Sorting networks

Lecture 20

November 4, 2014

Model of Computation

1. Q: Perform a computational task considerably faster by using a different architecture? Yep.
2. ***Spaghetti sort***!

Model of Computation

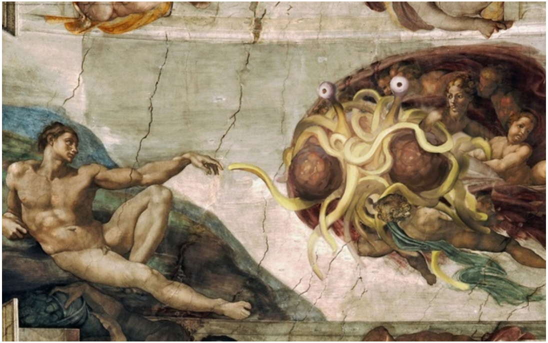
1. Q: Perform a computational task considerably faster by using a different architecture? Yep.
2. ***Spaghetti sort***!

Model of Computation

1. Q: Perform a computational task considerably faster by using a different architecture? Yep.
2. ***Spaghetti sort***!

Spaghetti

Spaghetti



Pastafarianism

Spaghetti

Spaghetti

Spaghetti

The spaghetti tree hoax was a three-minute hoax report broadcast on April Fools' Day 1957 by the BBC current-affairs programme Panorama, purportedly showing a family in southern Switzerland harvesting spaghetti from the family "spaghetti tree". At the time spaghetti was relatively little-known in the UK, so that many Britons were unaware that spaghetti is made from wheat flour and water; a number of viewers afterwards contacted the BBC for advice on growing their own spaghetti trees. Decades later CNN called this broadcast "the biggest hoax that any reputable news establishment ever pulled."

Spaghetti sort

1. Input: $S = \{s_1, \ldots, s_n\} \subseteq [1, 2]$.
2. Have much Spaghetti (this are longish and very narrow tubes of pasta).
3. cut ith piece to be of length s_i, for $i = 1, \ldots, n$.
4. take all these pieces of pasta in your hand..
5. make them stand up vertically, with their bottom end lying on a horizontal surface
6. lower your handle till it hit the first (i.e., tallest) piece of pasta.
7. Take it out, measure it height, write down its number
8. and continue in this fashion till done.
9. Linear time sorting algorithm.
10. ...but sorting takes $\Omega(n \log n)$ time.

Spaghetti sort

1. Input: $S = \{s_1, \ldots, s_n\} \subseteq [1, 2]$.
2. Have much Spaghetti (this are longish and very narrow tubes of pasta).
3. cut ith piece to be of length s_i, for $i = 1, \ldots, n$.
4. take all these pieces of pasta in your hand..
5. make them stand up vertically, with their bottom end lying on a horizontal surface
6. lower your handle till it hit the first (i.e., tallest) piece of pasta.
7. Take it out, measure it height, write down its number
8. and continue in this fashion till done.
9. Linear time sorting algorithm.
10. ...but sorting takes $\Omega(n \log n)$ time.

Spaghetti sort

1. Input: $S = \{s_1, \ldots, s_n\} \subseteq [1, 2]$.
2. Have much Spaghetti (this are longish and very narrow tubes of pasta).
3. cut ith piece to be of length s_i, for $i = 1, \ldots, n$.
4. take all these pieces of pasta in your hand..
5. make them stand up vertically, with their bottom end lying on a horizontal surface
6. lower your handle till it hit the first (i.e., tallest) piece of pasta.
7. Take it out, measure it height, write down its number
8. and continue in this fashion till done.
9. Linear time sorting algorithm.
10. ...but sorting takes $\Omega(n \log n)$ time.

Spaghetti sort

1. Input: $S = \{s_1, \ldots, s_n\} \subseteq [1, 2]$.
2. Have much Spaghetti (this are longish and very narrow tubes of pasta).
3. cut ith piece to be of length s_i, for $i = 1, \ldots, n$.
4. take all these pieces of pasta in your hand..
5. make them stand up vertically, with their bottom end lying on a horizontal surface
6. lower your handle till it hit the first (i.e., tallest) piece of pasta.
7. Take it out, measure it height, write down its number
8. and continue in this fashion till done.
9. Linear time sorting algorithm.
10. ...but sorting takes $\Omega(n \log n)$ time.

Spaghetti sort

1. Input: $S = \{s_1, \ldots, s_n\} \subseteq [1, 2]$.
2. Have much Spaghetti (this are longish and very narrow tubes of pasta).
3. cut ith piece to be of length s_i, for $i = 1, \ldots, n$.
4. take all these pieces of pasta in your hand..
5. make them stand up vertically, with their bottom end lying on a horizontal surface
6. lower your handle till it hit the first (i.e., tallest) piece of pasta.
7. Take it out, measure it height, write down its number
8. and continue in this fashion till done.
9. Linear time sorting algorithm.
10. ...but sorting takes $\Omega(n \log n)$ time.

Spaghetti sort

1. Input: $S = \{s_1, \ldots, s_n\} \subseteq [1, 2]$.
2. Have much Spaghetti (this are longish and very narrow tubes of pasta).
3. cut ith piece to be of length s_i, for $i = 1, \ldots, n$.
4. take all these pieces of pasta in your hand..
5. make them stand up vertically, with their bottom end lying on a horizontal surface
6. lower your handle till it hit the first (i.e., tallest) piece of pasta.
7. Take it out, measure it height, write down its number
8. and continue in this fashion till done.
9. Linear time sorting algorithm.
10. ...but sorting takes $\Omega(n \log n)$ time.

Spaghetti sort

1. Input: $S = \{s_1, \ldots, s_n\} \subseteq [1, 2]$.
2. Have much Spaghetti (this are longish and very narrow tubes of pasta).
3. cut ith piece to be of length s_i, for $i = 1, \ldots, n$.
4. take all these pieces of pasta in your hand..
5. make them stand up vertically, with their bottom end lying on a horizontal surface
6. lower your handle till it hit the first (i.e., tallest) piece of pasta.
7. Take it out, measure it height, write down its number
8. and continue in this fashion till done.
9. Linear time sorting algorithm.
10. ...but sorting takes $\Omega(n \log n)$ time.

Spaghetti sort

1. Input: $S = \{s_1, \ldots, s_n\} \subseteq [1, 2]$.
2. Have much Spaghetti (this are longish and very narrow tubes of pasta).
3. cut ith piece to be of length s_i, for $i = 1, \ldots, n$.
4. take all these pieces of pasta in your hand..
5. make them stand up vertically, with their bottom end lying on a horizontal surface
6. lower your handle till it hit the first (i.e., tallest) piece of pasta.
7. Take it out, measure it height, write down its number
8. and continue in this fashion till done.
9. Linear time sorting algorithm.
10. ...but sorting takes $\Omega(n \log n)$ time.

Spaghetti sort

1. Input: $S = \{s_1, \ldots, s_n\} \subseteq [1, 2]$.
2. Have much Spaghetti (this are longish and very narrow tubes of pasta).
3. cut ith piece to be of length s_i, for $i = 1, \ldots, n$.
4. take all these pieces of pasta in your hand..
5. make them stand up vertically, with their bottom end lying on a horizontal surface
6. lower your handle till it hit the first (i.e., tallest) piece of pasta.
7. Take it out, measure it height, write down its number
8. and continue in this fashion till done.
9. Linear time sorting algorithm.
10. ...but sorting takes $\Omega(n \log n)$ time.

Spaghetti sort

1. Input: $S = \{s_1, \ldots, s_n\} \subseteq [1, 2]$.
2. Have much Spaghetti (this are longish and very narrow tubes of pasta).
3. cut ith piece to be of length s_i, for $i = 1, \ldots, n$.
4. take all these pieces of pasta in your hand..
5. make them stand up vertically, with their bottom end lying on a horizontal surface
6. lower your handle till it hit the first (i.e., tallest) piece of pasta.
7. Take it out, measure it height, write down its number
8. and continue in this fashion till done.
9. Linear time sorting algorithm.
10. ...but sorting takes $\Omega(n \log n)$ time.

What is going on?

1. Faster algorithm achieved by changing the computation model.
2. allowed new "strange" operations
   (cutting a piece of pasta into a certain length, picking the longest one in constant time, and measuring the length of a pasta piece in constant time)
3. Using these operations we can sort in linear time.
4. So, are there other useful computation models?

What is going on?

1. Faster algorithm achieved by changing the computation model.
2. allowed new "strange" operations
   (cutting a piece of pasta into a certain length, picking the longest one in constant time, and measuring the length of a pasta piece in constant time)
3. Using these operations we can sort in linear time.
4. So, are there other useful computation models?

What is going on?

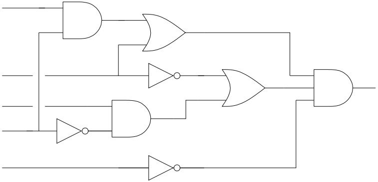
1. Faster algorithm achieved by changing the computation model.
2. allowed new "strange" operations
(cutting a piece of pasta into a certain length, picking the longest one in constant time, and measuring the length of a pasta piece in constant time)
3. Using these operations we can sort in linear time.
4. So, are there other useful computation models?

What is going on?

1. Faster algorithm achieved by changing the computation model.
2. allowed new "strange" operations
   (cutting a piece of pasta into a certain length, picking the longest one in constant time, and measuring the length of a pasta piece in constant time)
3. Using these operations we can sort in linear time.
4. So, are there other useful computation models?

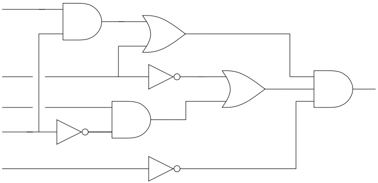
What is going on?

1. Faster algorithm achieved by changing the computation model.
2. allowed new "strange" operations
   (cutting a piece of pasta into a certain length, picking the longest one in constant time, and measuring the length of a pasta piece in constant time)
3. Using these operations we can sort in linear time.
4. So, are there other useful computation models?



Circuits are fast...

1. Computing the following circuit naively takes

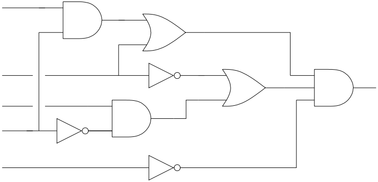


   8 units of time.

2. Use parallelism!
3. Circuits are really parallel...
4. Sorting numbers with circuits?
5. Q: Can sort in *sublinear* time by allowing parallel comparisons?

Circuits are fast...

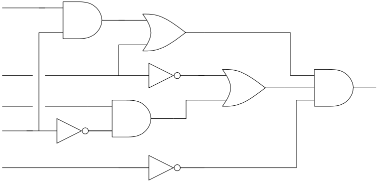
1. Computing the following circuit naively takes



   8 units of time.
2. Use parallelism!
3. Circuits are really parallel...
4. Sorting numbers with circuits?
5. Q: Can sort in *sublinear* time by allowing parallel comparisons?

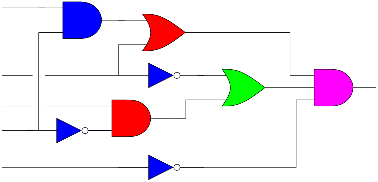
Circuits are fast...

1. Computing the following circuit naively takes



   8 units of time.

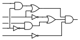
2. Use parallelism!



3. Circuits are really parallel...
4. Sorting numbers with circuits?

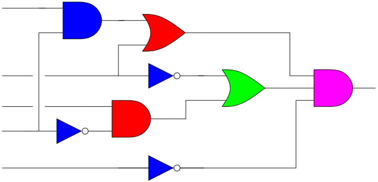
Circuits are fast...

1. Computing the following circuit naively takes



   8 units of time.

2. Use parallelism!

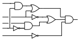


   4 time units!

3. Circuits are really parallel...
4. Sorting numbers with circuits?
5. Q: Can sort in *sublinear* time by allowing parallel comparisons?

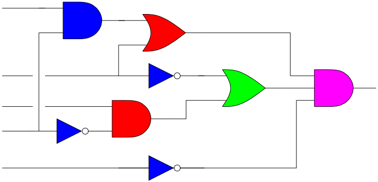
Circuits are fast...

1. Computing the following circuit naively takes



   8 units of time.

2. Use parallelism!

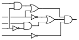


   4 time units!

3. Circuits are really parallel...
4. Sorting numbers with circuits?
5. Q: Can sort in *sublinear* time by allowing parallel comparisons?

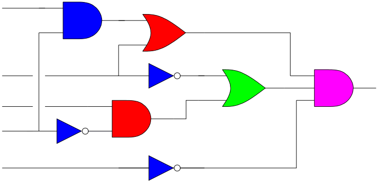
Circuits are fast...

1. Computing the following circuit naively takes



   8 units of time.

2. Use parallelism!

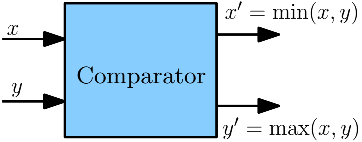


   4 time units!

3. Circuits are really parallel...
4. Sorting numbers with circuits?
5. Q: Can sort in *sublinear* time by allowing parallel comparisons?

Sorting with a circuit – a naive solution

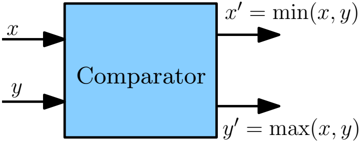
1. ***comparator*** gate:



2. Draw it as:

Sorting with a circuit – a naive solution

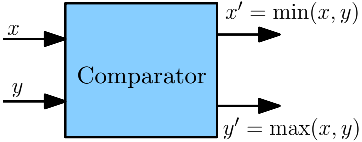
1. ***comparator*** gate:



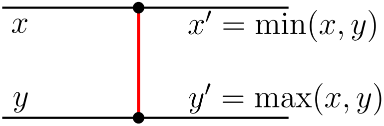
2. Draw it as:

Sorting with a circuit – a naive solution

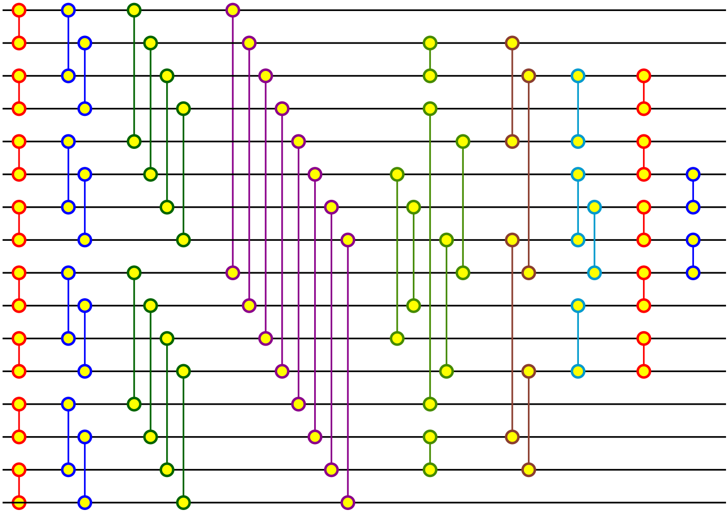
1. ***comparator*** gate:



2. Draw it as:

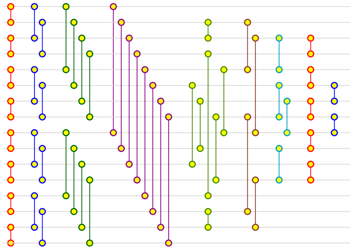


Sorting network - an example



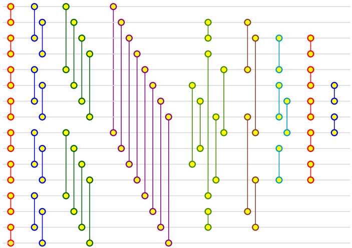
How to draw a circuit...

1. **wires**: horizontal lines
2. **gates**: vertical segments (i.e., gates) connecting lines.
3. Inputs arrive the wires from left.
4. Output on the right side of wires.
5. largest number is output on the bottom line.
6. Sorting algorithms $\implies$ sorting circuits.



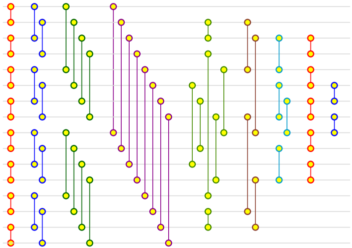
How to draw a circuit...

1. **wires**: horizontal lines
2. **gates**: vertical segments (i.e., gates) connecting lines.
3. Inputs arrive the wires from left.
4. Output on the right side of wires.
5. largest number is output on the bottom line.
6. Sorting algorithms $\Longrightarrow$ sorting circuits.



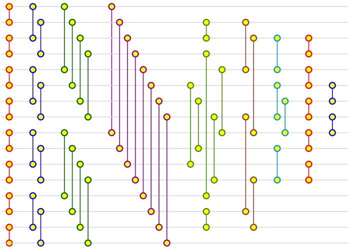
How to draw a circuit...

1. **wires**: horizontal lines
2. **gates**: vertical segments (i.e., gates) connecting lines.
3. Inputs arrive the wires from left.
4. Output on the right side of wires.
5. largest number is output on the the bottom line.
6. Sorting algorithms $\Longrightarrow$ sorting circuits.



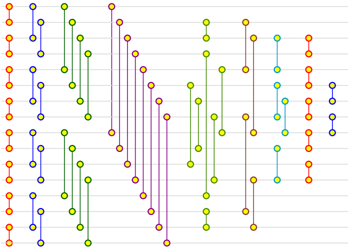
How to draw a circuit...

1. **wires**: horizontal lines
2. **gates**: vertical segments (i.e., gates) connecting lines.
3. Inputs arrive the wires from left.
4. Output on the right side of wires.
5. largest number is output on the bottom line.
6. Sorting algorithms $\implies$ sorting circuits.



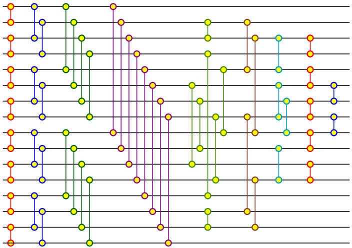
How to draw a circuit...

1. **wires**: horizontal lines
2. **gates**: vertical segments (i.e., gates) connecting lines.
3. Inputs arrive the wires from left.
4. Output on the right side of wires.
5. largest number is output on the bottom line.
6. Sorting algorithms $\Longrightarrow$ sorting circuits.



How to draw a circuit...

1. **wires**: horizontal lines
2. **gates**: vertical segments (i.e., gates) connecting lines.
3. Inputs arrive the wires from left.
4. Output on the right side of wires.
5. largest number is output on the bottom line.
6. Sorting algorithms $\Longrightarrow$ sorting circuits.



Definitions

Definition

A ***comparison network*** is a DAG, with n inputs and n outputs, where each gate has two inputs and two outputs.

Definition

depth of a wire is 0 at input. For gate with two inputs of depth d_1 and d_2 the depth on the output wire is $1 + \max(d_1, d_2)$.
depth of comparison network is maximum depth of an output wire.

Definition

sorting network: comparison network such that for any input, the output is monotonically sorted.
size: sorting network is number of gates.
running time of sorting network is its depth.

Definitions

Definition

A ***comparison network*** is a DAG, with n inputs and n outputs, where each gate has two inputs and two outputs.

Definition

depth of a wire is 0 at input. For gate with two inputs of depth d_1 and d_2 the depth on the output wire is $1 + \max(d_1, d_2)$.
depth of comparison network is maximum depth of an output wire.

Definition

sorting network: comparison network such that for any input, the output is monotonically sorted.
size: sorting network is number of gates.
running time of sorting network is its depth.

Definitions

Definition

A ***comparison network*** is a DAG, with n inputs and n outputs, where each gate has two inputs and two outputs.

Definition

depth of a wire is 0 at input. For gate with two inputs of depth d_1 and d_2 the depth on the output wire is $1 + \max(d_1, d_2)$.
depth of comparison network is maximum depth of an output wire.

Definition

sorting network: comparison network such that for any input, the output is monotonically sorted.
size: sorting network is number of gates.
running time of sorting network is its depth.

Sorting network based on insertion sort

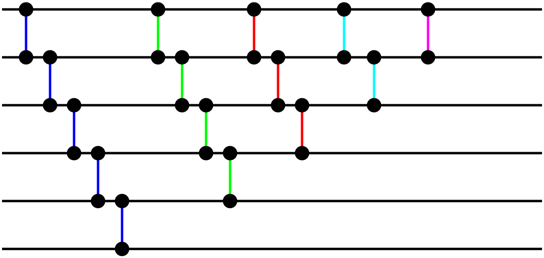
1. Inner loop of insertion sort is:

2. Insertion sort as a network:

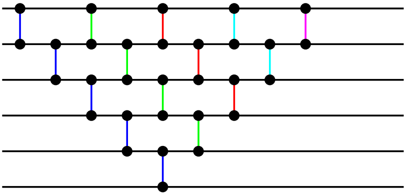
Sorting network based on insertion sort

1. Inner loop of insertion sort is:

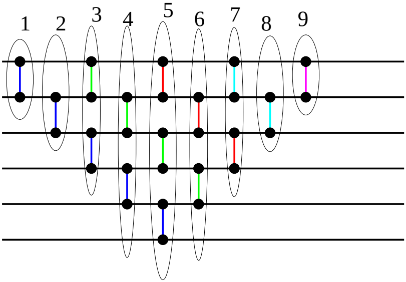
2. Insertion sort as a network:



Sorting network based on insertion sort



(i)

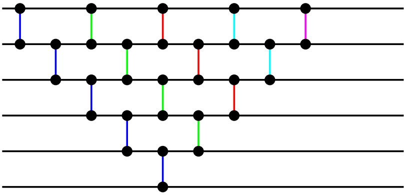


(ii)

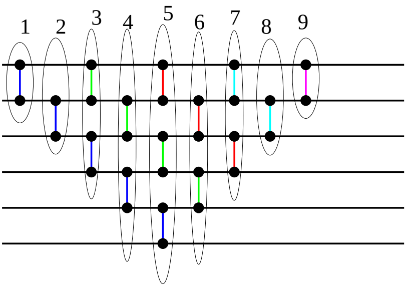
Lemma

The sorting network based on insertion sort has $O(n^2)$ gates, and requires $2n - 1$ time units to sort n numbers.

Sorting network based on insertion sort



(i)

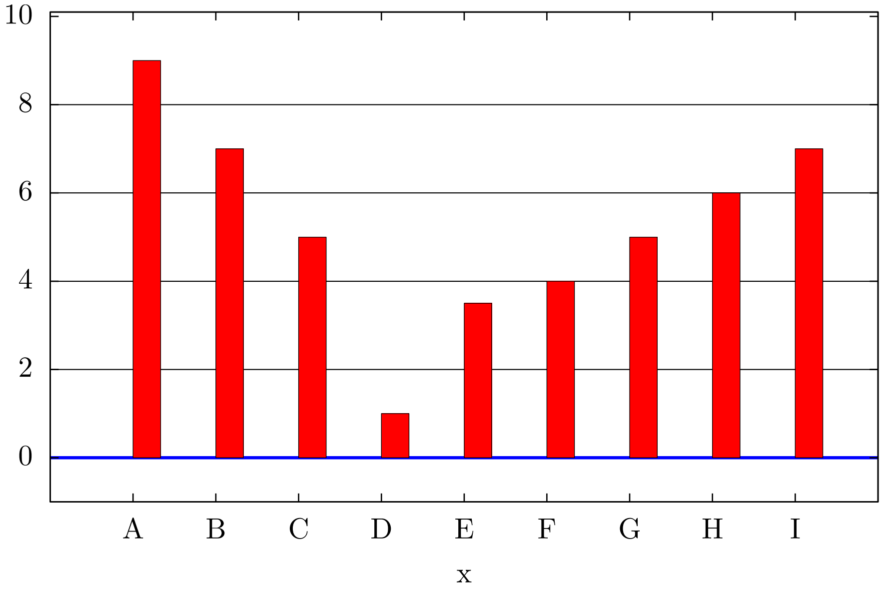


(ii)

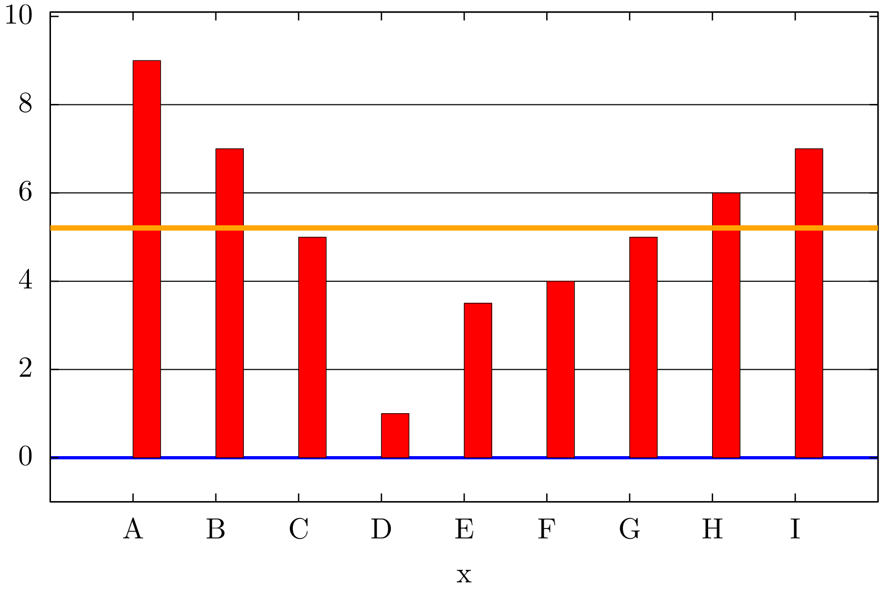
Lemma

The sorting network based on insertion sort has $O(n^2)$ gates, and requires $2n - 1$ time units to sort n numbers.

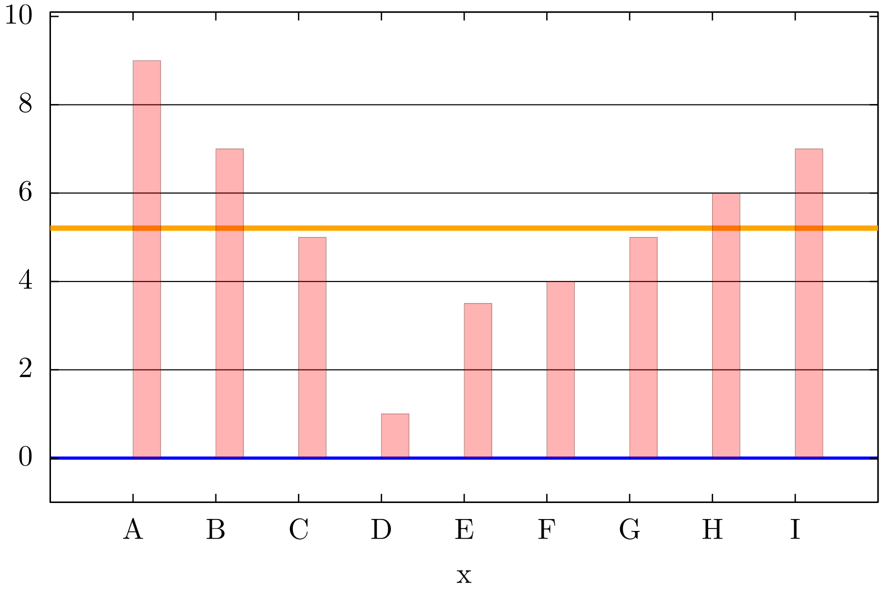
Converting a sequence into a binary sequence



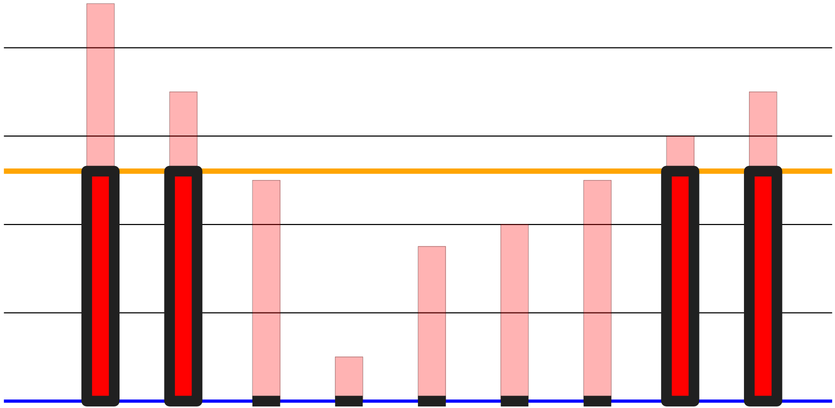
Converting a sequence into a binary sequence



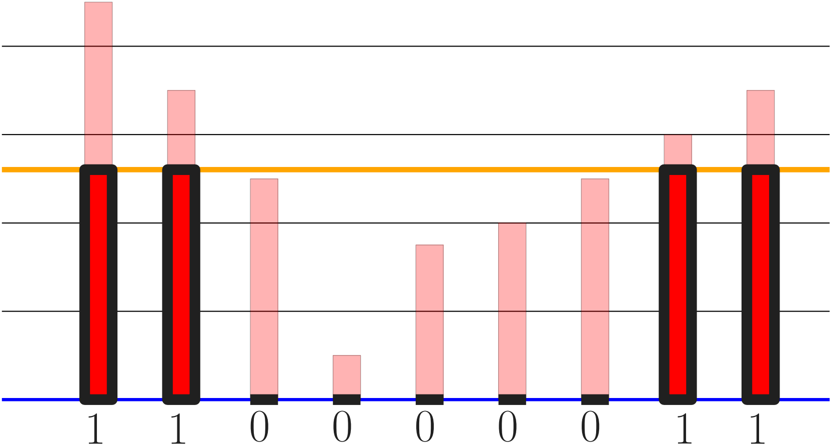
Converting a sequence into a binary sequence



Converting a sequence into a binary sequence



Converting a sequence into a binary sequence



The Zero-One Principle

Definition

zero-one principle states that if a comparison network sort correctly all binary inputs ($\boldsymbol{forall}$ inputs is 0 or 1) then it sorts correctly all inputs.

Need to prove the zero-one principle.

Lemma

A comparison network transforms input sequence

$$a = \langle a_1, a_2, \ldots, a_n \rangle \implies b = \langle b_1, b_2, \ldots, b_n \rangle$$

Then for any monotonically increasing function f, the network transforms

$$f(a) = \Big\langle f(a_1), \ldots, f(a_n) \Big\rangle \implies f(b) = \Big\langle f(b_1), \ldots, f(b_n) \Big\rangle$$

The Zero-One Principle

Definition

zero-one principle states that if a comparison network sort correctly all binary inputs ($\textit{forall}$ inputs is 0 or 1) then it sorts correctly all inputs.

Need to prove the zero-one principle.

Lemma

A comparison network transforms input sequence

$$a = \langle a_1, a_2, \ldots, a_n \rangle \implies b = \langle b_1, b_2, \ldots, b_n \rangle$$

Then for any monotonically increasing function f, the network transforms

$$f(a) = \Big\langle f(a_1), \ldots, f(a_n) \Big\rangle \implies f(b) = \Big\langle f(b_1), \ldots, f(b_n) \Big\rangle$$

The Zero-One Principle

Definition

zero-one principle states that if a comparison network sort correctly all binary inputs ($\boldsymbol{forall}$ inputs is 0 or 1) then it sorts correctly all inputs.

Need to prove the zero-one principle.

Lemma

A comparison network transforms input sequence

$$\boldsymbol{a} = \langle a_1, a_2, \ldots, a_n \rangle \implies \boldsymbol{b} = \langle b_1, b_2, \ldots, b_n \rangle$$

Then for any monotonically increasing function f, *the network transforms*

$$f(a) = \left\langle f(a_1), \ldots, f(a_n) \right\rangle \implies f(b) = \left\langle f(b_1), \ldots, f(b_n) \right\rangle$$

Proof

1. Induction on number of comparators.
2. Consider a comparator with inputs x and y, and outputs $x' = \min(x, y)$ and $y' = \max(x, y)$.
3. If $f(x) = f(y)$ then the claim trivially holds.
4. If $f(x) < f(y)$ then clearly

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

   since $f(\cdot)$ is monotonically increasing.
5. $\langle x, y \rangle$, for $x < y$, we have output $\langle x, y \rangle$.
6. Input: $\langle f(x), f(y) \rangle \implies$ output is $\langle f(x), f(y) \rangle$.
7. Similarly, if $x > y$, the output is $\langle y, x \rangle$. In this case, for the input $\langle f(x), f(y) \rangle$ the output is $\langle f(y), f(x) \rangle$. This establish the claim for a single comparator.

Proof

1. Induction on number of comparators.
2. Consider a comparator with inputs x and y, and outputs $x' = \min(x, y)$ and $y' = \max(x, y)$.
3. If $f(x) = f(y)$ then the claim trivially holds.
4. If $f(x) < f(y)$ then clearly

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

since $f(\cdot)$ is monotonically increasing.
5. $\langle x, y \rangle$, for $x < y$, we have output $\langle x, y \rangle$.
6. Input: $\langle f(x), f(y) \rangle \implies$ output is $\langle f(x), f(y) \rangle$.
7. Similarly, if $x > y$, the output is $\langle y, x \rangle$. In this case, for the input $\langle f(x), f(y) \rangle$ the output is $\langle f(y), f(x) \rangle$. This establish the claim for a single comparator.

Proof

1. Induction on number of comparators.
2. Consider a comparator with inputs x and y, and outputs $x' = \min(x, y)$ and $y' = \max(x, y)$.
3. If $f(x) = f(y)$ then the claim trivially holds.
4. If $f(x) < f(y)$ then clearly

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

   since $f(\cdot)$ is monotonically increasing.
5. $\langle x, y\rangle$, for $x < y$, we have output $\langle x, y\rangle$.
6. Input: $\langle f(x), f(y)\rangle \implies$ output is $\langle f(x), f(y)\rangle$.
7. Similarly, if $x > y$, the output is $\langle y, x\rangle$. In this case, for the input $\langle f(x), f(y)\rangle$ the output is $\langle f(y), f(x)\rangle$. This establish the claim for a single comparator.

Proof

1. Induction on number of comparators.
2. Consider a comparator with inputs x and y, and outputs $x' = \min(x, y)$ and $y' = \max(x, y)$.
3. If $f(x) = f(y)$ then the claim trivially holds.
4. If $f(x) < f(y)$ then clearly

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

   since $f(\cdot)$ is monotonically increasing.
5. $\langle x, y \rangle$, for $x < y$, we have output $\langle x, y \rangle$.
6. Input: $\langle f(x), f(y) \rangle \implies$ output is $\langle f(x), f(y) \rangle$.
7. Similarly, if $x > y$, the output is $\langle y, x \rangle$. In this case, for the input $\langle f(x), f(y) \rangle$ the output is $\langle f(y), f(x) \rangle$. This establish the claim for a single comparator.

Proof

1. Induction on number of comparators.
2. Consider a comparator with inputs x and y, and outputs $x' = \min(x, y)$ and $y' = \max(x, y)$.
3. If $f(x) = f(y)$ then the claim trivially holds.
4. If $f(x) < f(y)$ then clearly

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

   since $f(\cdot)$ is monotonically increasing.
5. $\langle x, y\rangle$, for $x < y$, we have output $\langle x, y\rangle$.
6. Input: $\langle f(x), f(y)\rangle \implies$ output is $\langle f(x), f(y)\rangle$.
7. Similarly, if $x > y$, the output is $\langle y, x\rangle$. In this case, for the input $\langle f(x), f(y)\rangle$ the output is $\langle f(y), f(x)\rangle$. This establish the claim for a single comparator.

Proof

1. Induction on number of comparators.
2. Consider a comparator with inputs x and y, and outputs $x' = \min(x, y)$ and $y' = \max(x, y)$.
3. If $f(x) = f(y)$ then the claim trivially holds.
4. If $f(x) < f(y)$ then clearly

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

   since $f(\cdot)$ is monotonically increasing.
5. $\langle x, y\rangle$, for $x < y$, we have output $\langle x, y\rangle$.
6. Input: $\langle f(x), f(y)\rangle \implies$ output is $\langle f(x), f(y)\rangle$.
7. Similarly, if $x > y$, the output is $\langle y, x\rangle$. In this case, for the input $\langle f(x), f(y)\rangle$ the output is $\langle f(y), f(x)\rangle$. This establish the claim for a single comparator.

Proof continued

1. Claim: if a wire carry a value a_i, when the sorting network get input $a_1, \ldots, a_n$, then for input $f(a_1), \ldots, f(a_n)$ this wire would carry the value $f(a_i)$.
2. Proof by induction on the depth on the wire at each point.
3. If point has depth 0, then its input and claim trivially hold.
4. Assume holds for all points in circuit of depth $\leq qi$, and consider a point p on a wire of depth $i+1$.
5. G: gate which this wire is an output of.
6. By induction, claim holds for inputs of G.
   Now, the claim holds for the gate G itself.
   Apply above single gate proof for G.
   $\implies$ claim holds at p.

Proof continued

1. Claim: if a wire carry a value a_i, when the sorting network get input $a_1, \ldots, a_n$, then for input $f(a_1), \ldots, f(a_n)$ this wire would carry the value $f(a_i)$.
2. Proof by induction on the depth on the wire at each point.
3. If point has depth 0, then its input and claim trivially hold.
4. Assume holds for all points in circuit of depth $\leq qi$, and consider a point p on a wire of depth $i + 1$.
5. G: gate which this wire is an output of.
6. By induction, claim holds for inputs of G.
   Now, the claim holds for the gate G itself.
   Apply above single gate proof for G.
   $\implies$ claim holds at p.

Proof continued

1. Claim: if a wire carry a value a_i, when the sorting network get input $a_1, \ldots, a_n$, then for input $f(a_1), \ldots, f(a_n)$ this wire would carry the value $f(a_i)$.
2. Proof by induction on the depth on the wire at each point.
3. If point has depth 0, then its input and claim trivially hold.
4. Assume holds for all points in circuit of depth $\leq qi$, and consider a point p on a wire of depth $i + 1$.
5. G: gate which this wire is an output of.
6. By induction, claim holds for inputs of G.
   Now, the claim holds for the gate G itself.
   Apply above single gate proof for G.
   $\implies$ claim holds at p.

Proof continued

1. Claim: if a wire carry a value a_i, when the sorting network get input $a_1, \ldots, a_n$, then for input $f(a_1), \ldots, f(a_n)$ this wire would carry the value $f(a_i)$.
2. Proof by induction on the depth on the wire at each point.
3. If point has depth 0, then its input and claim trivially hold.
4. Assume holds for all points in circuit of depth $\leq qi$, and consider a point p on a wire of depth $i+1$.
5. G: gate which this wire is an output of.
6. By induction, claim holds for inputs of G.
   Now, the claim holds for the gate G itself.
   Apply above single gate proof for G.
   $\implies$ claim holds at p.

Proof continued

1. Claim: if a wire carry a value a_i, when the sorting network get input $a_1, \ldots, a_n$, then for input $f(a_1), \ldots, f(a_n)$ this wire would carry the value $f(a_i)$.
2. Proof by induction on the depth on the wire at each point.
3. If point has depth 0, then its input and claim trivially hold.
4. Assume holds for all points in circuit of depth $\leq qi$, and consider a point p on a wire of depth $i+1$.
5. G: gate which this wire is an output of.
6. By induction, claim holds for inputs of G.
   Now, the claim holds for the gate G itself.
   Apply above single gate proof for G.
   $\implies$ claim holds at p.

$0/1$ sorting implies real sorting

Theorem

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

Proof: $0/1$ sorting implies real sorting

1. Assume for contradiction that fails for input $a_1, \ldots, a_n$. Let $b_1, \ldots b_n$ be the output sequence for this input.
2. Let $a_i < a_k$ be the two numbers that are output in incorrect order (i.e. a_k appears before a_i in output).
3. $f(x) = \begin{cases} 0 & x \leq a_i \\ 1 & x > a_i. \end{cases}$
4. By lemma for input $\langle f(a_1), \ldots, f(a_n) \rangle$, circuit would output $\langle f(b_1), \ldots, f(b_n) \rangle$.
5. This sequence looks like: $000..0????f(a_k)????f(a_i)??1111$
6. but $f(a_i) = 0$ and $f(a_j) = 1$. Namely, the output is a sequence of the form $????1????0????$, which is not sorted.
7. bin. input $\langle f(b_1), \ldots, f(b_n) \rangle$ sorting net' fails. A contradiction. ∎

Proof: $0/1$ sorting implies real sorting

1. Assume for contradiction that fails for input $a_1, \ldots, a_n$. Let $b_1, \ldots b_n$ be the output sequence for this input.
2. Let $a_i < a_k$ be the two numbers that are output in incorrect order (i.e. a_k appears before a_i in output).
3. $f(x) = \begin{cases} 0 & x \leq a_i \\ 1 & x > a_i. \end{cases}$
4. By lemma for input $\langle f(a_1), \ldots, f(a_n) \rangle$, circuit would output $\langle f(b_1), \ldots, f(b_n) \rangle$.
5. This sequence looks like: $000..0????f(a_k)????f(a_i)??1111$
6. but $f(a_i) = 0$ and $f(a_j) = 1$. Namely, the output is a sequence of the form $????1????0????$, which is not sorted.
7. bin. input $\langle f(b_1), \ldots, f(b_n) \rangle$ sorting net' fails. A contradiction. ∎

Proof: $0/1$ sorting implies real sorting

1. Assume for contradiction that fails for input $a_1, \ldots, a_n$. Let $b_1, \ldots b_n$ be the output sequence for this input.
2. Let $a_i < a_k$ be the two numbers that are output in incorrect order (i.e. a_k appears before a_i in output).
3. $f(x) = \begin{cases} 0 & x \leq a_i \\ 1 & x > a_i. \end{cases}$
4. By lemma for input $\langle f(a_1), \ldots, f(a_n) \rangle$, circuit would output $\langle f(b_1), \ldots, f(b_n) \rangle$.
5. This sequence looks like: $000..0????f(a_k)????f(a_i)??1111$
6. but $f(a_i) = 0$ and $f(a_j) = 1$. Namely, the output is a sequence of the form $????1????0????$, which is not sorted.
7. bin. input $\langle f(b_1), \ldots, f(b_n) \rangle$ sorting net' fails. A contradiction.

Proof: $0/1$ sorting implies real sorting

1. Assume for contradiction that fails for input $a_1, \ldots, a_n$. Let $b_1, \ldots b_n$ be the output sequence for this input.
2. Let $a_i < a_k$ be the two numbers that are output in incorrect order (i.e. a_k appears before a_i in output).
3. $f(x) = \begin{cases} 0 & x \leq a_i \\ 1 & x > a_i. \end{cases}$
4. By lemma for input $\langle f(a_1), \ldots, f(a_n) \rangle$, circuit would output $\langle f(b_1), \ldots, f(b_n) \rangle$.
5. This sequence looks like: $000..0????f(a_k)????f(a_i)??1111$
6. but $f(a_i) = 0$ and $f(a_j) = 1$. Namely, the output is a sequence of the form $????1????0????$, which is not sorted.
7. bin. input $\langle f(b_1), \ldots, f(b_n) \rangle$ sorting net' fails. A contradiction. ∎

Proof: $0/1$ sorting implies real sorting

1. Assume for contradiction that fails for input $a_1, \ldots, a_n$. Let $b_1, \ldots b_n$ be the output sequence for this input.
2. Let $a_i < a_k$ be the two numbers that are output in incorrect order (i.e. a_k appears before a_i in output).
3. $f(x) = \begin{cases} 0 & x \leq a_i \\ 1 & x > a_i. \end{cases}$
4. By lemma for input $\langle f(a_1), \ldots, f(a_n) \rangle$, circuit would output $\langle f(b_1), \ldots, f(b_n) \rangle$.
5. This sequence looks like: $000..0????f(a_k)????f(a_i)??1111$
6. but $f(a_i) = 0$ and $f(a_j) = 1$. Namely, the output is a sequence of the form $????1????0????$, which is not sorted.
7. bin. input $\langle f(b_1), \ldots, f(b_n) \rangle$ sorting net' fails. A contradiction. ∎

Proof: $0/1$ sorting implies real sorting

1. Assume for contradiction that fails for input $a_1, \ldots, a_n$. Let $b_1, \ldots b_n$ be the output sequence for this input.
2. Let $a_i < a_k$ be the two numbers that are output in incorrect order (i.e. a_k appears before a_i in output).
3. $f(x) = \begin{cases} 0 & x \leq a_i \\ 1 & x > a_i. \end{cases}$
4. By lemma for input $\langle f(a_1), \ldots, f(a_n) \rangle$, circuit would output $\langle f(b_1), \ldots, f(b_n) \rangle$.
5. This sequence looks like: $000..0????f(a_k)????f(a_i)??1111$
6. but $f(a_i) = 0$ and $f(a_j) = 1$. Namely, the output is a sequence of the form $????1????0????$, which is not sorted.
7. bin. input $\langle f(b_1), \ldots, f(b_n) \rangle$ sorting net' fails. A contradiction. ∎

Proof: $0/1$ sorting implies real sorting

1. Assume for contradiction that fails for input $a_1, \ldots, a_n$. Let $b_1, \ldots b_n$ be the output sequence for this input.
2. Let $a_i < a_k$ be the two numbers that are output in incorrect order (i.e. a_k appears before a_i in output).
3. $f(x) = \begin{cases} 0 & x \leq a_i \\ 1 & x > a_i. \end{cases}$
4. By lemma for input $\langle f(a_1), \ldots, f(a_n) \rangle$, circuit would output $\langle f(b_1), \ldots, f(b_n) \rangle$.
5. This sequence looks like: $000..0????f(a_k)????f(a_i)??1111$
6. but $f(a_i) = 0$ and $f(a_j) = 1$. Namely, the output is a sequence of the form $????1????0????$, which is not sorted.
7. bin. input $\langle f(b_1), \ldots, f(b_n) \rangle$ sorting net' fails. A contradiction.

Proof: $0/1$ sorting implies real sorting

1. Assume for contradiction that fails for input $a_1, \ldots, a_n$. Let $b_1, \ldots b_n$ be the output sequence for this input.
2. Let $a_i < a_k$ be the two numbers that are output in incorrect order (i.e. a_k appears before a_i in output).
3. $f(x) = \begin{cases} 0 & x \leq a_i \\ 1 & x > a_i. \end{cases}$
4. By lemma for input $\langle f(a_1), \ldots, f(a_n) \rangle$, circuit would output $\langle f(b_1), \ldots, f(b_n) \rangle$.
5. This sequence looks like: $000..0????f(a_k)????f(a_i)??1111$
6. but $f(a_i) = 0$ and $f(a_j) = 1$. Namely, the output is a sequence of the form $????1????0????$, which is not sorted.
7. bin. input $\langle f(b_1), \ldots, f(b_n) \rangle$ sorting net' fails. A contradiction. ■

Part I

A bitonic sorting network

Bitonic sorting network

Definition

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

example

The sequences $(1, 2, 3, \pi, 4, 5, 4, 3, 2, 1)$ and $(4, 5, 4, 3, 2, 1, 1, 2, 3)$ are bitonic, while the sequence $(1, 2, 1, 2)$ is not bitonic.

Bitonic sorting network

Definition

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

example

The sequences $(1, 2, 3, \pi, 4, 5, 4, 3, 2, 1)$ and $(4, 5, 4, 3, 2, 1, 1, 2, 3)$ are bitonic, while the sequence $(1, 2, 1, 2)$ is not bitonic.

Binary bitonic sequences

Observation

binary bitonic sequence is either of the form $0^i 1^j 0^k$ *or of the form* $1^i 0^j 1^k$, *where* 0^i *(resp,* 1^i*) denote a sequence of* i *zeros (resp., ones).*

Bitonic sorting network

Definition

A ***bitonic sorter*** is a comparison network that sorts all bitonic sequences correctly.

Half cleaner...

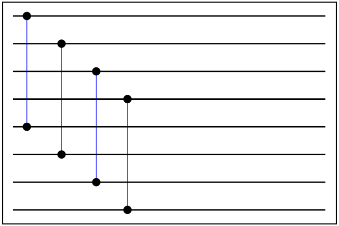
Definition

half-cleaner: a comparison network, connecting line i with line $i + n/2$.

Half cleaner...

Definition

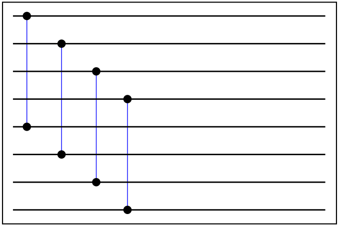
half-cleaner: a comparison network, connecting line i with line $i + n/2$.



Half cleaner...

Definition

half-cleaner: a comparison network, connecting line i with line $i + n/2$.

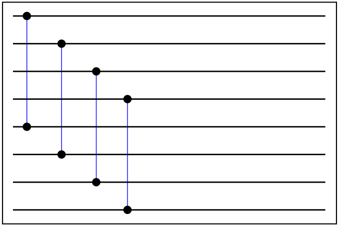


Half-Cleaner$[n]$ denote half-cleaner with n inputs.

Half cleaner...

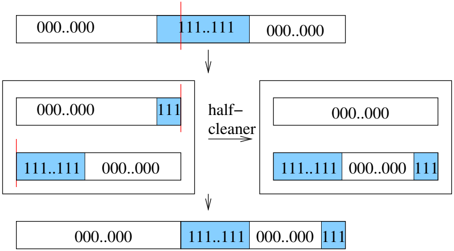
Definition

half-cleaner: a comparison network, connecting line i with line $i + n/2$.



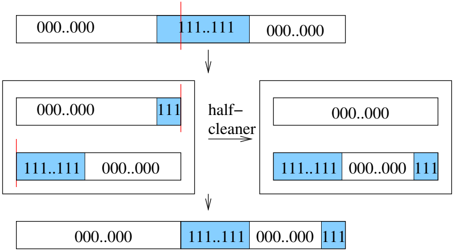
Half-Cleaner$[n]$ denote half-cleaner with n inputs.
Depth of **Half-Cleaner**$[n]$ is one.

Half cleaner on bitonic sequence...



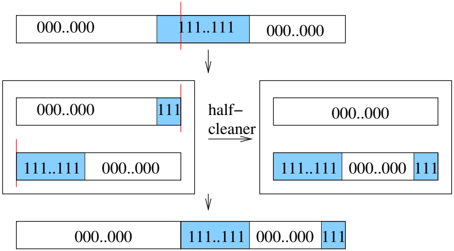
1. What a half-cleaner do to an input which is a (binary) bitonic sequence?
2. In example... left half size is clean and all equal to 0.
3. Right side of the output is bitonic.
4. Specifically, one can prove by simple (but tedious) case analysis that the following lemma holds.

Half cleaner on bitonic sequence...



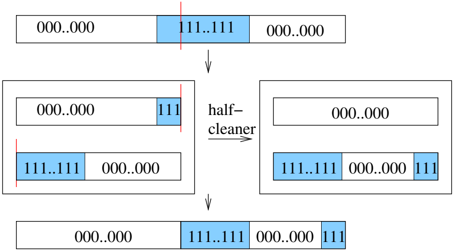
1. What a half-cleaner do to an input which is a (binary) bitonic sequence?
2. In example... left half size is clean and all equal to $\mathbf{0}$.
3. Right side of the output is bitonic.
4. Specifically, one can prove by simple (but tedious) case analysis that the following lemma holds.

Half cleaner on bitonic sequence...



1. What a half-cleaner do to an input which is a (binary) bitonic sequence?
2. In example... left half size is clean and all equal to **0**.
3. Right side of the output is bitonic.
4. Specifically, one can prove by simple (but tedious) case analysis that the following lemma holds.

Half cleaner on bitonic sequence...



1. What a half-cleaner do to an input which is a (binary) bitonic sequence?
2. In example... left half size is clean and all equal to $\mathbf{0}$.
3. Right side of the output is bitonic.
4. Specifically, one can prove by simple (but tedious) case analysis that the following lemma holds.

Half cleaner half sorts a bitonic sequence...

Lemma

If the input to a half-cleaner (of size n) is a binary bitonic sequence then for the output sequence we have that

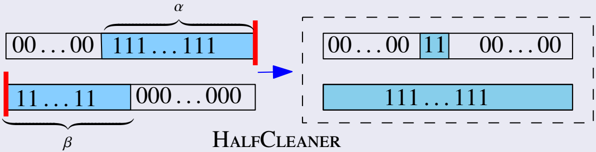
(i) the elements in the top half are smaller than the elements in bottom half, and

(ii) one of the halves is clean, and the other is bitonic.

Proof

Proof.

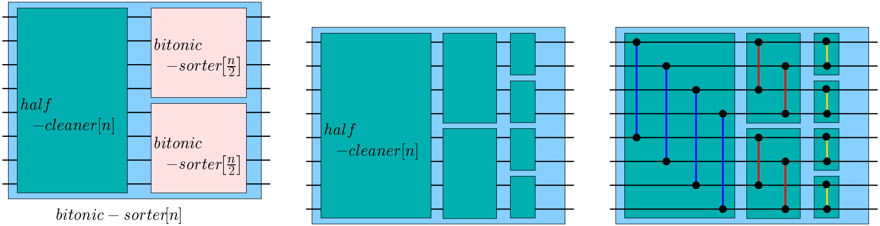
If the sequence is of the form $0^i1^j0^k$ and the block of ones is completely on the left side (i.e., its part of the first $n/2$ bits) or the right side, the claim trivially holds. So, assume that the block of ones starts at position $n/2 - \beta$ and ends at $n/2 + \alpha$.



If $n/2 - \alpha \geq \beta$ then this is exactly the case depicted above and claim holds. If $n/2 - \alpha < \beta$ then the second half is going to be all ones, as depicted on the right. Implying the claim for this case.

A similar analysis holds if the sequence is of the form $1^i0^j1^k$. □

Bitonic sorter - sorts bitonic sequences...



(i) (ii) (iii)

(i) recursive construction of **BitonicSorter**$[n]$,
(ii) opening up the recursive construction, and
(iii) the resulting comparison network.

Bitonic sorter... the result

Lemma

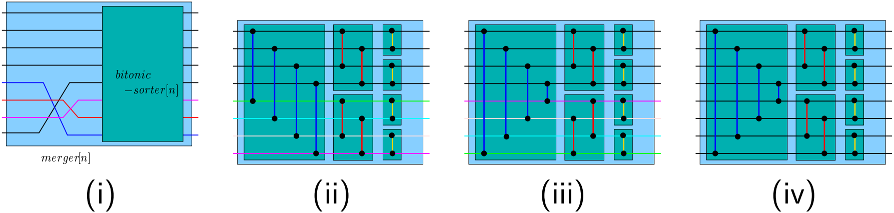
BitonicSorter$[n]$ *sorts bitonic sequences of length* $n = 2^k$*, it uses* $(n/2)k = (n/2)\lg n$ *gates, and it is of depth* $k = \lg n$.

Merging sequence

1. Merging question: Given two *sorted* sequences of length $n/2$, how do we merge them into a single sorted sequence?
2. Concatenate the two sequences...
3. ... second sequence is being flipped (i.e., reversed).
4. Easy to verify that the resulting sequence is bitonic, and as such we can sort it using the **BitonicSorter**$[n]$.
5. Given two sorted sequences $a_1 \leq a_2 \leq \ldots \leq a_n$ and $b_1 \leq b_2 \leq \ldots \leq b_n$, observe that the sequence $a_1, a_2, \ldots, a_n, b_n, b_{n-1}, b_{n-2}, \ldots, b_2, b_1$ is bitonic.

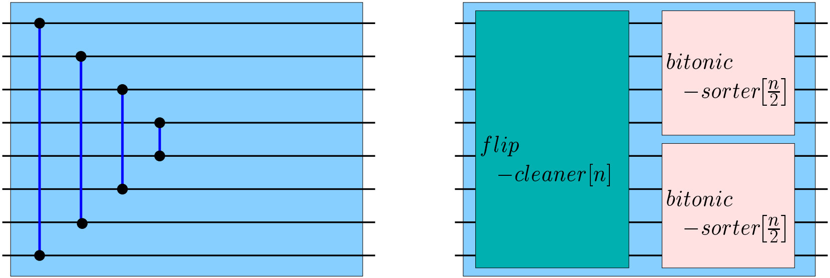
Merger[n]: Using a bitonic sorter

Merging two sorted sequences into a sorted sequence



(i) **Merger** via flipping the lines of bitonic sorter.
(ii) **BitonicSorter**.
(iii) **Merger** after we "physically" flip the lines.
(iv) Equivalent drawing of the resulting **Merger**.

Merger[n] described using FlipCleaner



(i) (ii)

(i) **FlipCleaner**$[n]$, and
(ii) **Merger**$[n]$ described using **FlipCleaner**.

What $\mathrm{Merger[n]}$ does...

Lemma

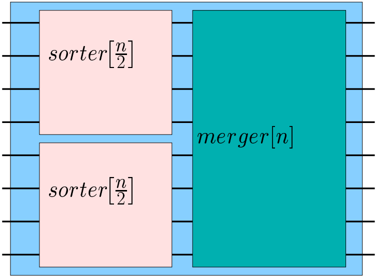
The circuit **Merger**$[n]$ *gets as input two sorted sequences of length* $n/2 = 2^{k-1}$*, it uses* $(n/2)k = (n/2)\lg n$ *gates, and it is of depth* $k = \lg n$*, and it outputs a sorted sequence.*

Sorting Network

Finally...

Implement ***merge sort*** using **Merger**$[n]$.

Sorter$[n]$:



Lemma

The circuit **Sorter**$[n]$ *is a sorting network (i.e., it sorts any* n *numbers) using* $G(n) = O(n \log^2 n)$ *gates. It has depth* $O(\log^2 n)$*. Namely,* **Sorter**$[n]$ *sorts* n *numbers in* $O(\log^2 n)$ *time.*

Proof

Proof.

The number of gates is

$$G(n) = 2G(n/2) + Gates(\text{Merger}[n]).$$

Which is $G(n) = 2G(n/2) + O(n \log n) = O(n \log^2 n)$.
As for the depth, we have that
$D(n) = D(n/2) + \text{Depth}(\text{Merger}[n]) = D(n/2) + O(\log(n))$,
and thus $D(n) = O(\log^2 n)$, as claimed. □

Resulting sorted

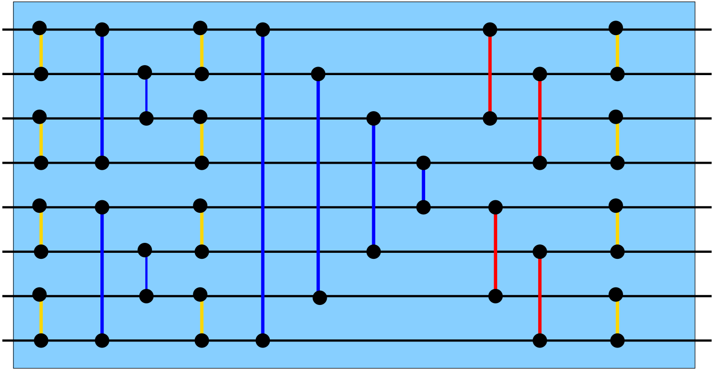


Figure: **Sorter**[8].

Faster sorting networks

1. Known: sorting network of logarithmic depth **?**.
2. Known as the ***AKS sorting network***.
3. Construction is complicated.
4. **?** is better than bitonic sort for n larger than 2^{8046}.

Notes

Notes

Notes

Notes