8 Sorting

"Indeed, I believe that virtually *every* important aspect of programming arises somewhere in the context of sorting and searching!"

– Donald E. Knuth, The Art of Computer Programming

8.1 Array

Definition 8.1 (Sorting) *We are given a graph with nodes $v_1, \ldots, v_n$. Initially each node stores a value. After applying a sorting algorithm, node v_k stores the kth smallest value.*

Remarks:

- What if you route all values to the same central node, then sort the values locally, then route them to the correct destinations? Great, but what about contention?
- Problem: The "classic" contention definition is on edges, not nodes; a star-graph gives therefore a trivial O(n) message and O(1) time sorter. We need a stronger contention definition than remark 4.3.
- In this chapter algorithms can only store $O(1)$ values at a time. (Or not quite as strong but sufficient: Use node contention instead of message contention.)
- Ranking + Routing = Sorting? Well... indeed routing and sorting have many commonalities. As in Chapter 4 we start with Arrays, go then to Meshes and finally end up with something like Butterflies!

Algorithm 8.2 (Odd/Even Sort) *We are given an array of n nodes, each storing a value (in scrambled order). At odd steps, we compare and exchange – if the value at the left node is larger than the value at the right node – the values in nodes 1 and 2, 3 and 4, etc. At even steps, we do the same for the pairs 2 and 3, 4 and 5, etc.*

Remarks:

- How fast is the algorithm, and how can we prove correctness/efficiency?
- The most interesting proof uses the so-called 0-1 Sorting Lemma. It allows us to restrict our attention to an input of 0's and 1's, and works for any "oblivious comparison-exchange" algorithm. (Which means: Whether you exchange two values must only depend on the relative order of the two values, and not on anything else.)

Lemma 8.3 (0-1 Sorting Lemma) *If an oblivious comparison-exchange algorithm sorts all inputs of 0's and 1's, then it sorts any input.*

Proof. We prove the opposite direction (does not sort any input $\Rightarrow$ does not sort 0's and 1's). Assume that there is an input $x = x_1, \ldots, x_n$ that is not sorted correctly. Then there is a smallest value k such that the value at node v_k after running the sorting algorithm is larger than the kth smallest value $x(k)$. Define an input $x_i^* = 0 \Leftrightarrow x_i \leq x(k)$, $x_i^* = 1$ else. Since $x_i \geq x_j \Rightarrow x_i^* \geq x_j^*$ all the compare-exchange operation are the same with x^* as with the original input x. The output with only 0's and 1's will also not be correct. ⊓

Theorem 8.4 (Analysis) *Algorithm 8.2 sorts correctly in n steps.*

Proof. Thanks to Lemma 8.3 we only need to consider an array with 0's and 1's. Let j_1 be the node with the rightmost 1. If j_1 is odd (even) it will move in the first (second) step. In any case it will move right in every following step until it reaches the rightmost node. Let j_k be the node with the k'th rightmost 1. We show by induction that j_k is not blocked anymore (moves until it reaches destination!) after step k. We have already anchored the induction at $k = 1$. Since j_{k-1} moves after step $k-1$, j_k gets a right 0-neighbor for each step after step k. (We supressed a couple of details.) ⊓

Remark:

- As for routing we continue with the two-dimensional array.

8.2 Mesh

Algorithm 8.5 (Shearsort) *We are given a mesh with m rows and m columns ($n = m^2$). The sorting algorithm operates in phases, and uses the odd/even sort algorithm on rows or columns. In particular, in the odd phases $1, 3, \ldots, \log m + 1$ we sort all the rows, in the even phases $2, 4, \ldots, \log m$ we sort all the columns. Columns are sorted such that the small values move up. Odd rows ($1, 3, \ldots, m-1$) are sorted such that small values move left. Even rows ($2, 4, \ldots, m$) are sorted such that small values move right.*

Theorem 8.6 (Analysis) *Algorithm 8.5 sorts the values in $m(\log n + 1)$ time in snake-like order.*

Proof. Since the algorithm is oblivious, we can use Lemma 8.3. We show that after a row and a column phase, half of the previously unsorted rows will be sorted. More formally, let us call a row with only 0's or 1's clean, a row with 0's and 1's dirty. At any stage, the rows of the mesh can be devided into three regions. In the north we have a region of all-0 rows, in the south all-1 rows, in the middle a region of dirty rows. Initially all rows can be dirty. Since neither the row- nor the column sort will touch the already clean rows, we can concentrate on the dirty rows.

First we run an odd phase. Then, in the even phase, we run a peculiar column sorter: We group two consecutive dirty rows into pairs. Such a pair can be in one of three states. Either we have more 0's than 1's, or more 1's than 0's, or an equal number of 0's and 1's. Column-sorting each pair will give us at least one clean row (and two clean rows if "$|0| = |1|$"). Then move the cleaned rows north/south and we will be left with half the dirty rows.

At first glance it appears that we need such a peculiar column sorter. However, any column sorter sorts the columns in exactly the same way (we are very grateful to have Lemma 8.3!!!).

All in all we need $2 \log m = \log n$ phases to remain only with 1 dirty row in the middle which will be sorted (not cleaned) with the last row-sort. $\sqcap$

Remarks:

- There are algorithms that sort in $3m + o(m)$ time on an m by m mesh (by diving the mesh into smaller blocks). This is asymptotically optimal, since a value might need to move $2m$ times.
- We know sequential algorithms that sort in $O(n \log n)$ time. With lots of parallelism, there might be a way to sort in time $O(\log n)$!?! The butterfly and its relatives have constant degree and only $O(\log n)$ diameter! Is there a network and a distributed algorithm that sorts in $O(\log n)$ time?!?

8.3 Sorting Networks

Definition 8.7 (Sorting Networks) *A comparator is a device with two inputs x, y and two outputs x', y' such that $x' = min(x, y)$ and $y' = max(x, y)$. We construct so-called comparison networks that consist of wires that connect comparators (the output port of a comparator is sent to an input port of another comparator). Some wires are not connected to output comparators, and some are not connected to input comparators. The first are called input wires of the comparison network, the second output wires. Given n values on the input wires, a sorting network sorts these values on the output wires. (These definitions are only comprehensable with the examples on blackboard. For a thorough introduction see any of the textbooks on sorting networks.)*

Remark:

- Note that a sorting network is an oblivious comparison-exchange network. Consequently we apply Lemma 8.3 throughout the Section.

Definition 8.8 (Depth) *The depth of an input wire is 0. The depth of a comparator is the maximum depth of its input wires plus one. The depth of an output wire of a comparator is the depth of the comparator. The depth of a comparison network is the maximum depth (of an output wire).*

Remarks:

- Often we will draw all the wires on n horizontal lines (n being the "width" of the network). Comparators are then vertically connecting two of these lines.

Definition 8.9 (Bitonic Sequence) *A bitonic sequence is a sequence of numbers that first monotonically increases, and then monotonically decreases, or vice versa.*

Remarks:

- $< 1, 4, 6, 8, 3, 2 >$ or $< 5, 3, 2, 1, 4, 8 >$ are bitonic sequences, but $< 9, 6, 2, 3, 5, 4 >$ is not.
- Since we restrict ourselves to 0's and 1's, bitonic sequences have the form $0^i 1^j 0^k$ or $1^i 0^j 1^k$ for $i, j, k \geq 0$.

Algorithm 8.10 (Half Cleaner) *A half cleaner is a comparison network of depth 1, where we compare wire i with wire $i + n/2$ for $i = 1, \ldots, n/2$ (we assume n to be even).*

Lemma 8.11 *Feeding a bitonic sequence into a half cleaner, the half cleaner cleans (makes all-*0 *or all-*1*) either the upper or the lower half of the* n *wires. The other half is bitonic.*

Proof. Assume that the input is of the form $0^i1^j0^k$ for $i, j, k \geq 0$. If the midpoint falls into the 0's, the input is already half clean and will stay so. If the midpoint falls into the 1's the half cleaner acts as Shearsort with two adjacent rows. (We refer to Theorem 8.6, and do details on the blackboard). The case $1^i0^j1^k$ is symmetric. ⊓

Algorithm 8.12 (Bitonic Sequence Sorter) *A bitonic sequence sorter of width* n *(*n *being a power of* 2*) consists of a half cleaner of width* n*, and then two bitonic sequence sorters of width* $n/2$ *each. A bitonic sequence sorter of width* 1 *is empty.*

Lemma 8.13 (Analysis) *A bitonic sequence sorter of width* n *sorts bitonic sequences. It has depth* $\log n$*.*

Proof. Follows directly from the definition of the bitonic sequence sorter (Algorithm 8.12) and Lemma 8.11. ⊓

Remark:

- Clearly we want to sort arbitrary and not only bitonic sequences!

Algorithm 8.14 (Merging Network) *A merging network of width* n *is a merger followed by two bitonic sequence sorters of width* $n/2$*. A merger is a depth-one network where we compare wire* i *with wire* $n - i + 1$*, for* $i = 1, \ldots, n/2$*.*

Remark:

- Indeed a merging network is a bitonic sequence sorter where we replace the biggest (first) half-cleaner by a merger.

Lemma 8.15 (Analysis) *A merging network merges two sorted input sequences into one.*

Proof. We have two sorted input sequences. The first merger step we know all-too-well already (Lemma 8.11, Theorem 8.6): After the merger step either the upper or the lower half is clean, the other is bitonic. The bitonic sequence sorters complete sorting, thus merge the two input sequences. ⊓

Remark:

- How do you sort n values when you are able to merge two sorted sequences of size $n/2$? Apply the merger recursively.

Algorithm 8.16 (Batcher's "Bitonic" Sorting Network) *A batcher sorting network of width* n *consists of two batcher sorting networks of width* $n/2$ *followed by a merging network of width* n*. A batcher sorting network of width* 1 *is empty.*

Theorem 8.17 (Analysis) *A batcher sorting network sorts an arbitrary sequence of values. It has depth* $O(\log^2 n)$*.*

Proof. Correctness is immediate: at recursive stage k ($k = 2, 4, 8, \ldots, n$) we merge $n/(2k)$ sorted sequences into n/k sorted sequences. The depth $d(n)$ of the sorter of level n is the depth of a sorter of level $n/2$ plus the depth $m(n)$ of a merger with width n. The depth of a sorter of level 1 is 0 since the sorter is empty. Since a merger of width n has the same depth as a bitonic sequence sorter of width n, we know by lemma 8.12 that $m(n) = \log n$. This gives a recursive formula for $d(n)$ which solves to $d(n) = \frac{1}{2}\log^2 n + \frac{1}{2}\log n$. ⊓

Remarks:

- Simulating Batcher's sorting network on an ordinary sequential computer takes time $O(n \log^2 n)$. As we all know there are sequential sorting algorithms that sort in asymptotically optimal time $O(n \log n)$. The question whether there is a sorting network with depth $O(\log n)$ (oblivious, optimal, parallel!) remained open for a long time. In 1983, Ajtai, Komlos, and Szemeredi presented a much-celebrated $O(\log n)$ depth sorting network. (Unlike Batcher's sorting network the constant hidden in the big-O of the "AKS" sorting network is too large to be practical.)

- It was shown that Batcher's sorting network and similarly others can be simulated by a Butterfly network and other hypercubic networks.

- What if a sorting network is asynchronous?!? Clearly we cannot do sorting anymore. But check out the next section!

8.4 Counting Networks

Definition 8.18 (Balancer) *A balancer is an asynchronous flip-flop which forwards messages that arrive on the left side to the wires on the right, the first to the upper, the second to the lower, the third to the upper, and so on.*

Algorithm 8.19 ("Bitonic" Counting Network) *Take Batcher's bitonic sorting network of width w and replace all the comparators with balancers. When a node wants to count, it sends a message to an arbitrary input wire. The message is then routed through the network, following the rules of the asynchronous balancers. Each output wire is completed with a "mini-counter." The mini-counter of wire k replies the value "$k + i \cdot w$" to the initiator of the ith message it receives.*

Remarks:

- In Chapter 7 we have seen several ways to implement a general read-modify-write (RMW) operation. One particular RMW is "fetch-and-increment," where you ask for the value of a global variable and atomically increment it by one ("counting"). Naturally counting is a most important primitive in distributed computing, with many applications. At first it seems that such a RMW needs exclusive access to the variable (see Chapter 7). A counting network, however, is a distributed (decentralized) way to implement a fetch-and-increment!

- But first let's see why the counting network counts.

Definition 8.20 (Step Property) *A sequence $y_0, y_1, \ldots, y_{w-1}$ is said to have the* step property, *if $0 \leq y_i - y_j \leq 1$, for any $i < j$.*

Remark:

- If the output wires have the step property, then with r requests, exactly the values $1, \ldots, r$ will be assigned by the mini-counters. All we need to show is that the counting network has the step property. For that we need some additional facts...

Facts 8.21 *For a balancer, we denote the number of consumed messages on the ith input wire with x_i, $i = 0, 1$. Similarly, we denote the number of sent messages on the ith output wire with y_i, $i = 0, 1$. A balancer has these properties:*

(1) A balancer does not generate output-messages; that is, $x_0 + x_1 \geq y_0 + y_1$ in any state.

(2) Every incoming message is eventually forwarded. In other words, if we are in a quiescent state, then $x_0 + x_1 = y_0 + y_1$.

(3) The number of messages sent to the upper output wire is at most one higher than the number of messages sent to the lower output wire: in any state $y_0 = \lceil (y_0+y_1)/2 \rceil$ (thus $y_1 = \lfloor (y_0+y_1)/2 \rfloor$).

Facts 8.22 *If a sequence $y_0, y_1, \ldots, y_{w-1}$ has the step property,*

(1) then all its subsequences have the step property.

(2) then its even and odd subsequences satisfy

$$\sum_{i=0}^{w/2-1} y_{2i} = \left\lceil \frac{1}{2} \sum_{i=0}^{w-1} y_i \right\rceil \text{ and } \sum_{i=0}^{w/2-1} y_{2i+1} = \left\lfloor \frac{1}{2} \sum_{i=0}^{w-1} y_i \right\rfloor.$$

Facts 8.23 *If two sequences $x_0, x_1, \ldots, x_{w-1}$ and $y_0, y_1, \ldots, y_{w-1}$ have the step property,*

(1) and $\sum_{i=0}^{w-1} x_i = \sum_{i=0}^{w-1} y_i$, then $x_i = y_i$ for $i = 0, \ldots, w-1$.

(2) and $\sum_{i=0}^{w-1} x_i = \sum_{i=0}^{w-1} y_i + 1$, then there exists a unique j ($j = 0, 1, \ldots, w-1$) such that $x_j = y_j + 1$, and $x_i = y_i$ for $i = 0, \ldots, w-1$, $i \neq j$.

Remark:

- That's enough to prove that a Merger preserves the step property.

Lemma 8.24 *Let $M[w]$ be a Merger of width w. For $M[w]$ in a quiescent state, if the inputs $x_0, x_1, \ldots, x_{w/2-1}$ resp. $x_{w/2}, x_{w/2+1}, \ldots, x_{w-1}$ have the step property, then the output $y_0, y_1, \ldots, y_{w-1}$ has the step property.*

Proof. By induction on the width w.

For $w = 2$: $M[2]$ is a balancer and a balancer's output has the step property (Fact 8.21.3).

For $w > 2$: Let $z_0, \ldots, z_{w/2-1}$ resp. $z'_0, \ldots, z'_{w/2-1}$ be the output of the upper resp. lower $M[w/2]$ subnetwork. Since $x_0, x_1, \ldots, x_{w/2-1}$ and $x_{w/2}, x_{w/2+1}, \ldots, x_{w-1}$ both have the step property by assumption, their even and odd subsequences also have the step property (Fact 8.22.1). By induction hypothesis, the output of both $M[w/2]$ subnetworks have the step property. Let $Z := \sum_{i=0}^{w/2-1} z_i$ and $Z' := \sum_{i=0}^{w/2-1} z'_i$. From Fact 8.22.2 we conclude that $Z = \lceil \frac{1}{2} \sum_{i=0}^{w/2-1} x_i \rceil + \lfloor \frac{1}{2} \sum_{i=w/2}^{w-1} x_i \rfloor$ and

$Z' = \lfloor \frac{1}{2} \sum_{i=0}^{w/2-1} x_i \rfloor + \lceil \frac{1}{2} \sum_{i=w/2}^{w-1} x_i \rceil$. Since $\lceil a \rceil + \lfloor b \rfloor$ and $\lfloor a \rfloor + \lceil b \rceil$ differ by at most 1 we know that Z and Z' differ by at most 1.

If $Z = Z'$, Fact 8.23.1 implies that $z_i = z'_i$ for $i = 0, \ldots, w/2 - 1$. Therefore, the output of $M[w]$ is $y_i = z_{\lfloor i/2 \rfloor}$ for $i = 0, \ldots, w - 1$. Since $z_0, \ldots, z_{w/2-1}$ has the step property, so does the output of $M[w]$ and the Lemma follows.

If Z and Z' differ by 1, Fact 8.23.2 implies that $z_i = z'_i$ for $i = 0, \ldots, w/2 - 1$, except a unique j such that z_j and z'_j differ by only 1, for $j = 0, \ldots, w/2 - 1$. Let $l := min(z_j, z'_j)$. Then, the output y_i (with $i < 2j$) is $l + 1$. The output y_i (with $i > 2j + 1$) is l. The output y_{2j} and y_{2j+1} are balanced by the final balancer resulting in $y_{2j} = l + 1$ and $y_{2j+1} = l$. Therefore $M[w]$ preserves the step property. ⊓

A bitonic counting network is constructed to fulfill Lemma 8.24 – the final output comes from a Merger whose upper and lower inputs are recursively merged. Therefore, the following Theorem follows immediately.

Theorem 8.25 (Correctness) *In a quiescent state, the w output wires of a bitonic counting network of width w have the step property.*

Remark:

- Is every sorting networks also a counting network? No. But surprisingly, the other direction is true!

Theorem 8.26 (Counting vs. Sorting) *The isomorphic network of a counting network is a sorting network but not vice versa.*

Proof. There are sorting networks that are not counting networks (e.g. odd/even sort, or insertion sort)

For the other direction, let C be a counting network and $I(C)$ be the isomorphic network, where every balancer is replaced by a comparator. Let $I(C)$ have an arbitrary input of 0's and 1's; that is, some of the input wires have a 0, all others have a 1. There is a message at C's ith input wire if and only if $I(C)$'s i input wire is 0. Since C is a counting network, all messages are routed to the upper output wires. $I(C)$ is isomorphic to C, therefore a comparator in $I(C)$ will receive a 0 on its upper (lower) wire if and only if the corresponding balancer receives a message on its upper (lower) wire. Using an inductive argument, the 0's and 1's will be routed through $I(C)$ such that all 0's exit the network on the upper wires whereas all 1's exit the network on the lower wires. Applying Lemma 8.3 shows that $I(C)$ is a sorting network. ⊓

Remark:

- We claimed that the counting network is correct. However, it is only correct in a quiescent state.

Definition 8.27 (Linearizable) *A system is linearizable if the order of the values assigned reflects the real-time order in which they were requested. More formally, if there is a pair of operations o_1, o_2, where operation o_1 terminates before operation o_2 starts, and the logical order is "o_2 before o_1," then a distributed system is not linearizable.*

Lemma 8.28 (Linearizability) *The bitonic counting network is not linearizable.*

Proof. Please consider the bitonic counting network with width 4 in Figure 1: Assume that two inc operations were initiated and the corresponding messages entered the network on wire 0 and 2 (both in light grey color). After having passed the second resp. the first balancer, these traversing messages "fall asleep"; In other words, both messages take unusually long time before they are received by the next balancer. Since we are in an asynchronous setting, this may be the case.

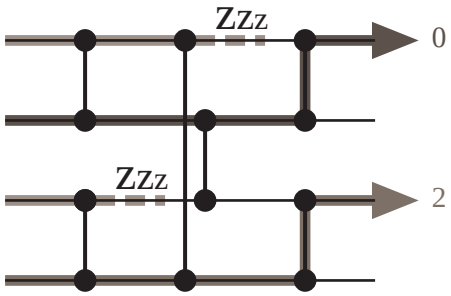

Figure 1: Linearizability Counter Example

In the meantime, another inc operation (medium grey) is initiated and enters the network on the bottom wire. The message leaves the network on wire 2, and the inc operation is completed.

Strictly afterwards, another inc operation (dark grey) is initiated and enters the network on wire 1. After having passed all balancers, the message will leave the network wire 0. Finally (and not depicted in figure 1), the two light grey messages reach the next balancer and will eventually leave the network on wires 1 resp. 3. Because the dark grey and the medium grey operation do conflict with Definition 8.27, the bitonic counting network is not linearizable. ⊓

Remark:

- Note that the example in Figure 1 behaves correctly in the quiescent state: Finally, exactly the values $0, 1, 2, 3$ are allotted.
- Several researchers investigated whether linearizable counting is possible at all. It was shown that linearizability comes at a high price (the depth grows linearly with the width).