Depth-13 Sorting Networks for 28 Channels

Chengu Wang

wangchengu@gmail.com

Abstract

We establish new depth upper bounds for sorting networks on 27 and 28 channels, improving the previous best bound of 14 to 13. Our 28-channel network is constructed with reflectional symmetry by combining high-quality prefixes of 16- and 12-channel networks, extending them greedily one comparator at a time, and using a SAT solver to complete the remaining layers.

1 Introduction

A comparator network is a circuit in which each gate is a comparator that takes two inputs and produces the minimum value (on top) and maximum value (on bottom) as outputs, illustrated as a vertical line in network diagrams. As with any circuit, we are concerned with its size and depth. The size of a comparator network is the number of comparators it contains. The depth of a comparator network is the maximum number of comparators along any path from input to output. A sorting network is a comparator network that sorts its inputs into non-decreasing order.

For example, Figure 1 shows a sorting network with 4 channels of depth 3.
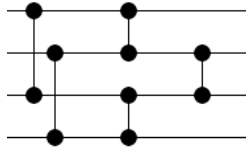

Figure 1: A sorting network for 4 channels of depth 3.

Finding optimal sorting networks is crucial for practical applications. Hardware implementations of sorting networks appear in GPU sorting, FPGA-based systems, and cryptographic protocols. Software-based sorting networks serve as building blocks for oblivious sorting algorithms used in secure computation and differential privacy systems.

Batcher developed two algorithms—odd-even mergesort and bitonic mergesort—that construct networks with depth $O((\log n)^2)$ and size $O(n(\log n)^2)$ [Bat68, Knu97]. A significant theoretical

breakthrough was later achieved by Ajtai, Komlós, and Szemerédi with their AKS network construction, which attains depth $O(\log n)$ and size $O(n \log n)$ [AKS83]. While these constructions are asymptotically optimal, the large constants hidden by the Big-O notation render them impractical for real-world applications. Furthermore, the theoretical understanding gained from studying small optimal networks can inform the design of more efficient asymptotic constructions.

For this reason, our work focuses on finding optimal-depth networks for specific small values of n. Table 1 shows the best-known lower and upper bounds on the depth of sorting networks for small input sizes n prior to our work.

Table 1: Known depth lower and upper bounds for $n = 1$ to 32

n	depth LB	depth UB	n	depth LB	depth UB
1	0 [Knu97]	0 [Knu97]	17	10 [CCFE$^+$19]	10 [EM14]
2	1 [Knu97]	1 [Knu97]	18	10 [CCFE$^+$19]	11 [AHB09]
3	3 [Knu97]	3 [Knu97]	19	10 [CCFE$^+$19]	11 [EM14]
4	3 [Knu97]	3 [Knu97]	20	10 [CCFE$^+$19]	11 [EM14]
5	5 [Knu97]	5 Shapiro [Knu97]	21	10 [CCFE$^+$19]	12 [AHB09]
6	5 [Knu97]	5 Shapiro [Knu97]	22	10 [CCFE$^+$19]	12 [AHB09]
7	6 [Knu97]	6 [Knu97]	23	10 [CCFE$^+$19]	12 [Ehl17]
8	6 [Knu97]	6 [Knu97]	24	10 [CCFE$^+$19]	12 [Ehl17]
9	7 [Par89]	7 Shapiro [Knu97]	25	10 [CCFE$^+$19]	13 [Dob]
10	7 [Par89]	7 Van Voorhis [Knu97]	26	10 [CCFE$^+$19]	13 [Dob]
11	8 [BZ14]	8 Shapiro [Knu97]	27†	10 [CCFE$^+$19]	14 Odd-Even Merge
12	8 [BZ14]	8 Shapiro [Knu97]	28†	10 [CCFE$^+$19]	14 Odd-Even Merge
13	9 [BZ14]	9 Van Voorhis [Knu97]	29	10 [CCFE$^+$19]	14 Odd-Even Merge
14	9 [BZ14]	9 Van Voorhis [Knu97]	30	10 [CCFE$^+$19]	14 Odd-Even Merge
15	9 [BZ14]	9 Van Voorhis [Knu97]	31	10 [CCFE$^+$19]	14 Odd-Even Merge
16	9 [BZ14]	9 Van Voorhis [Knu97]	32	10 [CCFE$^+$19]	14 Odd-Even Merge

$\dagger$ indicates the number of channels this paper is interested in.

The lower bounds for $n \leq 17$ are known to be tight. Look at the column "depth UB" in the right table. 10 appears once, 11 appears three times, 12 appears four times, 13 appears twice, and 14 appears six times. Numbers 10 and 13 are fewer than normal, and number 14 is more than normal. So, it suggests that depth upper bounds for $n = 18, 27$ and 28 might not be tight. Unfortunately, we were not able to improve the depth upper bound for 18 channels in this paper. This paper is for 27 and 28 channels.

A sorting network is reflection-symmetric if it is invariant under channel reflection (vertical reflection in the diagram). For example, Figure 1 is reflection-symmetric. Reflection-symmetric sorting networks are so powerful. For all the best-known sorting networks with minimum depths for even $n \leq 32$, there exists a reflection-symmetric sorting network with the same depth. The reflection-symmetry can be used to reduce the search space when improving the depth upper bound. SorterHunter [Dob] uses the reflection-symmetry and found many best-known depth upper bounds of sorting networks. Codish, Cruz-Filipe, Schneider-Kamp [CCFSK14], Bundala, and Závodnỳ [BZ14] also use it to reduce the search space of the prefixes.

Ehlers [Ehl17] improved the depth upper bound of the 24-channel networks to 12 layers. It stacked the prefixes of two 12-channel networks, and solved the remaining layers by a SAT solver.

Our construction of n=28 and d=13 is based on [Ehl17]. As we have more channels and one more layer, we use two new techniques to reduce the search space: assume the network is reflection-symmetric, and greedily extend the 24-channel prefix by one layer before solving the SAT problem. The full construction is found within 20 minutes on a common desktop computer.

2   Preliminaries and Redundant Prefixes

The search space for sorting networks grows exponentially with the number of channels, making exhaustive search computationally prohibitive for larger values of n. Consequently, researchers have developed various techniques to prune the search space and accelerate the discovery of optimal networks.

Lemma 1 (Zero-One Principle). *If a comparator network on n channels sorts all 2^n Boolean sequences into non-decreasing order, then it correctly sorts all sequences of arbitrary (comparable) values. [Knu97]*

The zero-one principle is a powerful result that allows us to verify the correctness of a sorting network by testing only the 2^n binary input sequences rather than $n!$ cases.

For comparator networks C and D with the same number of channels, denote $C \,\mathbin{\fatsemi}\, D$ the composition of C and D by putting D after C, i.e. for input x, $(C \,\mathbin{\fatsemi}\, D)(x) = D(C(x))$.

For a comparator network C, denote $\text{output}(C) = \{C(x) | x \in \{0,1\}^n\}$ the set of all possible outputs of C on Boolean inputs. The following lemma states that it suffices to consider prefixes P with minimal $\text{output}(P)$.

Lemma 2. *Let $P \,\mathbin{\fatsemi}\, S$ be a sorting network, and P' be a comparator network. If $\text{output}(P') \subseteq \text{output}(P)$, then $P' \,\mathbin{\fatsemi}\, S$ is a sorting network. [BZ14]*

To further reduce the search space, we use some symmetries: permutation and negation.

Lemma 3. *Let $P \,\mathbin{\fatsemi}\, S$ be a sorting network on n channels, and P' be a comparator network on n channels. If there exists a permutation $\sigma \in \mathbb{S}_n$ such that $\sigma(\text{output}(P')) \subseteq \text{output}(P)$, then there exists a comparator network S' with the same depth as S such that $P' \,\mathbin{\fatsemi}\, S'$ is a sorting network. [BZ14]*

We denote the bitwise-not operator by $\neg$. For example, $\neg 001_2 = 110_2$. When it is applied to a set of outputs, it flips each output, e.g. $\neg\{000_2, 001_2, 010_2\} = \{111_2, 110_2, 101_2\} = \{101_2, 110_2, 111_2\}$. Also, For an integer n, we denote $[n]$ the set $\{0, 1, \ldots, n-1\}$, and ρ_n the reflection permutation on $[n]$, i.e. $\rho_n(i) = n - 1 - i$.

Lemma 4. *If a comparator network S sorts input $x \in \{0,1\}^n$, then S sorts $\rho_n(\neg x)$.*

Bundala and Závodnỳ [BZ14] use this property to reduce the search space of the prefixes. We are giving a short proof here for completeness.

Proof. Denote $\neg S$ the network that changes all the comparators in S to max-min-comparators, which output max on top and min on bottom. Assume S sorts x. We flip every bit in the network, so $\neg S$ sorts $\neg x$ into the reverse order. Then, we vertically reflect the network along with the input x. Therefore, S sorts $\rho_n(\neg x)$. □

Putting them together, we get the following corollary that allows us to remove the redundant prefixes in the search space.

Corollary 1. *Let $P \mathbin{;} S$ be a sorting network on n channels, and P' be a comparator network on n channels. If there exists a permutation $\sigma \in \mathbb{S}_n$ such that $\sigma(\mathrm{output}(P')) \subseteq \mathrm{output}(P)$ or $\sigma(\neg \mathrm{output}(P')) \subseteq \mathrm{output}(P)$, then there exists a comparator network S' with the same depth as S such that $P' \mathbin{;} S'$ is a sorting network. [BZ14]*

The lemma states that if the outputs of P' (or their complements) are contained within the outputs of P up to permutation, then we can replace P with P' in the construction. This allows us to prune the search space by removing redundant prefixes.

In practice, it is not trivial to check if a network output set A is isomorphic to the subset of another one B. We use two techniques to speed up the check, one for positive cases, and one for negative cases.

Consider A as an $m_A \times n$ Boolean matrix, and $A_{i,j} \in \{0,1\}$ is the output on j-th channel of the i-th output. Similarly, we consider B as an $m_B \times n$ Boolean matrix. We sort the n columns of A and B by the column sum, respectively. If multiple columns have the same sum, we permute them randomly. If the rows of B covers the rows of A, then A is isomorphic to a subset of B. We repeat this random process a couple of times. In practice, it detects more than 90% of the redundant outputs efficiently.

To quickly filter negative cases, we compute the column sums and row sums of matrix A and B. Denote the row sum sequence of A by $\mathrm{RowSumA}_i$ for $i \in [m_A]$, and the column sum sequence of A by $\mathrm{ColSumA}_j$ for $j \in [n]$. Similarly, we compute $\mathrm{RowSumB}$ and $\mathrm{ColSumB}$. We sort $\mathrm{RowSumA}$, $\mathrm{RowSumB}$, $\mathrm{ColSumA}$, and $\mathrm{ColSumB}$, respectively. If A is isomorphic to a subset of B, then $\mathrm{ColSumA}_j \leq \mathrm{ColSumB}_j$ for all $j \in [n]$, and $\mathrm{RowSumA}$ is a subsequence of $\mathrm{RowSumB}$. This property is commonly used in bipartite graph isomorphism and embedding detections and sorting networks [CCFFSK16].

This paper focuses on reflection-symmetric sorting networks on even channels. We say a permutation $\sigma \in \mathbb{S}_n$ is reflection-symmetric if it is commutative with ρ_n, i.e. $\rho_n \circ \sigma = \sigma \circ \rho_n$. All these reflection-symmetric permutations form the centralizer of ρ_n in $\mathbb{S}_n$, denoted by $\mathcal{C}(\rho_n)$. It

is isomorphic to the wreath product of $\mathbb{C}_2$ and $\mathbb{S}_{n/2}$. In other words, we can permute the $n/2$ pairs (the $\mathbb{S}_{n/2}$ part), and independently flip each pair (the $\mathbb{C}_2$ part).

By applying the reflection-symmetry on Corollary 1, we obtain:

Corollary 2. *Let $P \,\mathring{,}\, S$ be a reflection-symmetric sorting network on n channels, and let P' be a reflection-symmetric comparator network on n channels. If there exists a reflection-symmetric permutation $\sigma \in \mathcal{C}(\rho_n)$ such that $\sigma(\mathrm{output}(P')) \subseteq \mathrm{output}(P)$ or $\sigma(\neg\,\mathrm{output}(P')) \subseteq \mathrm{output}(P)$, then there exists a reflection-symmetric comparator network S' with the same depth as S such that $P' \,\mathring{,}\, S'$ is a sorting network.*

We will use this corollary to eliminate redundant prefixes in Section 3. Specifically, if both P and P' are in our search space, we can remove P and keep only P'. The two speedup techniques described above also apply to the reflection-symmetric case by maintaining reflection symmetry at each step.

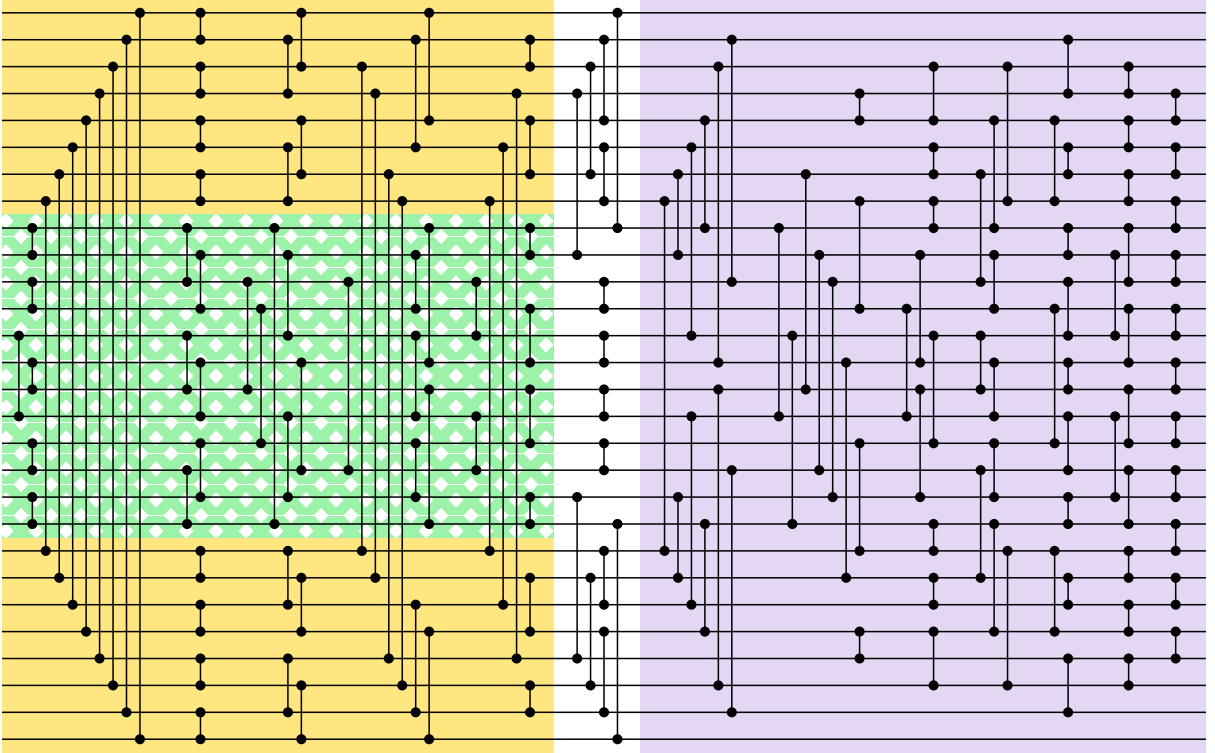# 3   Construction of Depth-13 Sorting Network for 28 Channels

Figure 2: A sorting network for 28 channels with 13 layers

We found a reflection-symmetric sorting network for 28 channels with 13 layers, shown in Figure 2 and listed below:

```
[(0,27),(1,26),(2,25),(3,24),(4,23),(5,22),(6,21),(7,20),(8,9),(10,11),(12,15),(13,14),(16,17),(18,19)]
[(0,1),(2,3),(4,5),(6,7),(8,10),(9,11),(12,14),(13,15),(16,18),(17,19),(20,21),(22,23),(24,25),(26,27)]
```

```
[(0,2),(1,3),(4,6),(5,7),(8,19),(9,12),(10,14),(11,16),(13,17),(15,18),(20,22),(21,23),(24,26),(25,27)]
[(0,4),(1,5),(2,20),(3,21),(6,24),(7,25),(8,13),(9,11),(10,17),(12,15),(14,19),(16,18),(22,26),(23,27)]
[(1,2),(3,24),(4,6),(5,22),(7,20),(8,9),(10,12),(11,13),(14,16),(15,17),(18,19),(21,23),(25,26)]
[(0,8),(1,4),(2,6),(3,9),(5,7),(10,11),(12,13),(14,15),(16,17),(18,24),(19,27),(20,22),(21,25),(23,26)]
[(1,10),(2,13),(4,8),(5,12),(6,9),(7,20),(14,25),(15,22),(17,26),(18,21),(19,23)]
[(3,4),(6,14),(7,11),(8,15),(9,17),(10,18),(12,19),(13,21),(16,20),(23,24)]
[(2,4),(5,6),(7,8),(9,13),(11,15),(12,16),(14,18),(19,20),(21,22),(23,25)]
[(2,7),(4,8),(6,10),(9,11),(12,14),(13,15),(16,18),(17,21),(19,23),(20,25)]
[(1,3),(4,7),(5,6),(8,9),(10,12),(11,16),(13,14),(15,17),(18,19),(20,23),(21,22),(24,26)]
[(2,3),(4,5),(6,7),(8,10),(9,12),(11,13),(14,16),(15,18),(17,19),(20,21),(22,23),(24,25)]
[(3,4),(5,6),(7,8),(9,10),(11,12),(13,14),(15,16),(17,18),(19,20),(21,22),(23,24)]
```

Our construction strategy consists of three phases: we stack a high-quality 16-channel 5-layer prefix (shown in green in the figure) with a high-quality 12-channel 5-layer prefix (shown in yellow), extend to 6 layers by greedily adding one comparator at a time while maintaining a small set of promising candidates (shown in white), and finally use an SAT solver to figure out layers 7 through 13 (shown in purple).

Merging is most efficient when the two parts have approximately equal numbers of channels. For $n = 28$, both $14 + 14$ and $16 + 12$ are promising decompositions. Networks are typically more efficient when the number of channels is a power of 2 or a multiple thereof. We chose the $16 + 12$ decomposition based on these considerations.

3.1   High-Quality 16-Channel and 12-Channel 5-Layer Prefixes

We enumerate all the non-redundant reflection-symmetric 12-channel 5-layer prefixes by the generate-and-prune approach [CCFFSK16] one layer at a time.

Starting from the set containing only the empty network, we extend the prefix set by one layer by enumerating all possible reflection-symmetric layers, then prune the resulting set using Corollary 2. This process yields 2164 non-redundant 12-channel 5-layer prefixes.

prefix depth	0	1	2	3	4	5
# non-redundant prefixes	1	4	41	1502	11753	2164

We consider a prefix P high-quality if the size of output(P) is small. Among the 2164 non-redundant prefixes, we select the best 4 prefixes, which have output sizes of 34, 34, 35, and 35. All other prefixes have output sizes larger than 35.

For $n = 16$, enumerating all non-redundant 5-layer prefixes is computationally challenging. However, there exists an excellent choice: the first 5 layers of Van Voorhis's sorting network, shown in Figure 3. The first four layers construct a 4-dimensional hypercube structure on the 16 channels. The 5th layer compares channels with the same Hamming weight symmetrically. This structure becomes clearer when we write the 5th layer in binary format: (0001,0010),(0100,1000); (0011,1100),(0101,1010),(0110,1001); (0111,1011),(1101,1110). This design preserves as much information as possible by comparing corresponding keys [AHB09].

We found two other output sets isomorphic to this one but not isomorphic under reflection-symmetric permutations in $\mathcal{C}(\rho_{16})$.
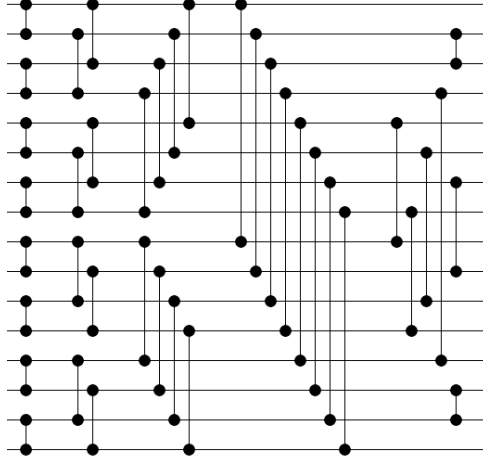

Figure 3: The first 5 layers of Van Voorhis's sorting network for 16 channels

With the best 4 prefixes for 12 channels and the best 2 prefixes for 16 channels, we combine them to obtain $4 \times 2 = 8$ prefixes for 28 channels with 5 layers.

3.2 Extending to 6 Layers

With 5 layers in hand, solving layers 6 through 13 directly with an SAT solver is too slow, even when restricting to reflection-symmetric solutions. However, solving layers 7 through 13 is feasible. Therefore, we extend the prefix by one additional layer. Rather than enumerating all possible symmetric layers on 28 channels—which would be prohibitively expensive—we extend the network one comparator at a time.

We begin with the 8 prefixes $\mathcal{P}_0$. At each step, we add one comparator along with its reflection (if not self-symmetric): $\mathcal{Q}_{i+1} = \{P \mathbin{\fatsemi} \{c, \rho_{28}(c)\} \mid P \in \mathcal{P}_i$ and c is a comparator connecting two unused channels in layer 6$\}$. We then prune $\mathcal{P}_i \cup \mathcal{Q}_{i+1}$ using Corollary 2 and retain only the best 64 prefixes $\mathcal{P}_{i+1}$—those with the smallest output sizes. We repeat this process until obtaining $\mathcal{P}_{14}$, the set of 64 prefixes with 6 layers.

3.3 SAT Solving the Remaining Layers

We now solve layers 7 through 13 using an SAT solver. We employ many optimization techniques from [CCFE$^+$19] to accelerate the SAT solving process, including improved SAT encodings with oneUp and oneDown techniques, constraints on the last two layers, and channel permutation to minimize the window size.

In practice, it suffices to solve for just the best 8 prefixes in $\mathcal{P}_{14}$. All 8 prefixes yield solutions. We then remove unused comparators—those whose first input is always less than or equal to

the second input. Figure 2 shows one such solution after removing unused comparators and permuting the input channels.

Theorem 1. *There exists a reflection-symmetric sorting network for 28 channels with 13 layers.*

Corollary 3. *There exists a sorting network for 27 channels with 13 layers.*

3.4   Implementation and Running Time

We ran our program on a Mac mini M2 with 16GB RAM. The entire computation takes less than 20 minutes to find the solution shown in Figure 2. Specifically, finding the best 4 prefixes for 12 channels with 5 layers takes 7 minutes, while finding the best 2 prefixes for 16 channels with 5 layers takes 1 minute. We use MiniSat to solve the 8 SAT instances in parallel. It takes half a minute to find one solution, and 5 minutes to solve all 8 problems. All other steps complete very quickly.

The complete program is available at `https://github.com/wcgbg/sorting-network-n28d13`.

4   Conclusion

In this paper, we improved the depth upper bound of sorting networks for 27 and 28 channels from 14 to 13. Our approach employs two key techniques to reduce the search space: (1) restricting the search to reflection-symmetric networks, and (2) extending the 6th layer incrementally by adding one comparator at a time while maintaining a small set of high-quality candidates.

Future work could explore more sophisticated prefix selection strategies, such as using reinforcement learning to predict which prefixes are most likely to yield solutions. In this paper, we used minimum output size as the objective function in the greedy extension step. A machine learning model could potentially identify better objective functions that lead to more efficient discovery of optimal networks.

References

[AHB09] Sherenaz Waleed Al-Haj Baddar. *Finding better sorting networks*. PhD thesis, Kent State University, 2009.

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. An o(n log n) sorting network. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 1–9, 1983.

[Bat68] Kenneth E Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 307–314, 1968.

[BZ14] Daniel Bundala and Jakub Závodnỳ. Optimal sorting networks. In *International Conference on Language and Automata Theory and Applications*, pages 236–247. Springer, 2014.

[CCFE+19] Michael Codish, Luís Cruz-Filipe, Thorsten Ehlers, Mike Müller, and Peter Schneider-Kamp. Sorting networks: To the end and back again. *Journal of Computer and System Sciences*, 104:184–201, 2019.

[CCFFSK16] Michael Codish, Luís Cruz-Filipe, Michael Frank, and Peter Schneider-Kamp. Sorting nine inputs requires twenty-five comparisons. *Journal of Computer and System Sciences*, 82(3):551–563, 2016.

[CCFSK14] Michael Codish, Luís Cruz-Filipe, and Peter Schneider-Kamp. The quest for optimal sorting networks: Efficient generation of two-layer prefixes. In *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 359–366. IEEE, 2014.

[Dob] Bert Dobbelaere. Sorterhunter. `https://github.com/bertdobbelaere/SorterHunter`.

[Ehl17] Thorsten Ehlers. Merging almost sorted sequences yields a 24-sorter. *Information Processing Letters*, 118:17–20, 2017.

[EM14] Thorsten Ehlers and Mike Müller. Faster sorting networks for 17, 19 and 20 inputs. *arXiv preprint arXiv:1410.2736*, 2014.

[Knu97] Donald Ervin Knuth. *The art of computer programming*, volume 3. Pearson Education, 1997.

[Par89] Ian Parberry. A computer assisted optimal depth lower bound for sorting networks with nine inputs. In *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, pages 152–161, 1989.