Sorting Networks

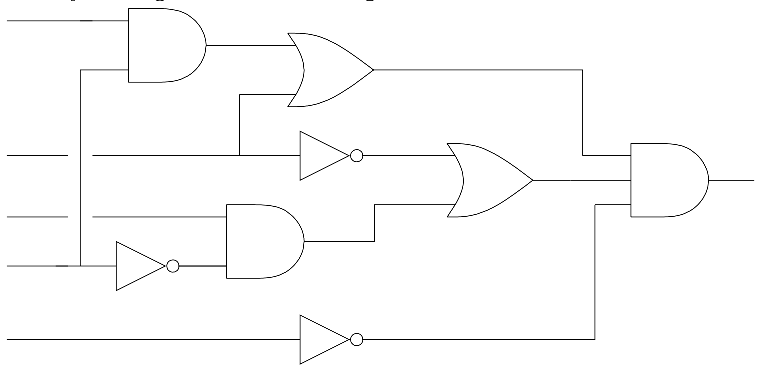
February 18, 2003

1 Model of Computation

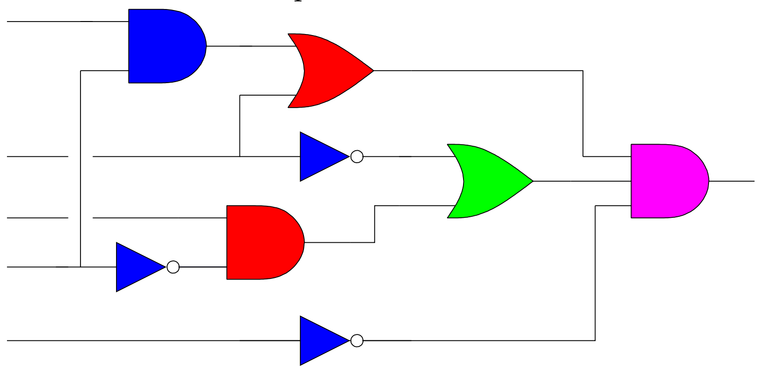
Can we do better by using different computes?

Example: Macaroni sort.

Can we do better by using the same computers?



How much time does it take to compute the value of this circuit?



We can compute this circuit in 4 time units. Namely, circuits are inherently parallel. Can we take advantage of this???

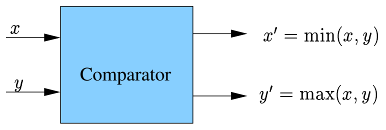
Let us consider the classical problem of sorting n numbers.

Q: Can one sort in *sublinear* by allowing parallel comparisons?

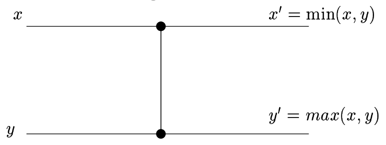
Q: What exactly is our computation model?

1.1 Computing with a circuit

We are going to design a circuit, where the inputs are the numbers, and we compare two numbers using a comparator gate:

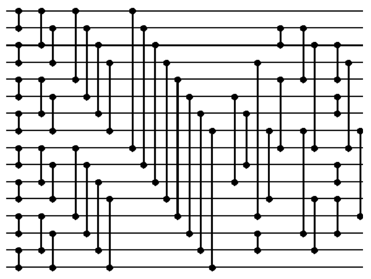


For our drawings, we will draw such a gate as follows:



So, circuits would just be horizontal lines, with vertical segments (i.e., gates) between them.

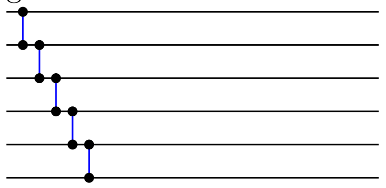
A complete sorting network, looks like:



The inputs come on the wires on the left, and are output on the wires on the right. The largest number is output on the bottom line.

The surprising thing, is that one can generate circuits from a sorting algorithm.

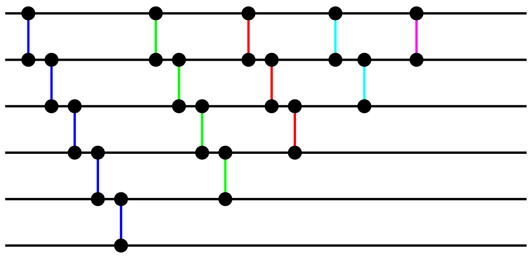
In fact, consider the following circuit:



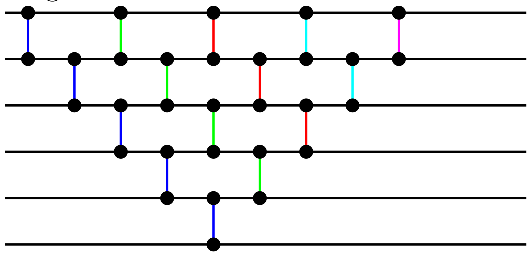
Q: What does this circuit does?

A: This is the inner loop of insertion sort.

Repeating this inner loop, we get the following sorting network:

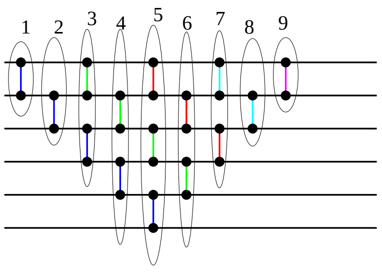


Alternative way of drawing it:



Q: How much time does it take for this circuit to sort the n numbers?

Running time = how many time clocks we have to wait till the result stabilizes. In this case:



In general, we get:

Lemma 1.1 *Insertion sort requires $2n - 1$ time units to sort n numbers.*

Q: Can we do better?

1.2 Definitions

Definition 1.2 A *comparison network* is a DAG (directed acyclic graph), with n inputs and n outputs, which each gate has two inputs and two outputs. (Note that a comparison

Definition 1.3 The *depth* of a wire is 0 at the input. For a gate with two inputs of depth d_1 and d_2 the depth of the output is $1 + \max(d_1, d_2)$.

The depth of a *comparison network* is the maximum depth of an output wire.

Definition 1.4 A *sorting network* is a comparison network such that for any input, the output is monotonically sorted. The *size* of a sorting network is the number of gates in the sorting network. The *running time* of a sorting network is just its depth.

2 The Zero-One Principle

The zero-one principle:

If a comparison network sort correctly all binary inputs (i.e.., every number is either 0 or 1) then it sorts correctly all inputs.We of course, need prove that the zero-one principle is true.

Lemma 2.1 *If a comparison network transforms the input sequence $a = \langle a_1, a_2, \ldots, a_n \rangle$ into the output sequence $b = \langle b_1, b_2, \ldots, b_n \rangle$, then for any monotonically increasing function f, the network transforms the input sequence $f(a) = \langle f(a_1), \ldots, f(a_n) \rangle$ into the sequence $f(b) = \langle f(b_1), \ldots, f(b_n) \rangle$.*

Proof: Consider a single comparator with inputs x, y, and outputs $x' = \min(x, y)$ and $y' = \max(x, y)$. If $f(x) = f(y)$ then the claim trivially holds for this comparator. If $f(x) < f(y)$ then clearly

$$\begin{aligned} \max(f(x), f(y)) &= f(\max(x, y)) \text{ and} \\ \min(f(x), f(y)) &= f(\min(x, y)) \end{aligned}$$

Thus, for input $\langle x, y \rangle$, for $x < y$, we have output $\langle x, y \rangle$ thus
for input $\langle f(x), f(y) \rangle$ the output is $\langle f(x), f(y) \rangle$
Thus, for input $\langle x, y \rangle$, for $x > y$, we have output $\langle y, x \rangle$ thus
for input $\langle f(x), f(y) \rangle$ the output is $\langle f(y), f(x) \rangle$
Establishing the claim for one comparator.

This implies that if a wire carry a value a_i when the network get input $a_1, \ldots, a_n$ then for the input $f(a_1), \ldots, f(a_n)$ this wire would carry the value $f(a_i)$. This follows immediately by using the above claim for a single comparator together with induction on the network structure.

This immediately implies the lemma. ■

Theorem 2.2 *If a comparison network with n inputs sorts all 2^n binary strings of length n correct, then it sorts all sequences correctly.*

Proof: Assume for the sake of contradiction, that it sorts incorrectly the sequence $a_1, \ldots, a_n$. Let $b_1, \ldots b_n$ be the output sequence for this input.

Let $a_i < a_k$ be the two numbers that outputted in incorrect order (i.e. a_k appears before a_i in the output). Let $f(x) = \begin{cases} 0 & x \leq a_i \\ 1 & x > a_i \end{cases}$. Clearly, by the above lemma, for the input
$f(a_1), \ldots, f(a_n)$ - binary circuit
the circuit would output $f(b_1), \ldots, f(b_n)$. But then, this sequence looks like 000..0????$f(a_k)$????$f(a_i)$??1 but $f(a_i) = 0$ and $f(a_j) = 1$. Namely, the output is

$$????1????0????.$$

Namely, we have a binary input $(f(b_1), \ldots, f(b_n))$ for which the comparison network does not sort it correctly. A contradiction to our assumption. ■

3 A bitonic sorting network

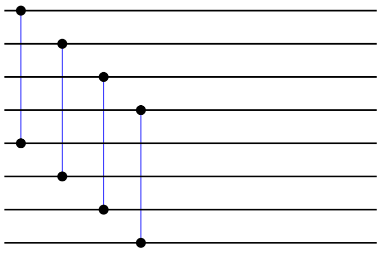
Definition 3.1 A bitonic sequence is a sequence which is first increasing and then decreasing, or can be circularly shifted to become so.

Example 3.2 $(1, 2, 3, \pi, 4, 5, 4, 3, 2, 1)$ - bitonic.
$(4, 5, 4, 3, 2, 1, 1, 2, 3)$ - bitonic
$(1, 2, 1, 2)$ - not bitonic.

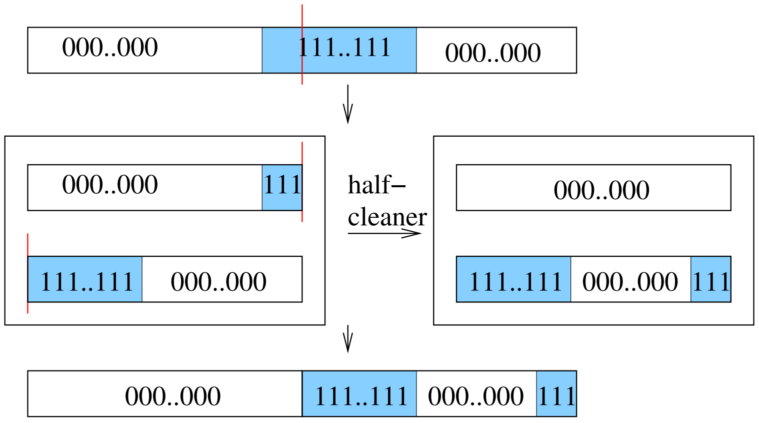
Observation 3.3 *A bitonic sequence over* $0, 1$ *is either of the form* $0^i 1^j 0^k$ *or of the form* $1^i 0^j 1^k$ *where* 0^i *denote a sequence of* i *zeros.*

Definition 3.4 A bitonic sorter is a comparison network that sort bitonic sequences.

Definition 3.5 A half-cleaner is a comparison network, connecting line i with line $i + n/2$.



A $Half-Cleaner[n]$: A half-cleaner with n inputs.
The depth of a $Half-Cleaner[n]$ is one.
What does a half-cleaner do for a (binary) bitonic sequence?



Example 3.6 Properties:

1. Left side is clean and all 0.

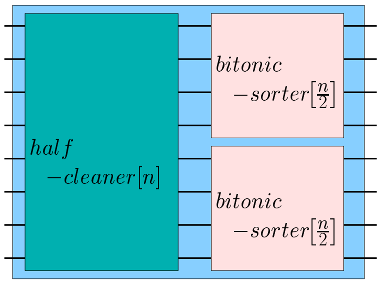
2. Right side is bitonic.

Lemma 3.7 *If the input to a half-cleaner is a binary bitonic sequence then for the output sequence:*

1. *The elements in the top half are smaller than the elements in bottom half.*

2. One of the halves is clean, and the other is bitonic.

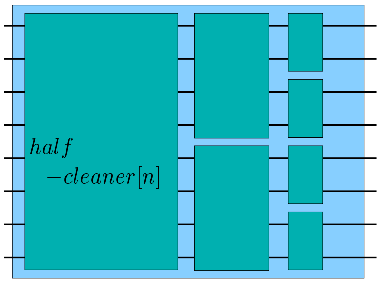
This suggests a simple recursive construction of $Bitonic - Sorter[n]$:



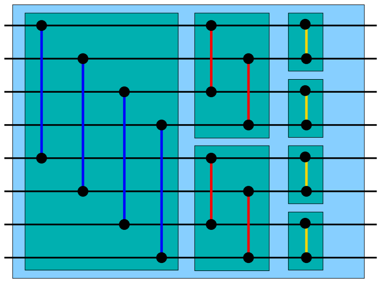
$half$ $-cleaner[n]$
$bitonic$ $-sorter[\frac{n}{2}]$
$bitonic$ $-sorter[\frac{n}{2}]$

$bitonic - sorter[n]$

Opening the recursion, we have:



Namely, $Bitonic - Sorter[n]$ is



Lemma 3.8 *$Bitonic - Sorter[n]$ sorts bitonic sequences of length $n = 2^k$, it uses $(n/2)k = \frac{n}{2}\lg n$ gates, and it is of depth $k = \lg n$.*

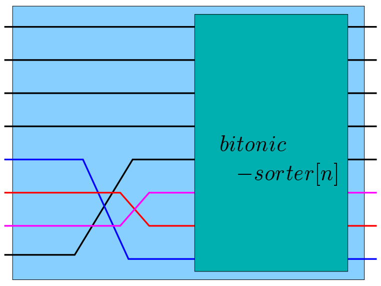
3.1 Merging sequence

Q: Given two *sorted* sequences of length $n/2$ how do we merge them into a single sorted sequence?

A:Concatinate the two sequence, where the second sequence is being flipped. It is easy to verify that the resulting sequence is bitonic, and as such we can sort it using the $Bitonic - Sorter[n]$.

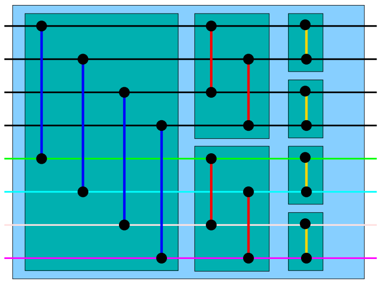
Observation 3.9 *Given two sorted sequences $a_1 \leq a_2 \leq \ldots \leq a_n$ and $b_1 \leq b_2 \leq \ldots \leq b_n$ the sequence $a_1, a_2, \ldots, a_n, b_n, b_{n-1}, b_{n-2}, \ldots, b_2, b_1$ is bitonic.*

Thus, to merge two sorted sequences of length $n/2$, just flip one of them, and use $bitonic - sorter[n]$:

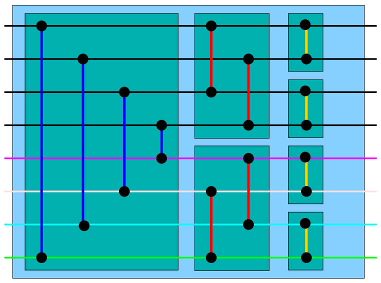


$merger[n]$

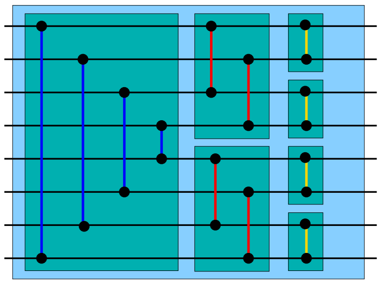
This is of course, illegal. What we do, is to take $bitonic - sorter[n]$ and physically flip the last $n/2$ entries:



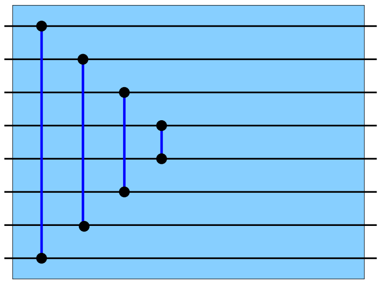
Thus, $Merger[n]$ is



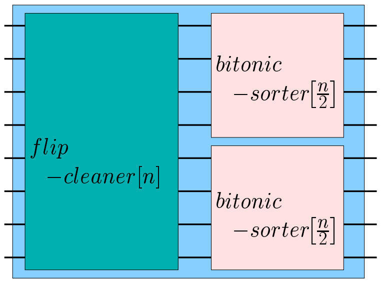
But this is equivalent to:



Which is $Bitonic - Sorter[n]$ with the first component flipped. Formally, let $flip - cleaner[n]$ to be the component:



Thus, $Merger[n]$ is just the comparator network:



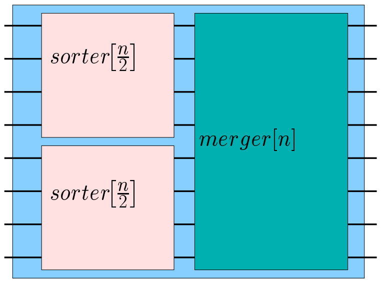
Lemma 3.10 *$Merger[n]$ gets as input two sorted sequences of length $n/2 = 2^{k-1}$, it uses $(n/2)k = \frac{n}{2}\lg n$ gates, and it is of depth $k = \lg n$, and it outputs a sorted sequence.*

4 Sorting Network

Q: How to build a sorting network?

A:Just implement merge sort using $Merger[n]$.

$sorter[n]$:



Lemma 4.1 *$Sorter[n]$ is a sorting network (i.e., it sorts any n numbers) using $G(n) = O(n\log^2 n)$ games, and of depth $O(\log^2 n)$. Namely, using sorting networks, one can sort n numbers in $O(\log^2 n)$ time.*

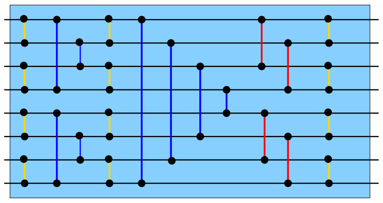
Proof: The number of gates is

$$G(n) = 2G(n/2) + Gates(Merger[n])$$

$$G(n) = 2G(n/2) + O(n\log n) = O(n\log^2 n)$$

As for the depth $D(n) = D(n/2) + Depth(Merger[n]) = D(n/2) + O(\log(n))$

Thus, $D(n) = O(\log^2 n)$. ■ Here is how $Sorter[8]$ looks like:



5 Faster sorting networks

One can build a sorting network of logarithmic depth (see [AKS83]). The construction however is very complicated. A simpler parallel algorithm would be discussed sometime in the next lectures. BTW, the AKS construction [AKS83] mentioed above, is better than bitonic sort for n larger than 2^{8046}.

References

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 1–9, 1983.