Math 262A Lecture Notes - Sorting Networks

Lecturer: Sam Buss
Scribe: Stefan Schneider

December 15, 2013

It is well known that sorting algorithms such as MergeSort and QuickSort can sort n elements in $O(n \log n)$ comparisons. Both MergeSort and QuickSort have the nice property that they can be implemented with only constant space overhead.

We can aim for the even higher goal of a sorting algorithm requiring no space overhead at all. In particular, we want an algorithm that is *oblivious*, the algorithm compares the same sequence of indices independent of the result of previous comparisons. Note that the previous example of MergeSort is not oblivious, since the merge operation requires us to keep two pointers and moving one or the other depending on a comparison.

The ambitious goal we set ourselves is to design an oblivious algorithm that not only requires $O(n \log n)$ comparisons, but also only needs depth $O(\log n)$. At any layer we can make at most $O\left(\frac{n}{2}\right)$ comparisons in parallel, hence $O(\log n)$ is a lower bound for the depth.

The computation model we discuss are *sorting networks* or *comparator networks*. The inputs are n elements from some linearly ordered domain (not necessarily numbers), and the basic building block is a *comparator gate*. A comparator gate takes two elements a_0 and a_1 from the domain as inputs and outputs $\min\{a_0, a_1\}$ as its first output and $\max\{a_0, a_1\}$ as its second output. Hence a comparator gate represents a conditional swap of two elements. Figure 1 represents a comparator gate.

If the input is Boolean, a comparator gate can be implemented with and AND-gate for $\min\{a_0, a_1\}$ and an OR-gate for $\max\{a_0, a_1\}$. Hence any depth d sorting network immediately implies a depth d monotone circuit for Th_k^n.

We discuss two sorting networks to sort n elements. The first algorithm is due to Batcher [1], and it requires depth $O((\log n)^2)$ and total size $O(n(\log n)^2)$. The second algorithm is due to Ajtai, Komlós, and Szemerédi [2] and was later simplified by Paterson [3] and Seiferas [4]. This algorithm does achieve our goal of depth $O(\log n)$. However, the leading constants are so forbiddingly large, that BatcherSort performs faster unless the number of inputs in larger than the number of atoms in the universe. It is still an open problem to design an $O(\log n)$ depth sorting network that improves

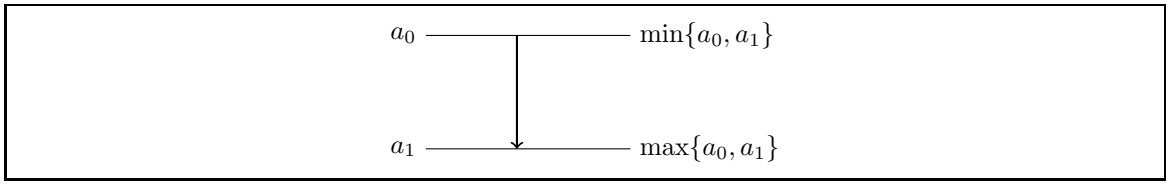

Figure 1: A single comparator gate

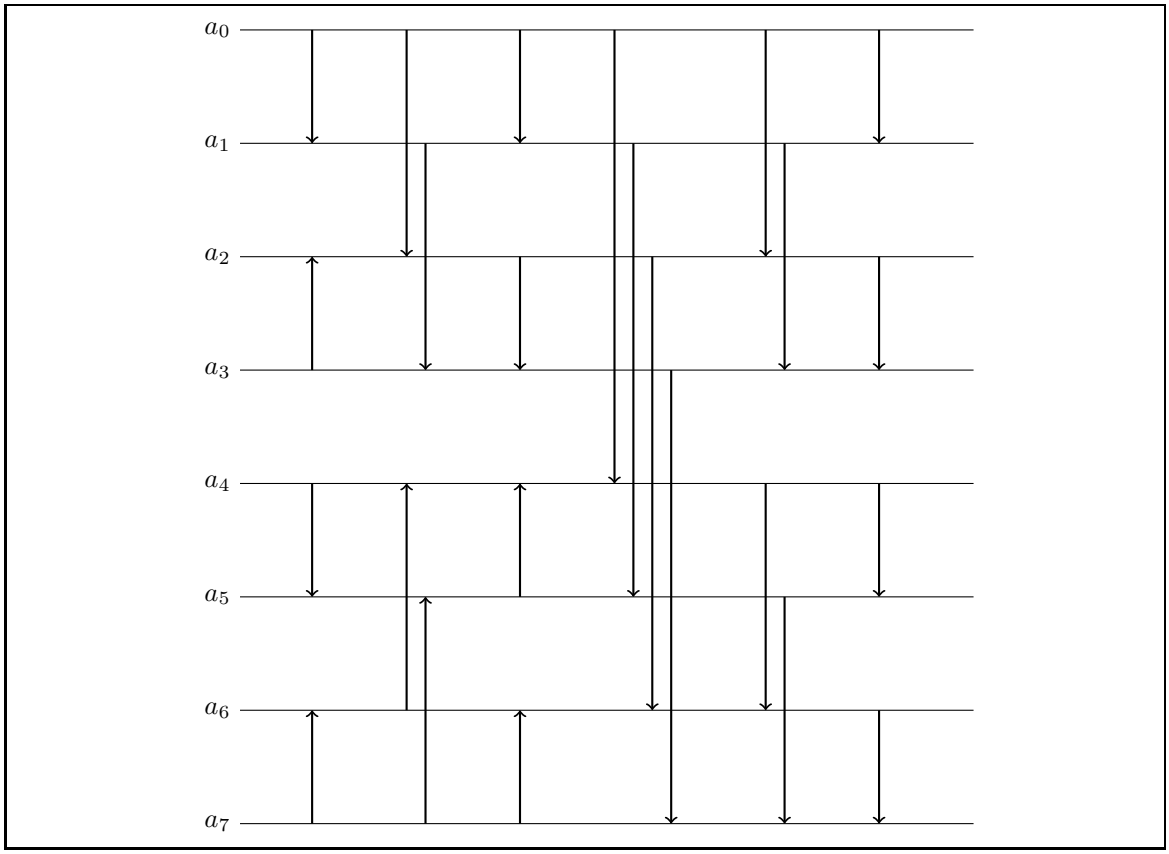

Figure 2: BatcherSort for 8 inputs

over Batcher's sorting network for a practical number of inputs.

BatcherSort

Figure 2 contains the whole sorting network for BatcherSort on 8 inputs. Note that the number of layers in this network is $6 = 3 + 2 + 1 = O((\log n)^2)$ where $3 = \log_2(8)$. The size of the network is $\frac{8}{2} \cdot 6 = O(n(\log n)^2)$

Definition 1. *Let $A = \langle a_0, \ldots, a_{n-1} \rangle$ be a sequence of length n.*

A rotation *of A is any sequence $\langle a_{l \mod n}, a_{l+1 \mod n}, \ldots, a_{l+(n-1) \mod n} \rangle$.*

The sequence A is vee-like *iff there is a k with $0 \leq k \leq n$ such that for all $i < k$ we have $a_i \geq a_{i+1}$ and for all $i \geq k$ we have $a_i \leq a_{i+1}$. The sequence is* wedge-like *if $a_i \leq a_{i+1}$ for $i < k$ and $a_i \geq a_{i+1}$ for $i \geq k$.*

A sequence A is bitonic *iff it has a rotation which is vee-like, or, equivalently, a rotation which is wedge-like.*

The two equivalent definitions of a bitonic sequence follow from the fact a sequence A which is vee-like with parameter k, the sequence $\langle a_{k \mod n}, a_{k+1 \mod n}, \ldots, a_{k+(n-1) \mod n} \rangle$ is wedge-like.

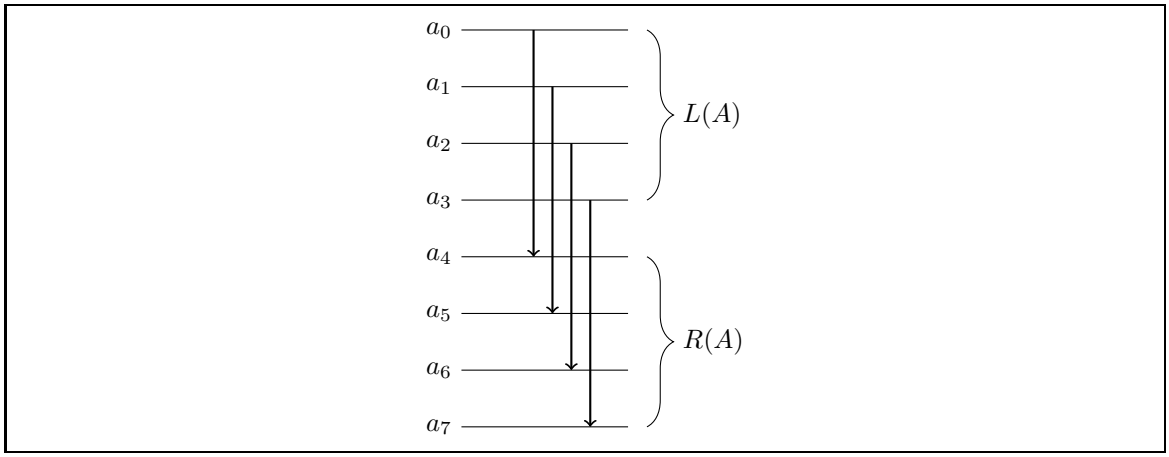

Figure 3: A single layer sorting network to compute L(A) and R(A)

Definition 2. *Suppose $A = \langle a_0, \ldots, a_{2m-1} \rangle$ is a sequence of even length $n = 2m$. Then*

$$L(A) = \langle \min\{a_0, a_m\}, \min\{a_1, a_{m+1}\}, \ldots \min\{a_{m-1}, a_{2m-1}\} \rangle$$
$$R(A) = \langle \max\{a_0, a_m\}, \max\{a_1, a_{m+1}\}, \ldots \max\{a_{m-1}, a_{2m-1}\} \rangle$$

Remark 1. *Figure 3 shows that we can compute both $L(A)$ and $R(A)$ (at the same time) with a single layer of comparator gates.*

Theorem 1. *If A is bitonic, then $L(A)$ and $R(A)$ are bitonic and for each $x \in L(A)$ and $y \in R(A)$ we have $x \leq y$.*

Proof. We can assume w.l.o.g. that A is vee-like by the following argument: To form $L(A)$ and $R(A)$ we compute $\min\{a_i, a_{i+\frac{n}{2}}\}$ and $\max\{a_i, a_{i+\frac{n}{2}}\}$ for all $0 \leq i \leq \frac{n}{2} - 1$. For a rotation A' of A with offset l, computing $L(A')$ and $R(A')$ requires us to compute $\min\{a_{i+l \mod n}, a_{i+l+\frac{n}{2} \mod n}\}$ and $\max\{a_{i+l \mod n}, a_{i+l+\frac{n}{2} \mod n}\}$ for $0 \leq i \leq \frac{n}{2} - 1$. However, these are the same comparisons.

Since A is w.l.o.g. vee-like, there is some value k such that for all $i < k$ we have $a_i \geq a_{i+1}$ and for all $i \geq k$ we have $a_i \leq a_{i+1}$. We assume without loss of generality that $k \leq \frac{n}{2}$. Otherwise, we can do the same proof on the reverse of A.

Since $a_k \leq a_{\frac{n}{2}}$, there must be some smallest s such that for all $s \leq s' \leq \frac{n}{2} - 1$ we have $a_{s'} \leq a_{\frac{n}{2}+s'}$ and for all $0 \leq s' \leq s$ we have $a_{s'} \geq a_{\frac{n}{2}+s'}$. Therefore we get

$$L(A) = \langle a_{\frac{n}{2}}, \ldots, a_{\frac{n}{2}+s-1}, a_s, \ldots, a_{\frac{n}{2}-1} \rangle$$

and

$$R(A) = \langle a_0, \ldots, a_{s-1}, a_{\frac{n}{2}+s}, \ldots, a_{n-1} \rangle$$

If $s \leq k$, then $R(A)$ is vee-like as $a_i \geq a_{i+1}$ for $i \leq s$ and $a_{\frac{n}{2}+i} \leq a_{\frac{n}{2}+i+1}$ for all $i \geq 0$. The sequence $L(A)$ is bitonic because the rotation $\langle a_k, \ldots, a_{\frac{n}{2}+s-1}, a_s, \ldots, a_{k-1} \rangle$ is wedge-like. For $s \geq k$ the same holds with a symmetric argument.

Figure 4 gives a pictorial representation of the proof. The second part of our claim is also immediately evident from the picture. □

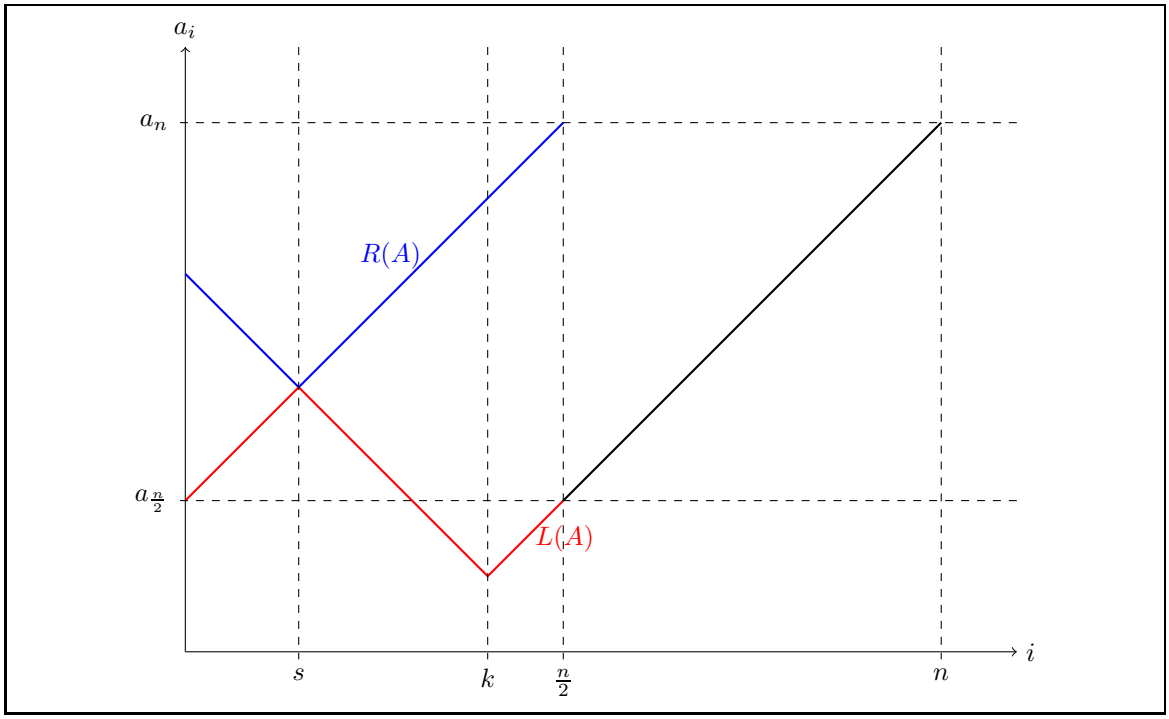

Figure 4: Pictorial representation of Theorem 1

Suppose we want to sort a bitonic sequence of length $n = 2^j$ for some j. Note that if n is not a power of 2, we can simply pad the input, which will double the input in the worst case. If $j = 0$, we are done, as the input is already sorted. For $j \geq 1$, we compute $L(A)$ and $R(A)$ with a single layer of comparator gates and then recursively sort the two subproblems independently of each other. Hence we can sort a bitonic sequence A with a network of only j levels.

In the general case, we need to first make the input sequence bitonic. We use the fact that if A_1 and A_2 are both sorted sequences, then $A_1 \circ \text{reversal}(A_2)$ is bitonic.

We can view the initial input as a concatenation of $\frac{n}{2}$ bitonic subsequences of length 2. Then as long as we have $\frac{n}{2^i} > 1$ subsequences of length 2^i, sort all of the sequences using the subroutine for bitonic inputs and reverse the order of every alternating subsequence. As a result, we get $\frac{n}{2^{i+1}}$ bitonic sequences of length 2^{i+1}. Once the whole sequence is bitonic, we can simply sort it using the subroutine.

Since the subroutine can work in parallel when operating on different parts of the input, the total depth of this sorting network is

$$\sum_{i=1}^{j} i = \frac{j(j+1)}{2} = O((\log n)^2)$$

where the last equality follows from $j = \log n$.

Outline of the AKS Sorting Network

In this section we outline the depth $O(\log n)$ sorting network due to Ajtai, Komlós, and Szemerédi [2]. The presentation follows the simplified proof by Seiferas [4]. The full proof can be found in the follow-up lecture notes. This section limits itself to an outline.

An important building block of the AKS network are approximate sorters.

Definition 3. *Let $\varepsilon > 0$ and $0 \leq \lambda \leq \frac{1}{2}$. A sorting network is an* ε-approximate λ-comparator *on n inputs iff for all $\lambda' \leq \lambda$ at most $\varepsilon\lambda' n$ of the $\lfloor \lambda' n \rfloor$ many smallest inputs are not among the leftmost $\lfloor \lambda n \rfloor$ many outputs. Dually, at most $\varepsilon\lambda' n$ of the $\lfloor \lambda' n \rfloor$ many largest inputs are not among the rightmost $\lfloor \lambda n \rfloor$ many outputs.*

We call an ε-approximate $\frac{1}{2}$-comparator an ε-halver.

An ε-approximate λ-comparator can act as an approximate sorter. The output is not completely sorted, but the smallest elements are likely to appear among the leftmost elements in the output, and the largest elements are likely to be among the rightmost elements.

The key lemma is that such approximate sorters exist.

Lemma 1. *Let $\varepsilon > 0$ and $0 \leq \lambda \leq \frac{1}{2}$. There is a constant depth sorting network that is both an ε-approximate λ-comparator and an ε-halver.*

For the proof, we will choose $\varepsilon = \lambda = \frac{1}{99}$.

If we had a 0-halver, a sorting network that splits the inputs into the smaller half and the larger half without any error, then we could easily build a sorter by simply recursing on the two halves. We can use our ε-halver with the same intent. However, there will be a small fraction of elements that we miscategorize. A crucial observation here is that miscategorized elements are among the smallest or largest elements in the subproblem, hence after applying our approximate sorter to the subproblem, its property as an ε-approximate λ-comparator means that the miscategorized elements are likely among the leftmost or rightmost outputs. The idea is then to send the leftmost and rightmost elements back to the previous iteration to be reconsidered. The full algorithm then repeatedly sends, for every subproblem, a large fraction of its inputs to its own subproblems and a small fraction back to its superproblem to be reconsidered. The key part of the proof is to show that after $O(\log n)$ iterations, this process not only results in all elements being categorized into constant size subproblems, but also that all miscategorizations have been corrected. Then, as a last step, we can use BatcherSort to sort the remaining, constant size subproblems.

References

[1] Kenneth E Batcher, *Sorting networks and their applications*. Proceedings of the April 30–May 2, 1968, spring joint computer conference, 1968.

[2] Miklor Ajtai, Jáno Komlós, Endre Szemerédi, *An $O(n \log n)$ sorting network*. STOC, 1983.

[3] MS Paterson, *Improved sorting networks with $O(\log n)$ depth*. Algorithmica, 1990.

[4] J Seiferas, *Sorting networks of logarithmic depth, further simplified*. Algorithmica, 2009.