Lecture 10: Sorting in (sub)linear time

1. Comparison networks. Sorting networks
2. Counting-based sorting

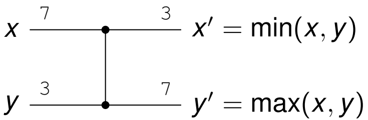
November 29, 2018

Comparison networks
Wires and comparators

A comparison-based model of computation (it can perform only comparisons) in which many comparison operations can be performed simultaneously.

- It is made of wires and comparators
- **comparator** = device with two inputs, x and y, and two outputs, x' and y', that performs the following function: $x' = \min(x, y), y' = \max(x, y)$.
  It is depicted by a vertical line segment.
- **wire**: transmits a value from place to place.
  It is depicted by a horizontal line segment.
- We assume that each comparator operates in $O(1)$ time.

Pictorial representation of a comparator:

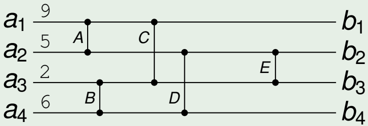


Comparison networks
Assumptions

A comparison network has n input wires $a_1, a_2, \ldots, a_n$, and n output wires $b_1, b_2, \ldots, b_n$ which produce the results computed by the comparison network.

- the input sequence is $\langle a_1, a_2, \ldots, a_n \rangle$, and the output sequence is $\langle b_1, b_2, \ldots, b_n \rangle$,

Example (a 4-input, 4-output comparison network)

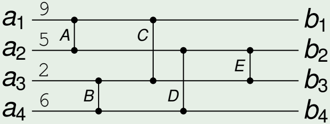


- Each comparator produces its output values only when both of its input values are available to it.
- **Main requirement:** the graph of interconnections must be acyclic $\Rightarrow$ we can draw the network with inputs on the left, and outputs on the right (see next slide)

Comparison networks

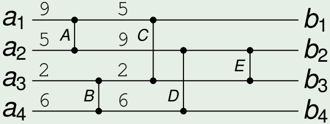
Example (a 4-input, 4-output comparison network)

at time 0



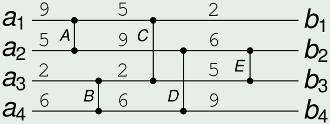
depth:

at time 1



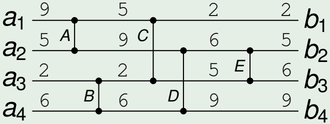
depth: 1 1

at time 2



depth: 1 1 2 2

at time 3



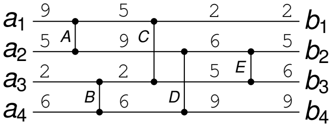
depth: 1 1 2 2 3

- The depth of a wire in a comparison network is defined recursively as follows:
  - an input wire has depth 0
  - If the input wires x, y of a comparator have depths d_x, d_y, then its output wires have depth $\max(d_x, d_y) + 1$.
    This is also the depth of the comparator.
- The depth of a comparison network is the maximum depth of an output wire.
- A sorting network is a comparison network for which the output sequence is monotonically increasing (that is, $b_1 \leq b_2 \leq \ldots \leq b_n$) for every input sequence $a_1, a_2, \ldots, a_n$.
  - The comparison network from the previous example is a sorting network: it has depth 3 $\Rightarrow$ it sorts any sequence $a = \langle a_1, a_2, a_3, a_4 \rangle$ in 3 steps.

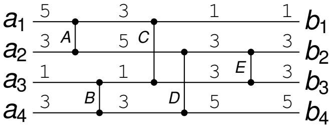
Fact: If a comparison network transforms the input sequence $a = \langle a_1, a_2, \ldots, a_n \rangle$ into the output sequence $b = \langle b_1, b_2, \ldots, b_n \rangle$, then for any monotonically increasing function f, the network transforms the input sequence $f(a) = \langle f(a_1), f(a_2), \ldots,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

PROOF HINT: We can prove by induction on the depth of each wire, the following stronger result: if a wire assumes the value a_i when the input sequence a is applied to the network, then it assumes the value $f(a_i)$ when the input sequence $f(a)$ is applied.

- A sorting network with input sequence $\langle 9, 5, 2, 6 \rangle$:



- The same sorting network with function $f(x) = \lceil x/2 \rceil$ applied to the inputs



Fact: If a comparison network with n inputs sorts all possible 2^n sequences of 0's and 1's correctly, then it sorts all sequences of arbitrary numbers correcty.

PROOF: By contradiction: assume there is a sequence of numbers $a = \langle a_1, a_2, \ldots, a_n \rangle$ that gets sorted incorrectly. This mean, there exists $a_i < a_j$ but the network places a_j before a_i in the output sequence $b = \langle b_1, b_2, \ldots, b_n \rangle$. Consider the monotonic function

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

Then $f(a)$ is a sequences of 0's and 1's that is sorted incorrectly by the comparison network $\Rightarrow$ contradiction.

Bitonic sequences

A bitonic sequence is a sequence of numbers that monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

- Examples: $\langle 1, 4, 6, 8, 3, 2 \rangle$, $\langle 6, 9, 4, 2, 3, 5 \rangle$, and $\langle 9, 8, 3, 2, 4, 6 \rangle$ are bitonic sequences.
- Remarks:
  - Every sequence of length 1 or 2 is bitonic.
  - The zero-one sequences that are bitonic are of the form $0^i 1^j 0^k$ or of the form $1^i 0^j 1^k$ for some $i, j, k \geq 0$.

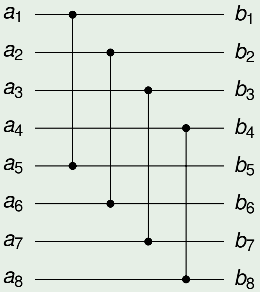
Half cleaners

A half-cleaner for a sequences of an even length n $a = \langle a_1, a_2, \ldots, a_n \rangle$ is a comparison network of depth 1 in which input line i is compared with line $i + n/2$ for $i = 1, 2, \ldots, n/2$.

- We denote the half cleaner for sequences of n numbers with HALF-CLEANER[n].

Example

HALF-CLEANER[8] is shown below:



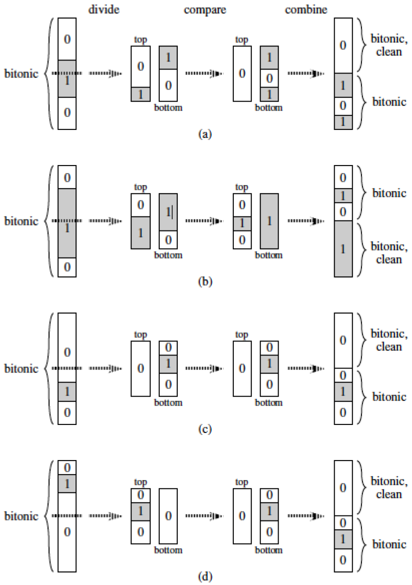
Fact: If n is even, $a = \langle a_1, a_2, \ldots, a_n \rangle$ is a bitonic sequence, and $b = \langle b_1, b_2, \ldots, b_n \rangle$ is the output of HALF-CLEANER[n] for input sequence a, then:

1. both the top half $\langle b_1, b_2, \ldots, b_{n_2} \rangle$ and the bottom half $\langle b_{n/2+1}, b_{n/2+2}, \ldots, b_n \rangle$ are bitonic.
2. every element in the top half is at least as small as every element of the bottom: $b_i \leq b_j$ whenever $i \leq n/2 < j$.
3. at least one half is clean (that is, consisting of only one number, either 0 or 1).

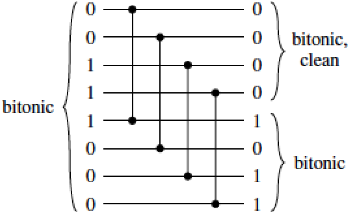
PROOF SKETCH: see next slide.

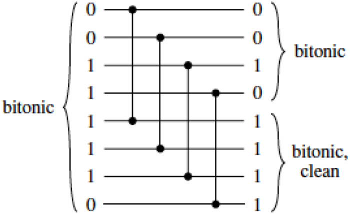
Half cleaners

Proof sketch of the remarkable property

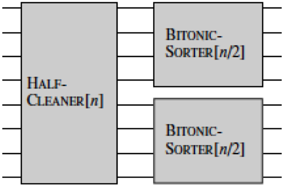


Half cleaners
Remarkable property: illustrated example

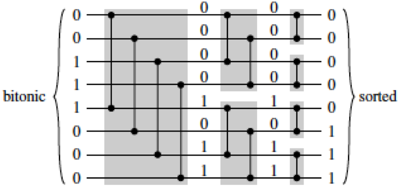




BITONICSORTER[n] is the comparison network with the following recursive structure:



For example, the complete picture of BITONICSORTER[8] is



From the recursive structure of BITONICSORTER[n], we learn that its depth $D(n)$ satisfies the recursive equation

$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ D(n/2) + 1 & \text{if } n = 2^k \text{ and } k \geq 1. \end{cases}$$

whose solution of $D(n) = \log_2 n$

$\Rightarrow$ BITONICSORTER[n] sorts bitonic sequences in $\log_2 n$ time.

Given two sorted input sequences $a = \langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $b = \langle a_{n/2+1}, \ldots, a_{n-1}, a_n \rangle$, where $n = 2^k$ for some $k \geq 1$

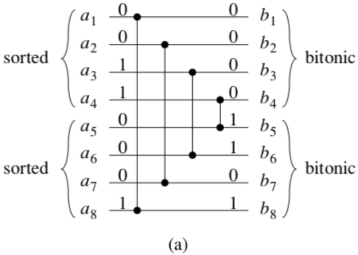
Define a comparison network MERGER[n] that merges a and b into one sorted output sequence.

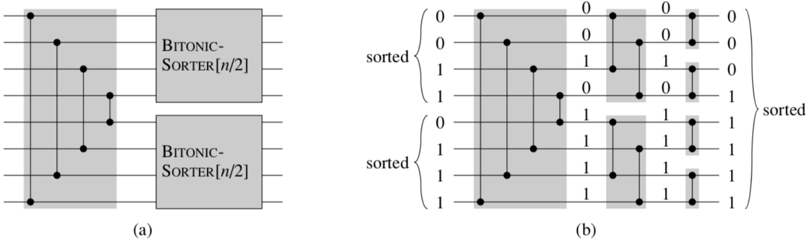
Remarks

- The sequence $c = \langle \underbrace{a_1, a_2, \ldots, a_{n/2}}_{a}, \underbrace{a_n, a_{n-1} \ldots, a_1}_{\text{reverse of } b} \rangle$ is bitonic $\Rightarrow$ we can use BITONICSORTER[n] to sort it.
- We can reconfigure easily BITONICSORTER[n] for input $\langle a_1, a_2, \ldots, a_n \rangle$ to behave like BITONICSORTER[n] for input c (see next slide).

MERGER[n] versus HALF-CLEANER[n]

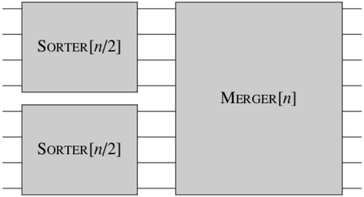
Structural comparison of their first stage for $n = 8$



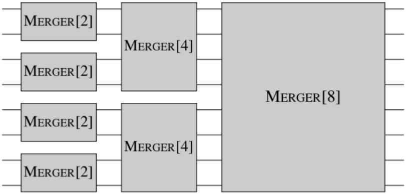


REMARK: The depth of MERGER[n] is the same as that of BITONIC-SORTER[n], that is, $\log_2 n$.

Recursive structure of SORTER[n] for $n = 2^k$

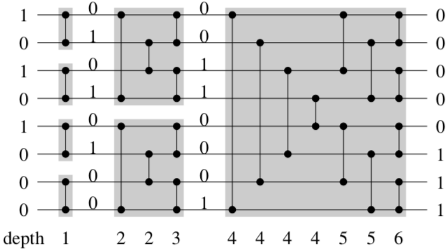


Special case, when $n = 8$



Sorting networks

Example: SORTER[8] looks as follows:



The depth of SORTER[n] satisfies the recursive relation

$$D[n] = \begin{cases} 0 & \text{if } n = 1 \\ D(n/2) + \log_2 n & \text{if } n = 2^k \text{ and } k \geq 1. \end{cases}$$

whose solution is $D(n) = \Theta(\log_2^2 n) < \Theta(n)$.

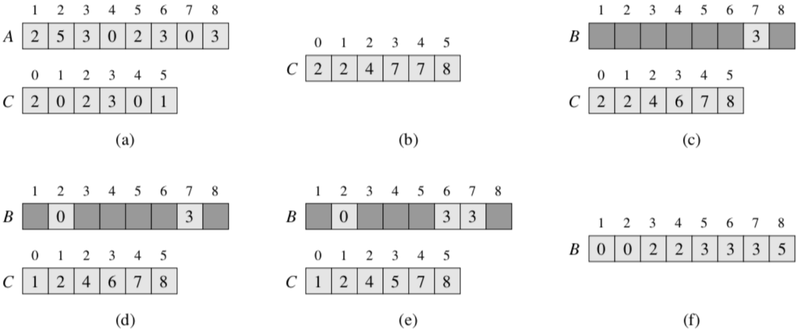
- We wish to sort an array $A[1..n]$ of integers, when we know that $0 \leq A[i] \leq k$ for all $1 \leq i \leq n$.
- **Known result:** this problem can be solved with counting sort in time $\Theta(n+k)$, which becomes $\Theta(n)$ when $k = O(n)$.
  - **Main idea:** for each input element x, count the number of elements less than x; this information can be used to place x directly into its position in the output array.

Counting sort
Pseudocode

```
COUNTING-SORT(A, B, k)
for i ← 0 to k
      do C[i] ← 0
for j ← 1 to length[A]
      do C[A[j]] ← C[A[j]] + 1
▷ C[i] now contains the number of elements equal to i.
for i ← 1 to k
      do C[i] ← C[i] + C[i − 1]
▷ C[i] now contains the number of elements less than or equal to i.
for j ← length[A] downto 1
      do B[C[A[j]]] ← A[j]
         C[A[j]] ← C[A[j]] − 1
```

Counting sort

Illustrated example for an input array $A[1..8]$



(a) Arrays A and C after execution of line 4.

(b) Array C after execution of line 7.

(c),(d)(e) Arrays B and C after 1,2, and 3 iterations of the loop in lines 9-11.

(f) The final sorted output array B.

References

T.H. Cormen *et al.*, Introduction to algorithms. Second Edition. The MIT Press. 2002.

- Chapter 27: Sorting networks.
- Chapter 8: Sorting in linear time.