The Bose-Nelson Sorting Problem for 11 and 12 Inputs

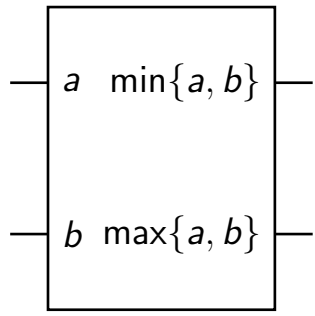
Jannis Harder

July 7, 2020

Overview

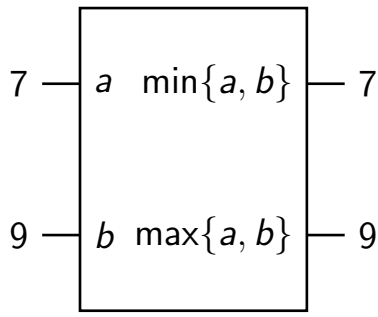
Comparator Networks and Sorting Networks

- Circuit of *comparator* gates



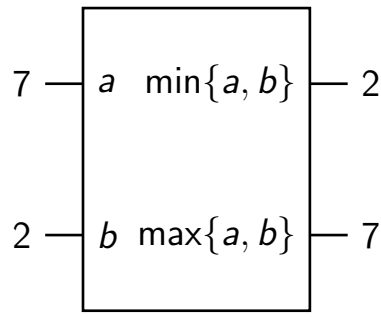
Comparator Networks and Sorting Networks

- Circuit of *comparator* gates
- Values from any totally ordered set



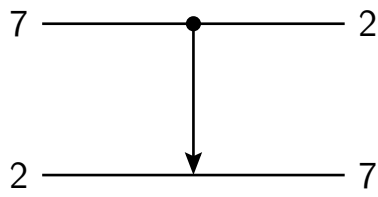
Comparator Networks and Sorting Networks

- Circuit of *comparator* gates
- Values from any totally ordered set



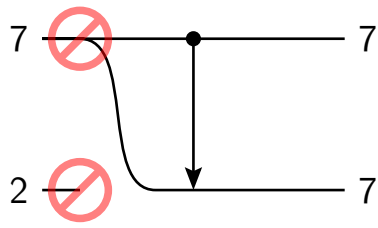
Comparator Networks and Sorting Networks

- Circuit of *comparator* gates
- Values from any totally ordered set



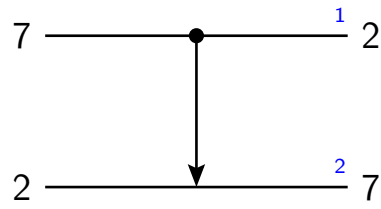
Comparator Networks and Sorting Networks

- Circuit of *comparator* gates
- Values from any totally ordered set
- No discarding or duplication



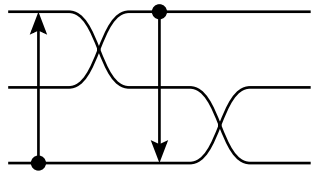
Comparator Networks and Sorting Networks

- Circuit of *comparator* gates
- Values from any totally ordered set
- No discarding or duplication
- Outputs are ordered (top to bottom)



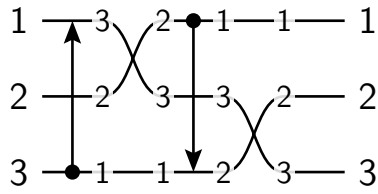
Comparator Networks and Sorting Networks

- Circuit of *comparator* gates
- Values from any totally ordered set
- No discarding or duplication
- Outputs are ordered (top to bottom)



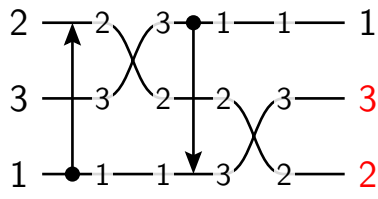
Comparator Networks and Sorting Networks

- Circuit of *comparator* gates
- Values from any totally ordered set
- No discarding or duplication
- Outputs are ordered (top to bottom)



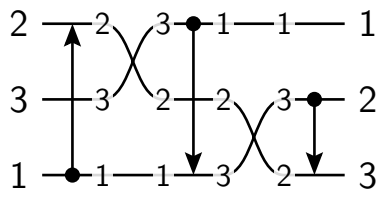
Comparator Networks and Sorting Networks

- Circuit of *comparator* gates
- Values from any totally ordered set
- No discarding or duplication
- Outputs are ordered (top to bottom)



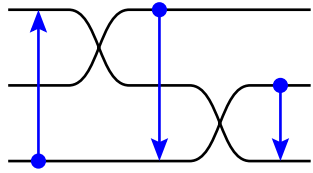
Comparator Networks and Sorting Networks

- Circuit of *comparator* gates
- Values from any totally ordered set
- No discarding or duplication
- Outputs are ordered (top to bottom)
- *Sorting network* if output always sorted



Comparator Networks and Sorting Networks

- Circuit of *comparator* gates
- Values from any totally ordered set
- No discarding or duplication
- Outputs are ordered (top to bottom)
- *Sorting network* if output always sorted
- The *size* is the number of comparators



The Bose-Nelson Sorting Problem

What is the minimum size s_n of a sorting network with n inputs?

Timeline of Selected Results

- Information theory bound: $s_n \geq \lceil \log_2 n! \rceil$, $\Omega(n \log n)$.

Timeline of Selected Results

- Information theory bound: $s_n \geq \lceil \log_2 n! \rceil$, $\Omega(n \log n)$.
- 1954 Armstrong, Nelson & O'Connor: $O(n^2)$ construction and smaller networks for up to 8 inputs and (US Patent 3,029,413)

Timeline of Selected Results

- Information theory bound: $s_n \geq \lceil \log_2 n! \rceil$, $\Omega(n \log n)$.
- 1954 Armstrong, Nelson & O'Connor: $O(n^2)$ construction and smaller networks for up to 8 inputs and (US Patent 3,029,413)
- 1961 Bose & Nelson: $O(n^{1.585})$ construction, conjectured optimal.

Timeline of Selected Results

- Information theory bound: $s_n \geq \lceil \log_2 n! \rceil$, $\Omega(n \log n)$.
- 1954 Armstrong, Nelson & O'Connor: $O(n^2)$ construction and smaller networks for up to 8 inputs and (US Patent 3,029,413)
- 1961 Bose & Nelson: $O(n^{1.585})$ construction, conjectured optimal.
- 1964 Batcher: Recursive $O(n\,(\log n)^2)$ construction.

Timeline of Selected Results

- Information theory bound: $s_n \geq \lceil \log_2 n! \rceil$, $\Omega(n \log n)$.
- 1954 Armstrong, Nelson & O'Connor: $O(n^2)$ construction and smaller networks for up to 8 inputs and (US Patent 3,029,413)
- 1961 Bose & Nelson: $O(n^{1.585})$ construction, conjectured optimal.
- 1964 Batcher: Recursive $O(n\,(\log n)^2)$ construction.
- 1966 Floyd & Knuth: Known networks up to 7 inputs optimal.

Timeline of Selected Results

- Information theory bound: $s_n \geq \lceil \log_2 n! \rceil$, $\Omega(n \log n)$.
- 1954 Armstrong, Nelson & O'Connor: $O(n^2)$ construction and smaller networks for up to 8 inputs and (US Patent 3,029,413)
- 1961 Bose & Nelson: $O(n^{1.585})$ construction, conjectured optimal.
- 1964 Batcher: Recursive $O(n (\log n)^2)$ construction.
- 1966 Floyd & Knuth: Known networks up to 7 inputs optimal.
- 1972 Van Voorhis: $s_n \geq s_{n-1} + \lceil \log_2 n \rceil$, known 8 inputs optimal.

Timeline of Selected Results

- Information theory bound: $s_n \geq \lceil \log_2 n! \rceil$, $\Omega(n \log n)$.
- 1954 Armstrong, Nelson & O'Connor: $O(n^2)$ construction and smaller networks for up to 8 inputs and (US Patent 3,029,413)
- 1961 Bose & Nelson: $O(n^{1.585})$ construction, conjectured optimal.
- 1964 Batcher: Recursive $O(n (\log n)^2)$ construction.
- 1966 Floyd & Knuth: Known networks up to 7 inputs optimal.
- 1972 Van Voorhis: $s_n \geq s_{n-1} + \lceil \log_2 n \rceil$, known 8 inputs optimal.
- 1983 Ajtai, Komlós & Szemerédi: $O(n \log n)$ construction.

Timeline of Selected Results

- Information theory bound: $s_n \geq \lceil \log_2 n! \rceil$, $\Omega(n \log n)$.
- 1954 Armstrong, Nelson & O'Connor: $O(n^2)$ construction and smaller networks for up to 8 inputs and (US Patent 3,029,413)
- 1961 Bose & Nelson: $O(n^{1.585})$ construction, conjectured optimal.
- 1964 Batcher: Recursive $O(n (\log n)^2)$ construction.
- 1966 Floyd & Knuth: Known networks up to 7 inputs optimal.
- 1972 Van Voorhis: $s_n \geq s_{n-1} + \lceil \log_2 n \rceil$, known 8 inputs optimal.
- 1983 Ajtai, Komlós & Szemerédi: $O(n \log n)$ construction.
- 2014 Codish, Cruz-Filipe, Frank & Schneider-Kamp: Best known networks with 9 and 10 inputs are optimal.

Known bounds for small n (OEIS A003075)

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
s_n	0	1	3	5	9	12	16	19	25	29	33	37	41	45
											⋮	⋮	⋮	⋮
											35	39	45	51

Upper bounds from TAOCPv3

Known bounds for small n (OEIS A003075)

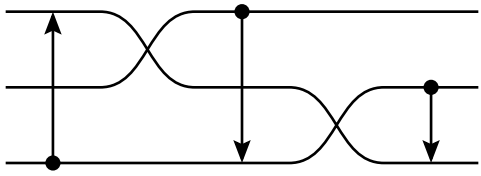
n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
s_n	0	1	3	5	9	12	16	19	25	29	**35**	**39**	**43**	**47**
													⋮	⋮
													45	51

Upper bounds from TAOCPv3

Sorting Network Preliminaries

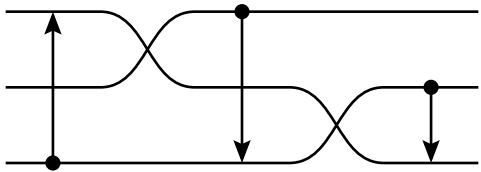
Algebraic Representation

- Sequence of operations on length-n sequences



Algebraic Representation

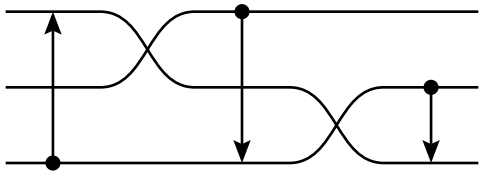
- Sequence of operations on length-n sequences
- Comparator $[i, j]$ and exchange (i, j) operations



$[3,1]\ (1,2)\ [1,3]\ (2,3)\ [2,3]$

Algebraic Representation

- Sequence of operations on length-n sequences
- Comparator $[i, j]$ and exchange (i, j) operations
- Written as postfix from left to right



$[3,1]\ (1,2)\ [1,3]\ (2,3)\ [2,3]$

$$(3,2,1)\ [3,1](1,2)[1,3](2,3)[2,3] = (1,2,3)$$

Standard Forms

- *Permuted standard form*: $i < j$ for all comparators $[i, j]$ and all comparators precede all exchanges

$$[1,2][2,3](1,2) \checkmark \quad [1,2][3,2](1,2) \times \quad [1,2](1,2)[2,3] \times$$

Standard Forms

- *Permuted standard form*: $i < j$ for all comparators $[i, j]$ and all comparators precede all exchanges

$$[1,2][2,3](1,2) \checkmark \quad [1,2][3,2](1,2) \times \quad [1,2](1,2)[2,3] \times$$

- *Standard form*: permuted standard form with no exchanges

$$[1,2][2,3][1,2] \checkmark \quad [1,2][2,3](1,2) \times$$

Standard Forms

- *Permuted standard form*: $i < j$ for all comparators $[i, j]$ and all comparators precede all exchanges

$$[1,2][2,3](1,2) \checkmark \quad [1,2][3,2](1,2) \times \quad [1,2](1,2)[2,3] \times$$

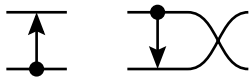
- *Standard form*: permuted standard form with no exchanges

$$[1,2][2,3][1,2] \checkmark \quad [1,2][2,3](1,2) \times$$

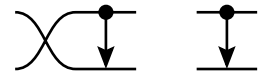
- Every sorting (or comparator) network is equivalent to a network in (permuted) standard form of the same size (Floyd & Knuth 1973)

Standard Forms

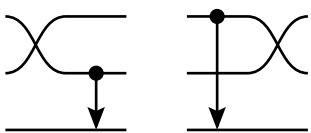
- Every sorting (or comparator) network is equivalent to a network in (permuted) standard form of the same size (Floyd & Knuth 1973)



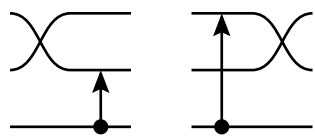
$[j,i] = (i,j)[i,j]$



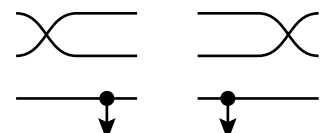
$[i,j](i,j) = [i,j]$



$(i,j)[j,k] = [i,k](i,j)$



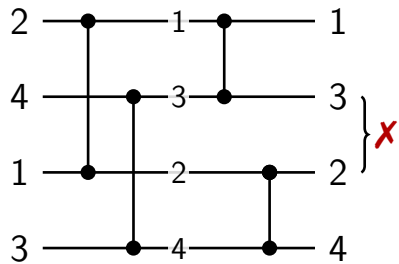
$(i,j)[k,j] = [k,i](i,j)$



$(i,j)[k,l] = [k,l](i,j)$

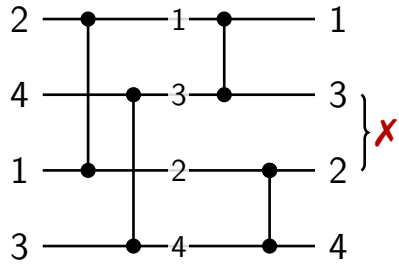
Zero-One Principle

- A comparator network is a sorting network if and only if it sorts any Boolean input sequence (Floyd & Knuth 1973)



Zero-One Principle

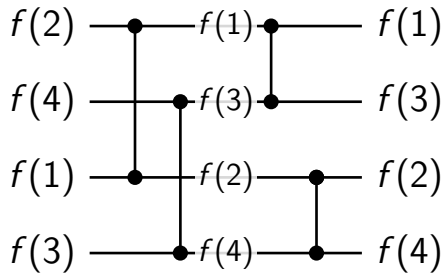
- A comparator network is a sorting network if and only if it sorts any Boolean input sequence (Floyd & Knuth 1973)



$$f(x) = \begin{cases} 0 & \text{if } x \leq 2 \\ 1 & \text{if } x > 2 \end{cases}$$

Zero-One Principle

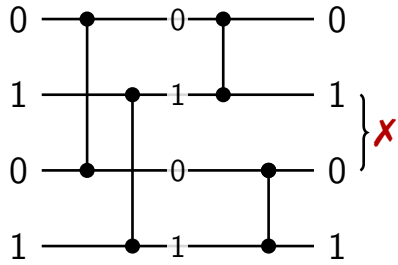
- A comparator network is a sorting network if and only if it sorts any Boolean input sequence (Floyd & Knuth 1973)



$$f(x) = \begin{cases} 0 & \text{if } x \leq 2 \\ 1 & \text{if } x > 2 \end{cases}$$

Zero-One Principle

- A comparator network is a sorting network if and only if it sorts any Boolean input sequence (Floyd & Knuth 1973)



$$f(x) = \begin{cases} 0 & \text{if } x \leq 2 \\ 1 & \text{if } x > 2 \end{cases}$$

Partial Sorting Networks

- A comparator network that sorts all Boolean sequences of a set X is a *partial sorting network* on X

Partial Sorting Networks

- A comparator network that sorts all Boolean sequences of a set X is a *partial sorting network* on X
- $s(X)$ is the minimum size over all partial sorting networks on X

Partial Sorting Networks

- A comparator network that sorts all Boolean sequences of a set X is a *partial sorting network* on X
- $s(X)$ is the minimum size over all partial sorting networks on X
- Let B_n be the set of all length-n Boolean sequences, then $s(B_n) = s_n$

Monotonicity, Symmetries & Successors

- $0 \leq s(Y) \leq s(X) \leq s(B_n) = s_n$ for $Y \subseteq X \subseteq B_n$

Monotonicity, Symmetries & Successors

- $0 \leq s(Y) \leq s(X) \leq s(B_n) = s_n$ for $Y \subseteq X \subseteq B_n$
- If there is a permutation π with $Y = X\pi$ we write $Y \approx X$ and have $s(Y) = s(X)$

Monotonicity, Symmetries & Successors

- $0 \leq s(Y) \leq s(X) \leq s(B_n) = s_n$ for $Y \subseteq X \subseteq B_n$
- If there is a permutation π with $Y = X\pi$ we write $Y \approx X$ and have $s(Y) = s(X)$
- $s(\overline{X}) = s(X)$ where $\overline{X}$ is elementwise negation of all sequences X

Monotonicity, Symmetries & Successors

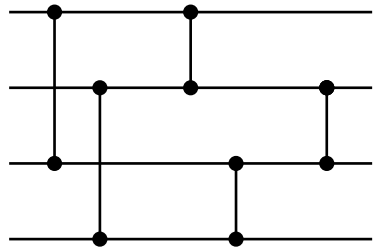
- $0 \leq s(Y) \leq s(X) \leq s(B_n) = s_n$ for $Y \subseteq X \subseteq B_n$
- If there is a permutation π with $Y = X\pi$ we write $Y \approx X$ and have $s(Y) = s(X)$
- $s(\overline{X}) = s(X)$ where $\overline{X}$ is elementwise negation of all sequences X
- If $s(X) > 0$ then $s(X) = 1 + \min\{s(X[i,j]) \mid i < j\}$

Monotonicity, Symmetries & Successors

- $0 \leq s(Y) \leq s(X) \leq s(B_n) = s_n$ for $Y \subseteq X \subseteq B_n$
- If there is a permutation π with $Y = X\pi$ we write $Y \approx X$ and have $s(Y) = s(X)$
- $s(\overline{X}) = s(X)$ where $\overline{X}$ is elementwise negation of all sequences X
- If $s(X) > 0$ then $s(X) = 1 + \min\{s(X[i,j]) \mid i < j, X[i,j] \not\approx X\}$

Van Voorhis' Bound 1/2

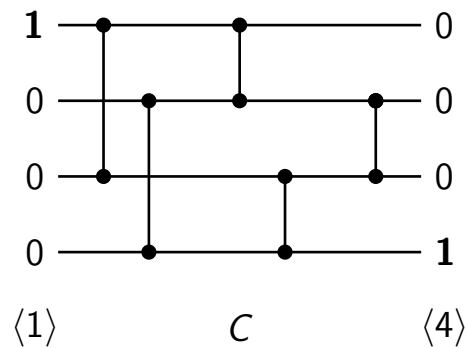
- Sorting network C of size s_n



C

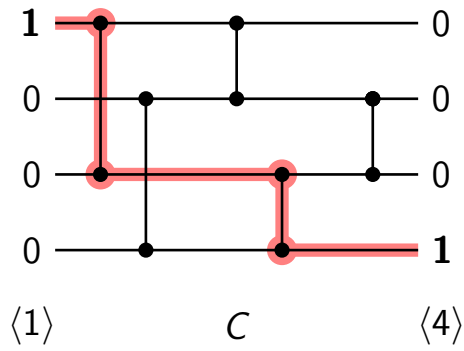
Van Voorhis' Bound 1/2

- Sorting network C of size s_n
- Take a one-hot sequence $\langle i \rangle$: $\langle i \rangle_i = 1$ and $\langle i \rangle_j = 0$ for $j \neq i$



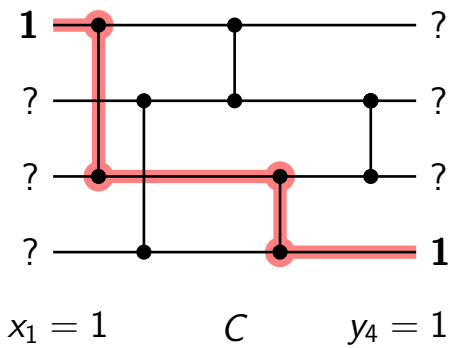
Van Voorhis' Bound 1/2

- Sorting network C of size s_n
- Take a one-hot sequence $\langle i \rangle$: $\langle i \rangle_i = 1$ and $\langle i \rangle_j = 0$ for $j \neq i$
- Trace the path taken by the 1



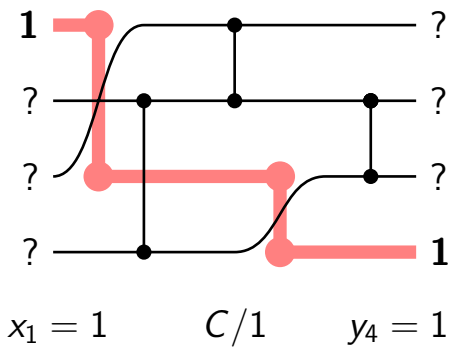
Van Voorhis' Bound 1/2

- Sorting network C of size s_n
- Take a one-hot sequence $\langle i \rangle$: $\langle i \rangle_i = 1$ and $\langle i \rangle_j = 0$ for $j \neq i$
- Trace the path taken by the 1



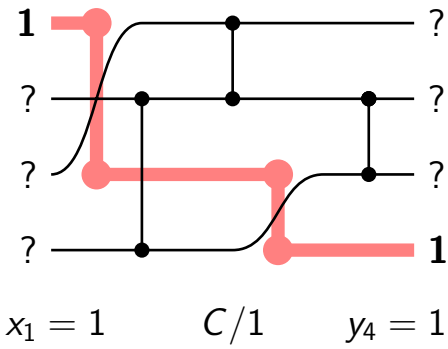
Van Voorhis' Bound 1/2

- Sorting network C of size s_n
- Take a one-hot sequence $\langle i \rangle$: $\langle i \rangle_i = 1$ and $\langle i \rangle_j = 0$ for $j \neq i$
- Trace the path taken by the 1
- Remove all comparators on that path: C/i



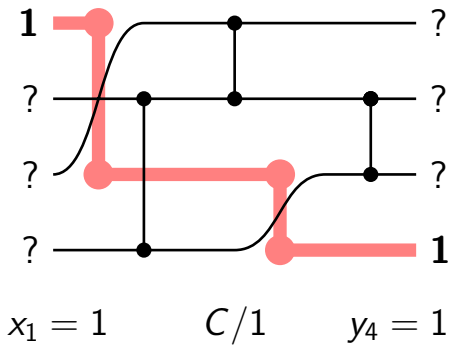
Van Voorhis' Bound 1/2

- Sorting network C of size s_n
- Take a one-hot sequence $\langle i \rangle$: $\langle i \rangle_i = 1$ and $\langle i \rangle_j = 0$ for $j \neq i$
- Trace the path taken by the 1
- Remove all comparators on that path: C/i
- Resulting network has $d_i(C)$ fewer comparators and sorts $n - 1$ inputs



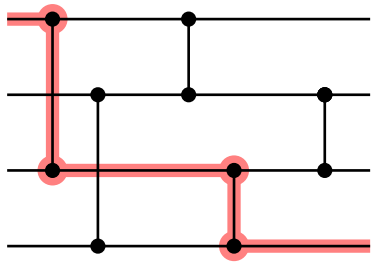
Van Voorhis' Bound 1/2

- Sorting network C of size s_n
- Take a one-hot sequence $\langle i \rangle$: $\langle i \rangle_i = 1$ and $\langle i \rangle_j = 0$ for $j \neq i$
- Trace the path taken by the 1
- Remove all comparators on that path: C/i
- Resulting network has $d_i(C)$ fewer comparators and sorts $n-1$ inputs
- $s_n \geq d_i(C) + s_{n-1}$



Van Voorhis' Bound 2/2

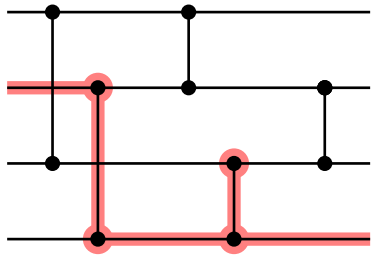
- Combining bounds for each channel: $s_n \geq \max_i d_i(C) + s_{n-1}$



C

Van Voorhis' Bound 2/2

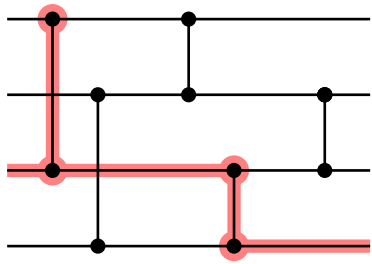
- Combining bounds for each channel: $s_n \geq \max_i d_i(C) + s_{n-1}$



C

Van Voorhis' Bound 2/2

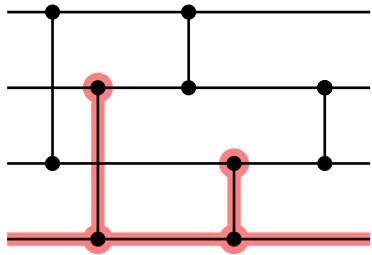
- Combining bounds for each channel: $s_n \geq \max_i d_i(C) + s_{n-1}$



C

Van Voorhis' Bound 2/2

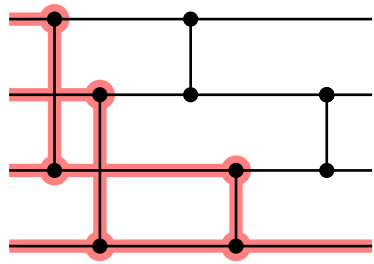
- Combining bounds for each channel: $s_n \geq \max_i d_i(C) + s_{n-1}$



C

Van Voorhis' Bound 2/2

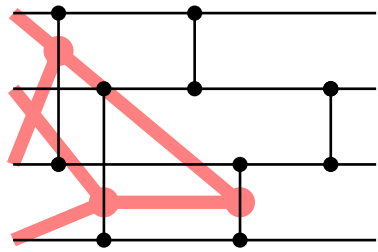
- Combining bounds for each channel: $s_n \geq \max_i d_i(C) + s_{n-1}$
- All these paths form a binary tree with n leaves



C

Van Voorhis' Bound 2/2

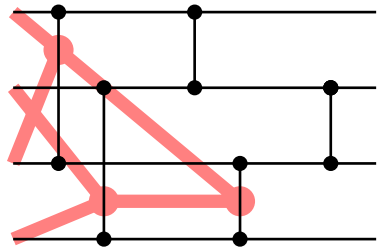
- Combining bounds for each channel: $s_n \geq \max_i d_i(C) + s_{n-1}$
- All these paths form a binary tree with n leaves
- Inner vertices are comparators so $d_i(C)$ are the leaves' depths



C

Van Voorhis' Bound 2/2

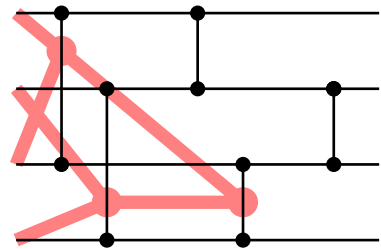
- Combining bounds for each channel: $s_n \geq \max_i d_i(C) + s_{n-1}$
- All these paths form a binary tree with n leaves
- Inner vertices are comparators so $d_i(C)$ are the leaves' depths
- A binary tree with n leaves height at least $\lceil \log_2 n \rceil$



C

Van Voorhis' Bound 2/2

- Combining bounds for each channel: $s_n \geq \max_i d_i(C) + s_{n-1}$
- All these paths form a binary tree with n leaves
- Inner vertices are comparators so $d_i(C)$ are the leaves' depths
- A binary tree with n leaves height at least $\lceil \log_2 n \rceil$
- $s_n \geq s_{n-1} + \lceil \log_2 n \rceil$



C

New Bound on Partial Sorting Networks

Generalizing Van Voorhis' Bound?

n	1	2	3	4	5	6	7	8	9	10	11	12
s_n	0	**1**	**3**	**5**	9	**12**	16	**19**	25	**29**	35	**39**
		↑	↑	↑		↑		↑		↑		↑

- Van Voorhis' bound is strict for more than half of the known values!
- Can we generalize it to partial sorting networks?

Pruned Sequence Set

- i is a prunable input of X if $\langle i \rangle \in X$

Pruned Sequence Set

- i is a prunable input of X if $\langle i \rangle \in X$
- x/i is subsequence of x with x_i removed
- Define $X/i = \{x/i \mid x \in X, x_i = 1\}$

  E.g. $B_n/i = B_{n-1}$, $\{000, 100, 010, 011, 111\}/2 = \{00, 01, 11\}$

  ✗ ✗ ✓ ✓ ✓

Pruned Sequence Set

- i is a prunable input of X if $\langle i \rangle \in X$
- x/i is subsequence of x with x_i removed
- Define $X/i = \{x/i \mid x \in X, x_i = 1\}$

  E.g. $B_n/i = B_{n-1}$, $\{\underset{\times}{000}, \underset{\times}{100}, \underset{\checkmark}{010}, \underset{\checkmark}{011}, \underset{\checkmark}{111}\}/2 = \{00, 01, 11\}$

- If C sorts X, C/i sorts X/i for prunable i of X

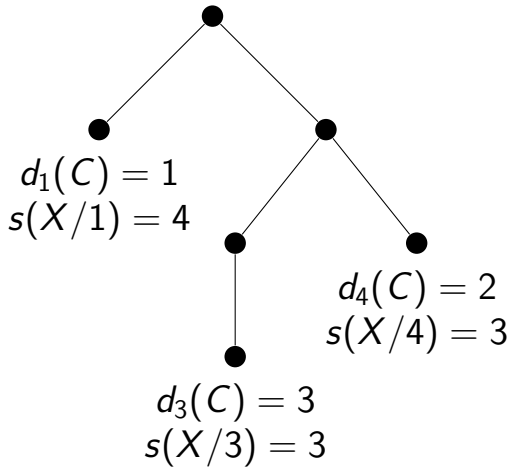
Bound on Partial Sorting Networks

- $s(X) \geq \max_{\langle i \rangle \in X}(d_i(C) + s(X/i))$

  for C on X with size $s(X)$

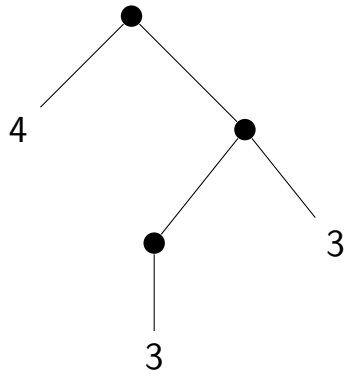
Bound on Partial Sorting Networks

- $s(X) \geq \max_{\langle i \rangle \in X}(d_i(C) + s(X/i))$
  for C on X with size $s(X)$



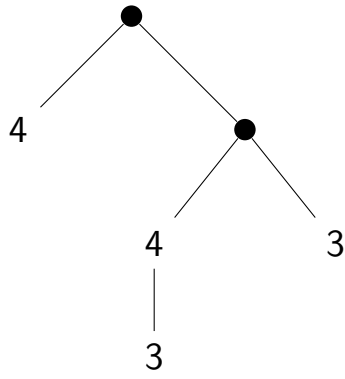
Bound on Partial Sorting Networks

- $s(X) \geq \max_{\langle i \rangle \in X}(d_i(C) + s(X/i))$
  for C on X with size $s(X)$
- Label leaves with $s(X/i)$



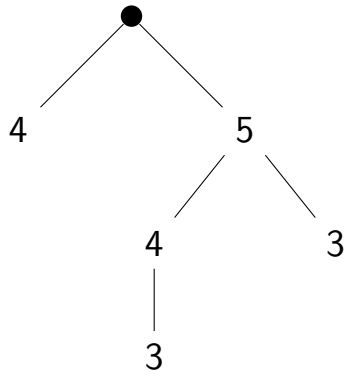
Bound on Partial Sorting Networks

- $s(X) \geq \max_{\langle i \rangle \in X}(d_i(C) + s(X/i))$
  for C on X with size $s(X)$
- Label leaves with $s(X/i)$
- Label inner vertices with
  $1 +$ max child label



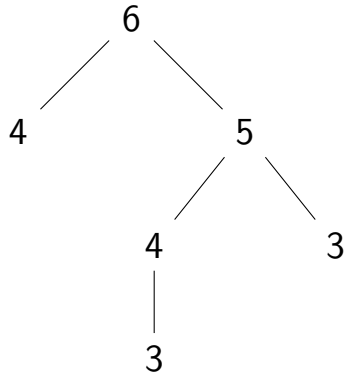
Bound on Partial Sorting Networks

- $s(X) \geq \max_{\langle i \rangle \in X}(d_i(C) + s(X/i))$
  for C on X with size $s(X)$
- Label leaves with $s(X/i)$
- Label inner vertices with
  $1 +$ max child label



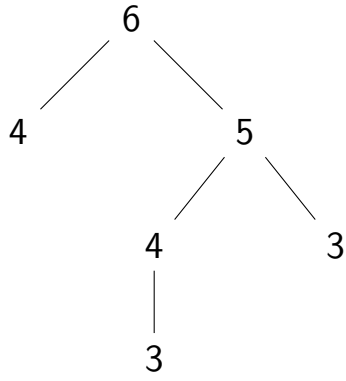
Bound on Partial Sorting Networks

- $s(X) \geq \max_{\langle i \rangle \in X}(d_i(C) + s(X/i))$
  for C on X with size $s(X)$
- Label leaves with $s(X/i)$
- Label inner vertices with $1 + \max$ child label
- Root is $\max_{\langle i \rangle \in X}(d_i(C) + s(X/i))$



Bound on Partial Sorting Networks

- $s(X) \geq \max_{\langle i \rangle \in X}(d_i(C) + s(X/i))$
  for C on X with size $s(X)$
- Label leaves with $s(X/i)$
- Label inner vertices with $1 + \max$ child label
- Root is $\max_{\langle i \rangle \in X}(d_i(C) + s(X/i))$
- Minimize over *all* trees with leaves $s(X/i)$



Huffman's algorithm

- A multiset of leaf values $v_1, \ldots v_k$
- A binary operation $a \star b$ used to label inner vertices

Huffman's algorithm

- A multiset of leaf values $v_1, \ldots v_k$
- A binary operation $a \star b$ used to label inner vertices, with

$$
\begin{aligned}
a &\leq a \star b \\
a \star b &= b \star a \\
(a \star b) \star (c \star d) &= (a \star c) \star (b \star d) \\
a \star c &\leq b \star c \qquad \text{if } a \leq b \\
(a \star b) \star c &\leq a \star (b \star c) \qquad \text{if } a \leq c
\end{aligned}
$$

Huffman Algebra (Knuth 1982)

Huffman's algorithm

- A multiset of leaf values $v_1, \ldots v_k$
- A binary operation $a \star b$ used to label inner vertices, with

$$
\begin{aligned}
a &\leq a \star b \\
a \star b &= b \star a \\
(a \star b) \star (c \star d) &= (a \star c) \star (b \star d) \\
a \star c &\leq b \star c \qquad \text{if } a \leq b \\
(a \star b) \star c &\leq a \star (b \star c) \qquad \text{if } a \leq c
\end{aligned}
$$

Huffman Algebra (Knuth 1982)

- Finds minimum root value $H_\star(v_1, \ldots, v_k)$ by repeatedly combining the two smallest values

Huffman Bound on Partial Sorting Networks

- $a \diamond b = 1 + \max\{a, b\}$ defines a Huffman algebra!

Huffman Bound on Partial Sorting Networks

- $a \diamond b = 1 + \max\{a, b\}$ defines a Huffman algebra!
- Optimal tree cannot have non-full vertices
  (Replace non-full nodes with their only child)

Huffman Bound on Partial Sorting Networks

- $a \diamond b = 1 + \max\{a, b\}$ defines a Huffman algebra!
- Optimal tree cannot have non-full vertices
  (Replace non-full nodes with their only child)
- $s(X) \geq H_{\diamond}(s(X/i_1), \ldots, s(X/i_k))$ for prunable $i_1, \ldots, i_k$ of X

Computer Search

Dynamic Programming

- Goal: Compute $s(X)$

Dynamic Programming

- Goal: Compute $s(X)$
- Reduce computing $s(Y)$ to computing several $s(Y_1), \ldots s(Y_k)$

Dynamic Programming

- Goal: Compute $s(X)$
- Reduce computing $s(Y)$ to computing several $s(Y_1), \ldots s(Y_k)$
- Apply recursively starting with $s(X)$, cache overlapping subproblems

Dynamic Programming

- Goal: Compute $s(X)$
- Reduce *improving bounds on* $s(Y)$ to *improving bounds on* several $s(Y_1), \ldots s(Y_k)$
- Apply recursively starting with $s(X)$, cache overlapping subproblems

Dynamic Programming

- Goal: Compute $s(X)$
- Reduce *improving bounds on* $s(Y)$ to *improving bounds on* several $s(Y_1), \ldots s(Y_k)$
- Apply recursively starting with $s(X)$, cache overlapping subproblems
- Maintain intervals $b(Y)$ so that $s(Y) \in b(Y)$

Dynamic Programming

- Goal: Compute $s(X)$
- Reduce *improving bounds on* $s(Y)$ to *improving bounds on* several $s(Y_1), \ldots s(Y_k)$
- Apply recursively starting with $s(X)$, cache overlapping subproblems
- Maintain intervals $b(Y)$ so that $s(Y) \in b(Y)$
- $b(Y) = \{0\}$ if "sorted", otherwise $b(Y) = \{1, \ldots, u\}$ with $u \geq s_n$

Dynamic Programming

- Goal: Compute $s(X)$
- Reduce *improving bounds on* $s(Y)$ to *improving bounds on* several $s(Y_1), \ldots s(Y_k)$
- Apply recursively starting with $s(X)$, cache overlapping subproblems
- Maintain intervals $b(Y)$ so that $s(Y) \in b(Y)$
- $b(Y) = \{0\}$ if "sorted", otherwise $b(Y) = \{1, \ldots, u\}$ with $u \geq s_n$
- Improve bounds until $b(X) = \{s(X)\}$

Canonicalization

- Equiv. modulo permutation & negation is a congruence for $s(X)$

Canonicalization

- Equiv. modulo permutation & negation is a congruence for $s(X)$
- Canonicalization procedure to map all X in one class to the same X'

Canonicalization

- Equiv. modulo permutation & negation is a congruence for $s(X)$
- Canonicalization procedure to map all X in one class to the same X'
- Less memory, more overlapping subproblems

Canonicalization

- Equiv. modulo permutation & negation is a congruence for $s(X)$
- Canonicalization procedure to map all X in one class to the same X'
- Less memory, more overlapping subproblems
- Represent X as labeled graph and use canonicalization
- Very efficient in practice (nauty & Traces by McKay & Piperno)

Updates

1. Goal: Improve a non-singleton $b(X)$
2. Lift the Huffman bound and the successor recurrence to intervals

Updates

1. Goal: Improve a non-singleton $b(X)$
2. Lift the Huffman bound and the successor recurrence to intervals
3. If the low Huffman bound is an improvement, apply it, return

Updates

1. Goal: Improve a non-singleton $b(X)$
2. Lift the Huffman bound and the successor recurrence to intervals
3. If the low Huffman bound is an improvement, apply it, return
4. If the high Huffman would be an improvement, recurse on some non-singleton $b(X/i)$, goto previous step

Updates

1. Goal: Improve a non-singleton $b(X)$
2. Lift the Huffman bound and the successor recurrence to intervals
3. If the low Huffman bound is an improvement, apply it, return
4. If the high Huffman would be an improvement, recurse on some non-singleton $b(X/i)$, goto previous step
5. Intersect $b(X)$ with the successor recurrence, return if improved

Updates

1. Goal: Improve a non-singleton $b(X)$
2. Lift the Huffman bound and the successor recurrence to intervals
3. If the low Huffman bound is an improvement, apply it, return
4. If the high Huffman would be an improvement, recurse on some non-singleton $b(X/i)$, goto previous step
5. Intersect $b(X)$ with the successor recurrence, return if improved
6. Recurse on some non-singleton $b(X[i,j])$, goto previous step

Results

- Computation of $s_{11} = 35$

Results

- Computation of $s_{11} = 35$
- Multi-threaded optimized implementation in Rust
- AMD EPYC 7401P 24-Core Processor (48 threads)

Results

- Computation of $s_{11} = 35$
- Multi-threaded optimized implementation in Rust
- AMD EPYC 7401P 24-Core Processor (48 threads)
- Took 4 h 51 min and 178 GB of RAM
- Computed partial sorting network bounds for 2 462 890 689 sets < 76 bytes per $b(X)$ entry

Results

- Computation of $s_{11} = 35$
- Multi-threaded optimized implementation in Rust
- AMD EPYC 7401P 24-Core Processor (48 threads)
- Took 4 h 51 min and 178 GB of RAM
- Computed partial sorting network bounds for 2 462 890 689 sets < 76 bytes per $b(X)$ entry
- Van Voorhis' bound gives us $s_{12} = 39$

Results

- Computation of $s_{11} = 35$
- Multi-threaded optimized implementation in Rust
- AMD EPYC 7401P 24-Core Processor (48 threads)
- Took 4 h 51 min and 178 GB of RAM
- Computed partial sorting network bounds for 2 462 890 689 sets < 76 bytes per $b(X)$ entry
- Van Voorhis' bound gives us $s_{12} = 39$
- Computing $s_9 = 25$, $s_{10} = 29$ takes less than 5 s and 40 MB of RAM on my laptop

Formal Verification

Certifcates

- Idea: Write a DAG of the exact steps to derive every intermediate bound
- Use a much simpler, verified program to check all edges

Certifcates

- Idea: Write a DAG of the exact steps to derive every intermediate bound
- Use a much simpler, verified program to check all edges
- Problem: Just storing the bounds requires over 90 GB

Certifcates

- Idea: Write a DAG of the exact steps to derive every intermediate bound
- Use a much simpler, verified program to check all edges
- Problem: Just storing the bounds requires over 90 GB
- Pairwise inclusion modulo symmetries check of all X with the same $s(X)$ to remove redundant bounds (52 h)

Certifcates

- Idea: Write a DAG of the exact steps to derive every intermediate bound
- Use a much simpler, verified program to check all edges
- Problem: Just storing the bounds requires over 90 GB
- Pairwise inclusion modulo symmetries check of all X with the same $s(X)$ to remove redundant bounds (52 h)
- Find and log a derivation for every remaining bound (20 h)

Certifcates

- Idea: Write a DAG of the exact steps to derive every intermediate bound
- Use a much simpler, verified program to check all edges
- Problem: Just storing the bounds requires over 90 GB
- Pairwise inclusion modulo symmetries check of all X with the same $s(X)$ to remove redundant bounds (52 h)
- Find and log a derivation for every remaining bound (20 h)
- Resulting certificate is below 3 GB and contains 12 659 079 steps

Formalization of the Problem

Formalization of s_n using Isabelle/HOL:

type-synonym *vect* = ‹*nat* ⇒ *bool*›

definition *fixed-width-vect* :: ‹*nat* ⇒ *vect* ⇒ *bool*› **where**
‹*fixed-width-vect n v* = (∀ *i* ≥ *n*. *v i* = *True*)›

type-synonym *cmp* = ‹*nat* × *nat*›

definition *apply-cmp* :: ‹*cmp* ⇒ *vect* ⇒ *vect*› **where**
‹*apply-cmp c v* = (
let (*a*, *b*) = *c*
in v(
a := *min* (*v a*) (*v b*),
b := *max* (*v a*) (*v b*)))›

definition *lower-size-bound* :: ‹*vect set* ⇒ *nat* ⇒ *bool*› **where**
‹*lower-size-bound V b* = (∀ *cn*. (∀ *v* ∈ *V*. *mono* (*fold apply-cmp cn v*)) ⟶ *length cn* ≥ *b*)›

definition *lower-size-bound-for-width* :: ‹*nat* ⇒ *nat* ⇒ *bool*› **where**
‹*lower-size-bound-for-width w b* = *lower-size-bound* {*v*. *fixed-width-vect w v*} *b*›

Formally Verified Checker

- Checker split into certificate parser and certificate checker
- Parser implemented in Haskell
- Checker implemented in Isabelle/HOL, extracted to Haskell

Formally Verified Checker

- Checker split into certificate parser and certificate checker
- Parser implemented in Haskell
- Checker implemented in Isabelle/HOL, extracted to Haskell
- 87-page correctness proof in Isabelle/HOL ending in:

lemma *check-proof-get-bound-spec*:
assumes ‹*check-proof-get-bound cert* = *Some* (*width*, *bound*)›
shows ‹*lower-size-bound-for-width* (*nat width*) (*nat bound*)›
using *assms* **by** (*rule Checker.check-proof-get-bound-spec*)

Formally Verified Checker

- Checker split into certificate parser and certificate checker
- Parser implemented in Haskell
- Checker implemented in Isabelle/HOL, extracted to Haskell
- 87-page correctness proof in Isabelle/HOL ending in:

lemma *check-proof-get-bound-spec*:
assumes ‹*check-proof-get-bound cert* = *Some* (*width*, *bound*)›
shows ‹*lower-size-bound-for-width* (*nat width*) (*nat bound*)›
using *assms* **by** (*rule Checker.check-proof-get-bound-spec*)

- Running the checker on the 11-input certificate prints `Just (11,35)` and took 3 h and below 8 GB on my laptop

- Sourcecode & formal proof: `github.com/jix/sortnetopt`
- Twitter: `@jix_`
- Email: `me@jix.one`
- Slides: `files.jix.one/s/20200707.pdf`

References

Donald E. Knuth. *The Art of Computer Programming: Volume 3: Sorting and Searching*. 2nd ed. Addison-Wesley, 1998.

Robert W. Floyd and Donald E. Knuth. "The Bose-Nelson Sorting Problem". In: *A Survey of Combinatorial Theory*. North-Holland Publishing Company, 1973.

David C. Van Voorhis. "An Improved Lower Bound for Sorting Networks". In: *IEEE Transactions on Computers* C-21.6 (1972).

Donald E. Knuth. "Huffman's Algorithm via Algebra". In: *Journal of Combinatorial Theory*, Series A 32.2 (1982).

Brendan D. McKay and Adolfo Piperno, "Practical Graph Isomorphism, II". In: *Journal of Symbolic Computation*, 60 (2014).

Michael Codish, Luís Cruz-Filipe, Michael Frank, and Peter Schneider-Kamp. "Sorting Nine Inputs Requires Twenty-Five Comparisons". In: *Journal of Computer and System Sciences* 82.3 (2016).