Construction of a sorting network in three steps

Step 1: Construct a **bitonic sorter** (which sorts binary bitonic sequences).

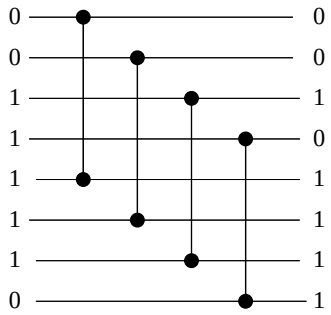
Bitonic sequence: $\nearrow\searrow$, or can be shifted circulary to look like that.

Binary bitonic sequences: $0\cdots01\cdots10\cdots0$, shortened 0 1 0

or $1\cdots10\cdots01\cdots1$, shortened 1 0 1 (where some segments may be empty).

Part network: **half-cleaner**

Example:



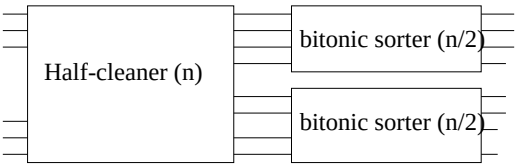
Depth is 1, and at least one of top half and bottom half is **clean** (having only 0's or only 1's).

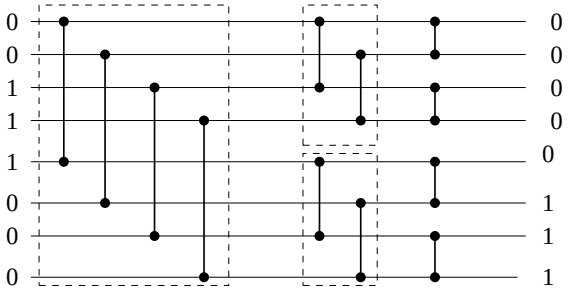
> If the input is bitonic the half-cleaner gives that:
> both output halves are bitonic (and at least one is clean);
> all elements in top half $\leq$ all elements in bottom half

Proof: We go through Figure 27.8, page 714, for 0 1 0. The argument is similar for 1 0 1

Bitonic sorter is built recursively with half-cleaners.

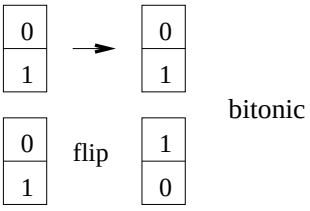
Example:





Depth: $D(n) = D(n/2) + 1 = \log n$, if $n = 2^k,\ k \geq 1$, with $D(1) = 0$.

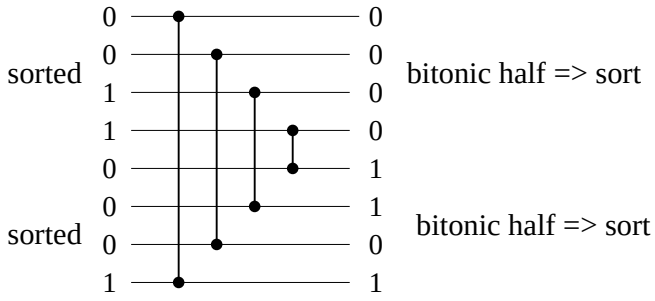
Step 2: Construct a **merging network** that merges two sorted sequences. Flip the second sequence and concatenate the two sequences. This gives a bitonic sequence.



We modify the first half-cleaner of the bitonic sorter. Perform the flip of the second half implicitly. Comparators instead of data are swapped (Figure 27.10, page 717).

By theorem in step 1, the top and bottom halves in the output sequence is thereby bitonic (and at least one half is clean). Hence we can apply the bitonic sorter.

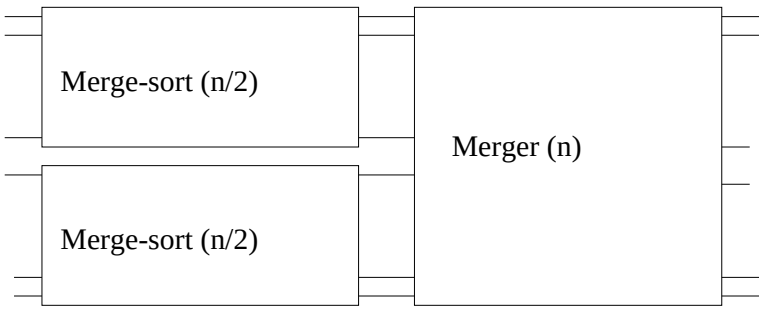
Merger(n):



Note that at least one of the bitonic halves has to be clean for the whole sequence to be properly sorted.

Depth is still $\log n$, only the first stage is different from the bitonic sorter.

Step 3: Merge-sort(n)



Depth: $D(1) = 0$ and $D(n) = D(n/2) + \log n \ = \ \Theta(\log^2 n)$.

Size of the network? Try compute it until tutorial 5.

By the 0-1 principle this network also sorts arbitrary numbers