Sorting Networks

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Spring 2012 ; distribution outside this group of students is NOT allowed.*

(c) Copyright A. Gerbessiotis. All rights reserved.

Comparison Networks

A sorting network is a (comparison) network that always sorts its inputs by performing comparisons only (i.e. countsort, radix sort can not be implemented in such networks). Sorting networks are special cases of a broader class of networks known as comparison networks.

A comparison network consists of wires and comparators. A **comparator** is a box with two inputs x and y and two outputs a and b. The comparator compares x and y and outputs through a the minimum of x and y and through b the maximum of x and y. The box of Fig 1. in most cases is simplified into the vertical line of Fig 2. Each comparator takes $O(1)$ time steps to compare the two keys and output them in proper order. The wires transmit values as a whole (think of parallel communication rather than serial) from left to right. If multiple comparators are attached to a horizontal line that transfers the input values the result of a preceding comparator must become available before a succeeding comparator starts performing the comparison required. The **depth** of a sorting network is the maximum number of comparators attached in a path (not line) from an input to an output. Thus if all input lines are of depth 0, and the input wires to a comparator are of depth d_1 and d_2 then the two output wires of the comparator are of the same depth $\max\{d_1, d_2\} + 1$.

A sorting network is a comparison network whose output lines generate the input sequence in monotonically increasing order top-to-bottom (i.e. smallest element appear on top-right output line and largest in bottom-right).

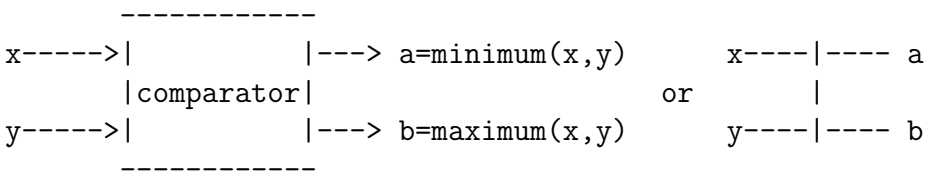


Fig 1. Fig 2.

The sorting algorithm implied by a sorting network is an **oblivious sorting algorithm**. An oblivious algorithm is an algorithm whose actions are always the same and independent of the input and output: in an oblivious sorting algorithm the sequence of comparisons performed is the same for all inputs and outputs.

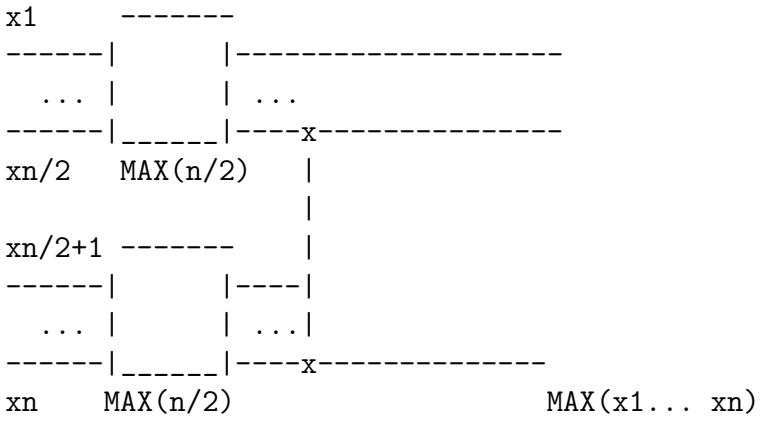
(c) Copyright A. Gerbessiotis. CS 667-101 : Spring 2012 . All rights reserved.

Finding the maximum

Suppose we are interested in Finding the maximum of n values. Let n be a power of two.

One way to build a comparison network for finding the maximum is divide and conquer. Suppose that a comparison network with n input lines and n output lines finds the maximum of n values fed through the input lines and this maximum appears through the bottom wire of the output. We can construct this network recursively by first constructing two networks, one that finds the maximum of the first $n/2$ values and one that finds the maximum of the remaining $n/2$ values. Then, a single comparison determines the maximum of the two maxima, i.e. the maximum of the n values.

```
MAX(i): A box with i input and i output wires.
        The maximum appears through the bottom output wire
```



Let $S(n)$ be the size of $MAX(n)$. Then,

$$S(n) = 2S(n/2) + 1$$

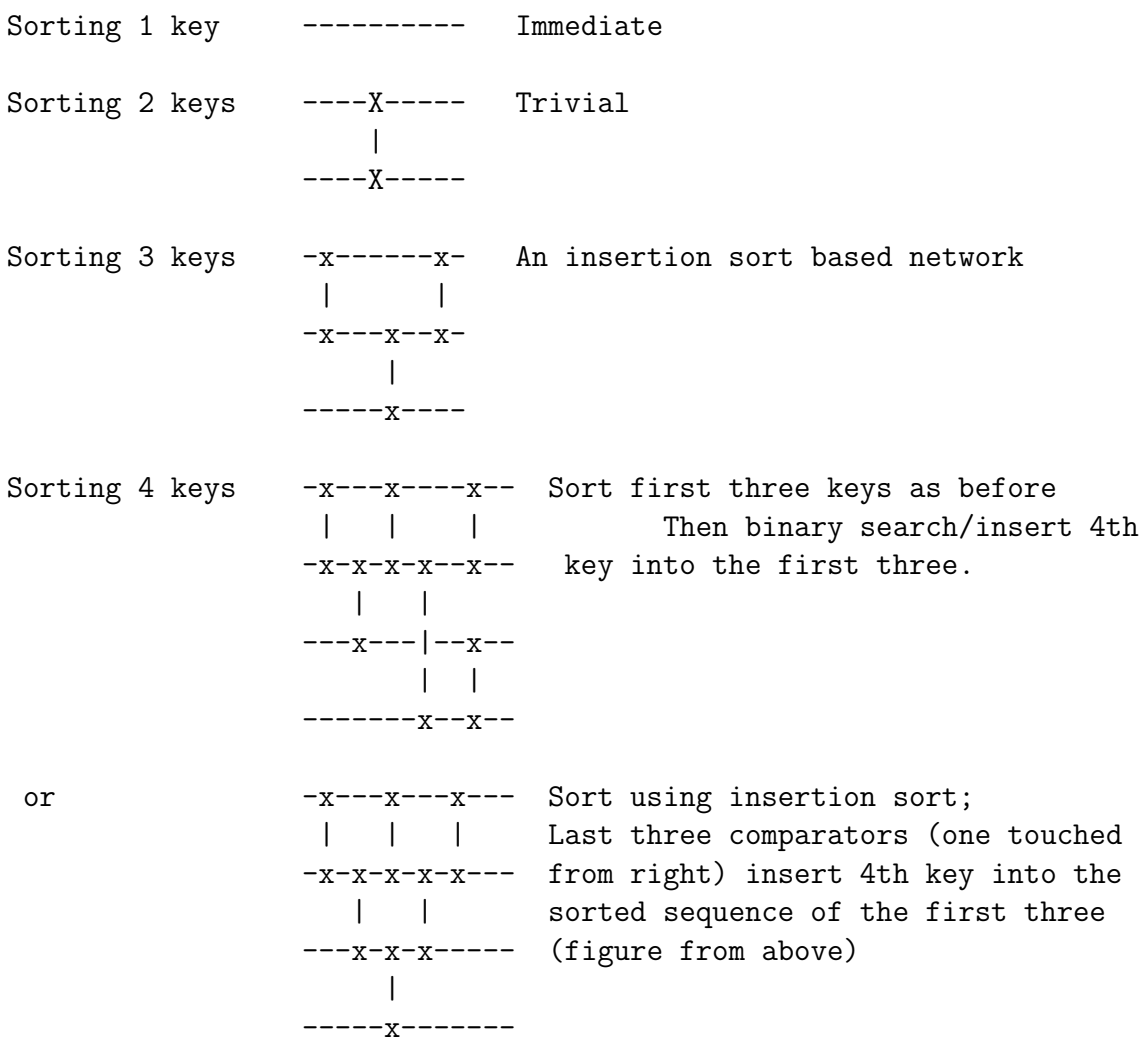
We have $S(2) = 1$. A solution for $S(n)$ is $S(n) = n - 1$. Let $D(n)$ be the depth/delay of $MAX(n)$. Then,

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

We have $D(2) = 1$. A solution for $D(n)$ is $D(n) = \lg n$.

(c) Copyright A. Gerbessiotis. All rights reserved.

Some simple networks



(c) Copyright A. Gerbessiotis. All rights reserved.

0-1 Sorting Lemma

The 0-1 Sorting Lemma also referred to as 0-1 (Sorting) principle can be used to show that a comparison network is indeed a sorting network. The 0-1 sorting lemma says that if a sorting network sorts all inputs of zeroes and ones correctly, then it can sort any type of inputs as well, be them floating point numbers, integers, etc. Note that the lemma applies to sorting "algorithms" that are oblivious and use only comparisons and do not inspect the values of the input.

Lemma MonotonicP. If a comparison network transforms $a = \langle a_1, \ldots, a_n \rangle$ into a $b = \langle b_1, \ldots, b_n \rangle$, then for any monotonically increasing function f, the network transforms $f(a) = \langle f(a_1), \ldots, f(a_n) \rangle$, into $f(b) = \langle f(b_1), \ldots, f(b_n) \rangle$.

Proof MonotonicP. A proof of the monotonic property will be by induction.

The basis of the induction is the proof that a single comparator has the monotonic property. Suppose that the inputs to a comparator are x and y. If $x \leq y$, then $f(x) \leq f(y)$ by the monotonicity of f. In the former case x is output on the top output line and in the latter case $f(x)$ is output on the top output line. A similar observation applies to the case $x \geq y$, where $y, f(y)$ are on the top output line. Therefore $\min(f(x), f(y)) = f(\min(x, y))$, and $\max(f(x), f(y)) = f(\max(x, y))$.

A comparator C at depth d has input lines that are of depth strictly less that d. If these input lines carry a_i and a_j when the input is a, they will carry, by the inductive hypothesis $f(a_i)$ and $f(a_j)$, when the input is $f(a)$. This completes the induction, as C would produce. $f(\min(a_i, a_j))$, $f(\max(a_i, a_j))$ by the base case.

0-1 Sorting Lemma If a comparison network with n inputs sort all 2^n possible input sequences of zeroes and ones, then it sort all sequences of arbitrary numbers correctly.

Proof 0-1SL. Suppose that the comparison network sorts all 2^n binary n inputs, but fails to sort a sequence s of arbitrary numbers. Then this means that in sequence s there are two numbers s_i and s_j that are out of order, i.e. $s_i < s_j$ but the network places s_j before s_i. We then define the following monotonically increasing sequence.

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

Since the network places s_j before s_i then given a monotonically increasing f it would place $f(s_j) = 1$ before $f(s_i) = 0$ in the output sequence, i.e. it would place an 1 before a 0. Then this specific input sequence of 0's and 1's (implied by f) would not be sorted contradicting the assumption that all binary sequences are sorted correctly.

(c) Copyright A. Gerbessiotis. All rights reserved.

Building Steps

In order to build a sorting network, we will proceed in steps.

- We first show how to sort more structured sequences called bitonic sequences, and thus how to construct a bitonic sorting network.
- We then show how to merge two sorted sequences and thus how to construct a merging sorting network.
- We finally show how to sort an arbitrary sequence.

(c) Copyright A. Gerbessiotis. All rights reserved.

Bitonic sorting

A **bitonic sequence** is a sequence that

S1 monotonically increases and then monotonically decreases or

S2 is a circular shift (rotation) of a sequence described in case S1.

For example $\langle 4, 7, 8, 6, 5, 3, 2, 1\rangle$ is a bitonic sequence, and so is $\langle 7, 8, 6, 5, 3, 2, 1, 4\rangle$ that results from shifting/rotating the former sequence one position to the left.

We first show how to sort a bitonic sequence of length n.

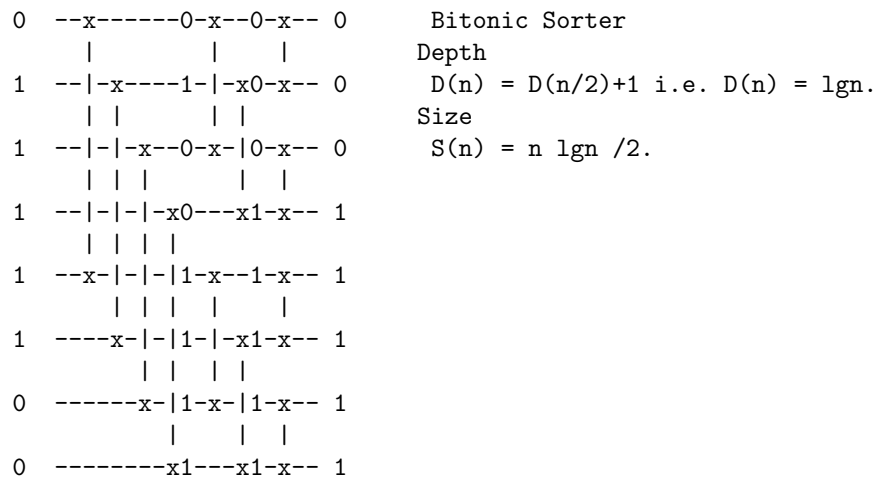
In the remainder we will assume that n is a power of two. If this is not the case, we can pad keys in the sequence so that its length becomes a power of two.

Idea. A comparison network where line i is connected to line $i + n/2$, $0 \leq i < n/2$, turns a bitonic sequence of 0's and 1's into two bitonic sequences of half the original size, one of which is clean (all output lines contain one kind of input).

Algorithm BitonicSort. Use the idea recursively until we "clean" all the lines. The first application cleans half the input (which becomes a trivial bitonic sequence) and turns the other half into a bitonic sequence whose elements are smaller than (or equal to) the cleaned ones if it is the top-half (and at least than the cleaned elements if it is the bottom half).

Analysis of BitonicSort. This will take $\lg n$ phases. Each phase consists of $n/2$ comparators. The depth of the network is $\lg n$ and its size $n \lg n/2$.

(c) Copyright A. Gerbessiotis. All rights reserved.



BitonicSort Algorithm Correctness

In order to prove the correctness of this scheme we distinguish four cases assuming the input is of the form $\bar{0}\bar{1}\bar{0}$, where $\bar{1}, \bar{0}$ denote a block of 1's and 0's of arbitrary (potentially not equal) size respectively. The symmetric case $\bar{1}\bar{0}\bar{1}$ is treated analogously.

a $\bar{1}$ does not include the midpoint and falls in top half.

b $\bar{1}$ does not include the midpoint and falls in bottom half.

c $\bar{1}$ includes midpoint and length less than $n/2$.

d $\bar{1}$ includes midpoint and length not less than $n/2$.

In case (a) a block $\bar{0}$ of length at least $n/2$ results in top half (i.e it is clean). Same for case (b). In case (c) we have the same. In case d we have a block $\bar{1}$ of length $n/2$ in the bottom/upper half, and a bitonic sequence in the bottom half.

In all cases we obtain at least one clean half, two bitonic sequences, and every element in the top half is less than or equal to every element in the bottom half.

(c) Copyright A. Gerbessiotis. All rights reserved.

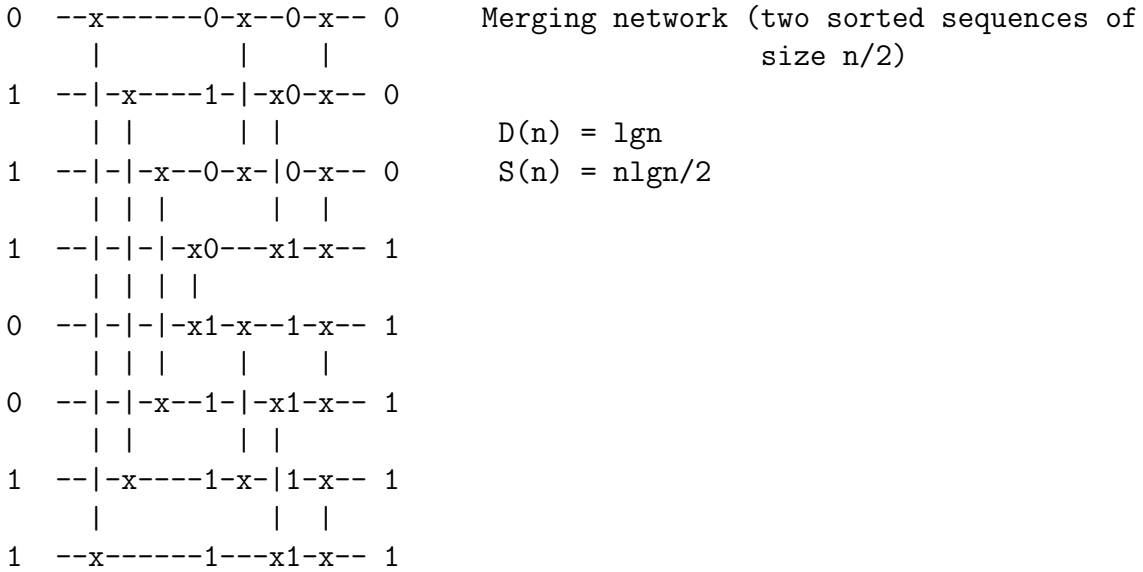
Merging

The problem of merging deals with merging two sorted sequences into one sorted sequence. Suppose the total length of the sorted sequence is $n = k + l$, and k, l are the lengths of the two sequences to be merged. In the remainder we will assume that $k = l = n/2$. **The sequence formed by the first sequence and the reverse of the latter is bitonic**. Since it is bitonic it can be sorted with the bitonic sorter presented in the previous section.

The only problem that needs to be resolved is transforming two sorted sequences into a bitonic sequence. This transformation can be incorporated in the first phase of the bitonic network described in the previous section.

We recall that in the first phase of the bitonic network line i is compared to $i + n/2$ and a clean sequence (which is also trivially bitonic) of length $n/2$ results in one of the halves, and a bitonic sequence results in the other half.

The key of the input sequence that should be in the $i + n/2$ line if the input sequences was to be converted into a bitonic sequence would have been the i-th largest key of the second sequence that would be reversed. However that key is currently in line $n - i + 1$. The easy way to resolve this problem is by performing a comparison not between i and $i + n/2$ but between i and $n - i + 1$ instead. After the first phase is completed we get two bitonic sequences for the top and bottom half and thus the remaining phase of the bitonic sorter need not be modified.



Sorting Networks

We are now ready to introduce the algorithm for sorting, i.e. build a sorting network that sorts its input. The algorithm is merge-sort based. In order to sort n keys we produce two sorted sequences recursively (divide and conquer step) and merge these sequences (combining step), by the n-line merging network of the previous section. The basis of the recursive decomposition is easy. A sequence consisting of a single key is already sorted.

Therefore using this divide and conquer construction of a sorting network we conclude the following. The depth and size of such a network can be found as follows.

$$D(n) = D(n/2) + \lg n$$

This gives $D(n) = O(\lg^2 n)$ by the master method and $\lg n(1 + \lg n)/2$ by the iteration method.

$$S(n) = 2S(n/2) + O(n \lg n)$$

This gives $S(n) = O(n \lg^2 n)$ by the master method.

