hexagon:2610.00549v2

SORTING NETWORKS ON 13 INPUTS NEED 45 COMPARATORS

TEJASVI SINGH TOMAR

Abstract. Let $S(N)$ be the least number of comparators in a sorting network on N inputs (the Bose–Nelson problem). We prove $S(13) \geq 45$, so Juillé's network is optimal and $S(13) = 45$. We also prove $P(2, N) = \lceil \log_2 F(N) \rceil$ for $3 \leq N \leq 32$, where $P(2, N)$ is the least worst-case number of comparators met by the two largest inputs in a network that delivers them to fixed outputs, and F is an explicit function of Van Voorhis. This proves the twenty tabulated lower bounds on $S(N)$ for $13 \leq N \leq 32$, which rested on an inequality of Van Voorhis whose published proof is invalid, and $S(13) = 45$ raises each of them by one. Both proofs use computer searches. Harder's method for $S(11) = 35$ applies Van Voorhis's one-value argument to the set of 0-1 vectors that a prefix of a network produces; for $S(13) \geq 45$ we extend the method from the largest value to the two largest. A search over prefixes bounds the rest of each network by deleting the comparators that the two largest, or the two smallest, inputs meet in it; the costs of the remaining sets are bounded using $S(11) = 35$ and exact values computed with Harder's program. The search writes a certificate, which is checked by a separate verifier and by a kernel-checked proof in Lean 4.

1. Introduction

A comparator network on N inputs is a circuit of comparators, each of which outputs the larger and the smaller of its two input values on two separate leads. A sorting network delivers every input sequence in sorted order. The Bose–Nelson problem [1] asks for $S(N)$, the least number of comparators in a sorting network on N inputs. Floyd and Knuth determined $S(N)$ for $N \leq 8$ [6], Codish, Cruz-Filipe, Frank and Schneider-Kamp proved $S(9) = 25$ and $S(10) = 29$ [2], and Harder proved $S(11) = 35$ and $S(12) = 39$ [8]. Harder's bound $S(11) \geq 35$ is backed by a certificate, which is checked by a program verified in Isabelle/HOL [7, 9]. Such certification goes back to the proof of $S(9) = 25$: Cruz-Filipe and Schneider-Kamp checked its computation with a formally verified program [3], and Cruz-Filipe, Larsen and Schneider-Kamp later extended this work [4]. For 13 inputs the smallest known sorting network, found by Juillé [11], has 45 comparators, and Van Voorhis's one-value bound $S(N) \geq S(N-1) + \lceil \log_2 N \rceil$ [22] gives $S(13) \geq 43$. We close this gap. Throughout, a dagger (†) marks a statement whose proof includes a computation.

Theorem 1.1. $S(13) = 45$.†

The upper bound is Juillé's network. Both bounds are also proved in Lean 4 (Appendix A).

Two values. In a chapter published the same year as the one-value bound [23], Van Voorhis followed the two largest inputs instead of one. Put the largest value on input x and the second largest on input y. The comparators that these two values meet can be deleted, and the rest of the network sorts the other $N - 2$ inputs. This gives

$$S(N) \;\geq\; S(N-2) + P(2, N), \tag{1}$$

Date: 17 November 2026.

2020 *Mathematics Subject Classification.* Primary 68P10; Secondary 68Q06, 05C05, 68V05, 68V20.

Key words and phrases. Sorting networks, Bose–Nelson problem, comparator networks, lower bounds, computer-assisted proof, formal verification, Lean theorem prover.

where $P(2, N)$ is the least possible worst-case number of comparators met by the two values, over all networks that deliver them to fixed outputs (Section 2). Van Voorhis stated that $P(2, N) \geq \lceil \log_2 F(N) \rceil$ for an explicit function F (Section 2.4). With $F(13) = 392$ and $S(11) = 35$ this gives $S(13) \geq 35 + 9 = 44$. Dobbelaere's table of sorting networks [5] has listed 44 as the lower bound for 13 inputs, together with lower bounds for $14 \leq N \leq 32$, with the note "on suggestion from Jelmer Firet and based on principles in" Van Voorhis's chapter. In an audit of the chapter [15], however, Mericanii showed that its proof of $P(2, N) \geq \lceil \log_2 F(N) \rceil$ is invalid. The proof identifies the comparators on the paths of the maximum with the $N - 1$ comparators at which two such paths merge. This fails when a path of the maximum passes a comparator whose other input never carries the maximum; Mericanii calls such a comparator a *pass-through* [15, Definition 1]. Pass-throughs occur already in a size-optimal network on 3 inputs. Mericanii proves the inequality for two classes of networks and leaves it open in general. The same work derives the tabulated lower bounds for $13 \leq N \leq 32$ from (1), the one-value bound and $P(2, N) \geq \lceil \log_2 F(N) \rceil$, and finds that each of them exceeds what the one-value bound alone gives. Each of these twenty values therefore depends on the disputed inequality.

Instead of repairing Van Voorhis's argument, we compute $P(2, N)$ for $3 \leq N \leq 32$.

Theorem 1.2. $P(2, N) = \lceil \log_2 F(N) \rceil$ *for* $3 \leq N \leq 32$.[†]

The upper bound comes from an explicit network for each N (Proposition 2.6). The lower bound rests on a computation, and two observations keep it small. First, the second maximum can re-enter the tree of paths of the maximum only through pass-throughs. After the deletions of Lemma 3.5, the numbers of comparators met by the two values, for all $N(N-1)$ choices of $x \neq y$, are therefore determined by a small amount of data (Theorem 3.8): the tree, the number of pass-throughs on each of its edges, and, for each place where the second maximum leaves the tree, the pass-through at which it re-enters the tree, or its output if it never does, with the number of comparators it meets on the way. Second, the paths of the maximum are short in any network in which the two values meet few comparators for every choice of $x \neq y$ (Lemma 3.10), which bounds the height of the tree. The proofs in Lean cover Theorem 1.2 only for $3 \leq N \leq 13$ (Appendix A.6).

Two values in the suffix. By Theorem 1.2, inequality (1) proves the bound $S(13) \geq 44$ above, and no more, since $P(2, 13) = 9$ is exact. It treats the comparators that the two values do not meet only through the single number $S(11)$. Harder [8] applied the one-value argument to the suffix of a network rather than to the whole network, and introduced partial sorting networks, which need to sort only the set of vectors that a prefix produces. The Huffman bound [8, Theorem 26] generalizes Van Voorhis's one-value bound to these networks: it bounds the cost of a set produced by a prefix in terms of the costs of its pruned sets. To reach 45 we apply the two-value argument to the suffix in the same way. Let T be a sorting network with at most 44 comparators, and let X be the set of 0-1 vectors produced by its first k comparators. The remaining comparators sort X, and there are at most $44 - k$ of them. Suppose that X contains the vector with a single 1 on a channel x and the vector with 1s on x and on another channel y (Definition 5.5 gives the precise condition). Run the suffix with the largest value on x and the second largest on y. Deleting the comparators that these two values meet leaves a network that sorts a set Y_{xy} on 11 channels, the *core* of the pair. The suffix therefore meets the two values at most $44 - k - s(Y_{xy})$ times in the run from (x, y), where $s(Y)$ denotes the least size of a network that sorts Y. Different pairs now have different budgets. These budgets form a *weighted instance* (Definition 6.1), which is *feasible* if a single network delivers the two values to the same two outputs from each of these pairs, within the budget of each pair. If it is infeasible, the prefix cannot be completed within 44 comparators.

Deciding a weighted instance is a finite problem: the structure theory for all pairs of inputs extends to an arbitrary set of pairs, and a computer search over the possible configurations decides such instances. The costs of the cores are bounded below through $S(11) = 35$ and, more sharply, by the same test one level down, on 11 and then 9 channels, with exact costs of sets on 9 and 7 channels computed by Harder's program [9].

Following the generate-and-prune approach of [2], a breadth-first search over prefixes builds one layer per comparator and identifies sets up to permutations of the channels and complementation. It discards the set of a prefix of length k when its weighted instance with budget $44 - k$ is infeasible for the two largest values or, symmetrically, for the two smallest (the two polarities of Definition 5.2). The layers grow and then shrink, and layer 25 is empty.

Larger networks. Theorems 1.1 and 1.2 give lower bounds for larger N as well.

Corollary 1.3. *For $13 \le N \le 32$, $S(N) \ge \Lambda(N) + 1$, where $\Lambda(N)$ is the value in Table 4.*[†]

The values $\Lambda(N)$ are the twenty tabulated lower bounds discussed above.

Organization. Sections 2 to 4 prove Theorem 1.2, Sections 5 to 13 prove Theorem 1.1 and describe its verification, and Section 14 proves Corollary 1.3. Appendix A describes the proofs in Lean. The programs, certificates and logs are in the supplements [17, 18], and the kernel-checked proofs of both bounds in [19].

2. Comparator networks and the two-value bound

In this section we fix the model, prove the two-value bound $S(N) \ge S(N-2) + P(2, N)$ and the monotonicity of $P(2, N)$, and construct the networks that give the upper bound in Theorem 1.2.

2.1. **Comparator networks.** A *comparator network* T on a finite set A of channels is a finite acyclic circuit made of input leads, one for each channel, and of comparators. A comparator g has two input leads and two output leads, $\mathrm{hi}(g)$ and $\mathrm{lo}(g)$. Each lead is either an input lead of T or an output lead of exactly one comparator, and it enters at most one comparator. The leads that enter no comparator are the *outputs* of T, and there are $|A|$ of them. Given values on the input leads, each comparator puts the larger of its two input values on $\mathrm{hi}(g)$ and the smaller on $\mathrm{lo}(g)$. The size $|T|$ is the number of comparators. We write $[n] = \{0, \dots, n-1\}$ and speak of a network on n inputs when $|A| = n$; renaming the channels then gives $A = [n]$. Such a network is a *sorting network* if its outputs can be listed as $o_0, \dots, o_{n-1}$ so that, for all distinct input values, o_i carries the value of rank i, counted from 0.

Drawing T on channels along a topological order of its comparators gives a sequence of standard comparators, defined below, followed by a permutation of the channels [8, Section 2]. Conversely, a sequence of comparators on channels is a network: the comparator (i, j), for channels $i \ne j$, puts the maximum on channel i and the minimum on channel j, and the outputs are identified with the channels. The comparator (i, j) is *standard* if $i > j$. A sorting network in the circuit model can be untangled into a standard sorting network with the same number of comparators [12, Exercise 5.3.4–16], so $S(N)$ is the same in both models.

2.2. **Runs and deleted paths.** For distinct channels x and y, the *run* (x, y) puts the largest value M on x, the second largest value m on y, and smaller distinct values on the other inputs. The value M leaves every comparator it enters on hi. The value m leaves a comparator on lo if M is on the other input lead, and on hi otherwise. Hence the leads that carry M and m depend only on (x, y). We write Π_x for the set of leads that carry M, which depends only on x, $\delta(x)$ for the number of comparators that M enters, and $W(x, y)$ for the set of comparators with M or m on an input lead. A comparator that m enters while M is on its other input lead is a *meeting*.

Deleting a path is the operation of [8, Definition 13]. For a channel x, the network T/x on $A \setminus \{x\}$ is obtained from T by deleting the leads of Π_x and the comparators that they enter, and joining the other input lead of each such comparator to its lo output lead. The input lead x and the output lead at the end of Π_x disappear.

Lemma 2.1. *Let $x \neq y$. In the run (x, y) the value m never occupies a lead of Π_x. The comparators of $W(x, y)$ are the $\delta(x)$ comparators that M enters and the comparators of T/x on the path that the maximum of T/x takes from y, which are the comparators that m enters and that are not meetings. Deleting this path from T/x gives the network $(T/x)/y$, which has $|T| - |W(x, y)|$ comparators.*

Proof. Since M occupies every lead of Π_x in the run (x, y), the value m occupies none of them. When m enters a comparator g of Π_x, the value M is on the other input lead, and m leaves on $\mathrm{lo}(g)$, which is where the join of T/x leads. At any other comparator m leaves on hi, as the maximum of T/x does. Thus the leads of m are those of the path of the maximum of T/x from y, with the joins expanded, and the comparators that m enters outside Π_x are the comparators of T/x on that path. □

2.3. The quantity $P(2, N)$ and the two-value bound.

Definition 2.2. A comparator network on $N \geq 2$ inputs is *top-two correct* if it has outputs $o_M \neq o_m$ such that in each run (x, y) the value M ends on o_M and m ends on o_m. For such a network T let $p_2(T) = \max_{x \neq y} |W(x, y)|$. Let $P(2, N)$ be the least value of $p_2(T)$ over all top-two correct networks T on N inputs.

A sorting network is top-two correct, with $o_M = o_{N-1}$ and $o_m = o_{N-2}$. The next lemma is the two-value bound. It is equation (2) of [23] (see [15, Section 2]). Its proof extends Harder's argument for one value [8, Lemma 16] to two values.

Lemma 2.3 (two-value pruning)**.** *Let T be a sorting network on $N \geq 3$ inputs. Then $|T| \geq S(N-2) + p_2(T)$. Hence $S(N) \geq S(N-2) + P(2, N)$.*

Proof. We fix $x \neq y$ and call a lead *special* if it carries M or m in the run (x, y). A comparator with two special input leads has two special output leads. A comparator with exactly one special input lead sends the special value to one output lead and the value of its other input lead to the other output lead, which is not special. We delete the comparators of $W(x, y)$ and the special leads, and for each deleted comparator with one special input lead we join its other input lead to its output lead that is not special. The result T' is a comparator network with $|T| - |W(x, y)|$ comparators whose inputs are the $N - 2$ inputs of T other than x and y. Let z be distinct values on these inputs, and run T on z together with M on x and m on y, where $M > m > \max z$. Induction along a topological order shows that each lead of T' carries the same value in T' as in T. Since T sorts, M and m end on o_{N-1} and o_{N-2}, and $o_0, \ldots, o_{N-3}$ carry z in increasing order. Therefore T' is a sorting network on $N - 2$ inputs, and $|T| - |W(x, y)| \geq S(N-2)$. Taking (x, y) with $|W(x, y)| = p_2(T)$ gives the first inequality, and applying it to a sorting network with $S(N)$ comparators gives the second. □

Lemma 2.4 (monotonicity)**.** *$P(2, N) \leq P(2, N+1)$ for $N \geq 2$.*

Proof. Let T be a top-two correct network on $N + 1$ inputs with $p_2(T) = P(2, N+1)$, and fix an input w. A value below all others on w leaves every comparator it enters on lo, so it follows a fixed path. Delete each comparator on this path and join its other input lead to its hi output lead. The result T' is a comparator network on the N inputs other than w. In a run (x, y) of T', the values M and m follow the same leads as in the run (x, y) of T with the smallest value on w, except that they skip the deleted comparators, which they would have left on hi in T as well.

Hence M and m end on o_M and o_m, the network T' is top-two correct, and each of its runs meets at most as many comparators as the same run in T. This gives $P(2,N) \le p_2(T') \le p_2(T)$. $\square$

Lemma 2.5 (Kraft)**.** *Consider rooted trees in which each vertex has zero or two children.*

(a) If the leaves of such a tree have depths $d_1, \dots, d_k$, then $\sum_i 2^{-d_i} = 1$.

(b) If integers $d_1, \dots, d_k \ge 0$ satisfy $\sum_i 2^{-d_i} = 1$, some such tree has leaves at depths $d_1, \dots, d_k$.

(c) If $k \ge 1$ and integers $\ell_1, \dots, \ell_k \ge 0$ satisfy $\sum_i 2^{-\ell_i} \le 1$, some such tree has k leaves at depths $d_i \le \ell_i$.

Proof. (a) Replacing two sibling leaves at depth d by their parent at depth $d-1$ preserves the sum, so induction on the number of vertices applies. (b) If $k = 1$ then $d_1 = 0$. Otherwise let $D = \max_i d_i \ge 1$. Multiplying the equation by 2^D shows that the number of indices with $d_i = D$ is even, so we may replace two of them by one index with value $D-1$, build a tree by induction, and give that leaf two children. (c) While the sum is below 1, decrease a largest ℓ_i by one. The sum is a multiple of $2^{-\max \ell_i}$, so it stays at most 1. When it reaches 1, apply (b). $\square$

2.4. **Van Voorhis's function and the upper bound.** Let B be a rooted tree with $N \ge 2$ leaves in which each vertex has zero or two children. We call such a tree a *binary tree*. For a vertex u of B let $D(u)$ be its depth, the root having depth 0, and for an internal vertex c let $h_1(c)$ and $h_2(c)$ be the heights of the subtrees at its two children, a leaf having height 0. Put

$$a(c) = D(c) + 1 + h_1(c) + h_2(c), \qquad f(B) = \sum_c 2^{a(c)}, \qquad F(N) = \min_B f(B),$$

where c ranges over the internal vertices of B and B over the binary trees with N leaves. Van Voorhis introduced the function F [23]. The closed form above is taken from Mericanii [15, Definition 1], who checks in [15, Section 2] that f satisfies the recursion of [23, Theorem 1] and that $\min_B f(B)$ reproduces Van Voorhis's table of $F(N)$ for $N \le 16$. Like Mericanii, we take this closed form as the definition of F for all N. Table 1 lists $F(N)$ for $N \le 32$. We computed it by a dynamic program over the pairs (height, f), and the values for $13 \le N \le 32$ agree with [15, Section 4.2].

N	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$F(N)$	8	16	36	52	80	96	168	200	256	288	392	424	480	512	784
$\lceil \log_2 F(N) \rceil$	3	4	6	6	7	7	8	8	8	9	9	9	9	9	10
$\nu(N)$	3	4	5	6	6	6	7	8	8	8	8	8	8	8	9
N	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
$F(N)$	848	960	1024	1232	1296	1408	1472	1872	1936	2048	2112	2320	2384	2496	2560
$\lceil \log_2 F(N) \rceil$	10	10	10	11	11	11	11	11	11	11	12	12	12	12	12
$\nu(N)$	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10

TABLE 1. Van Voorhis's function $F(N)$, the value $\lceil \log_2 F(N) \rceil$, and the bound $\nu(N) = \lceil \log_2 N \rceil + \lceil \log_2 (N-1) \rceil$ of Corollary 3.11.

Proposition 2.6. *For each $N \ge 2$ there is a top-two correct network on N inputs with $2N-3$ comparators and $p_2 \le \lceil \log_2 F(N) \rceil$. Hence $P(2,N) \le \lceil \log_2 F(N) \rceil$.*

Proof. Let B be a binary tree with $f(B) = F(N)$, and let $K = \lceil \log_2 F(N) \rceil$. The leaves of B are the inputs, and each internal vertex c of B is a comparator whose input leads are the input leads or the hi outputs at its two children. The hi output of the root is o_M. In the run (x,y) the value M passes the $D(x)$ comparators above x. The value m passes the comparators above y

up to $c = \mathrm{lca}_B(x, y)$, meets M at c and leaves on $\mathrm{lo}(c)$. These two paths contain at most $a(c)$ comparators, because x and y lie below different children of c. Each $a(c)$ is at most K, and $\sum_c 2^{-(K-a(c))} = 2^{-K} f(B) \leq 1$. By Lemma 2.5(c) there is a binary tree R whose leaves are the $N-1$ leads $\mathrm{lo}(c)$, at depths at most $K - a(c)$. Each internal vertex of R is a comparator whose input leads are the leads at its children, and the lead at the root of R is o_m. The value M never enters R, so m leaves each comparator of R on hi and reaches o_m after at most $K - a(c)$ of them. So $|W(x, y)| \leq K$. The network has $(N-1) + (N-2)$ comparators. □

3. The structure of top-two correct networks

In this section we show that the costs of all runs of a top-two correct network are determined by a small amount of data; the search of Section 4 enumerates these data. Section 7 extends the argument from all pairs of inputs to the pairs of a weighted instance.

3.1. **The tree of maximum paths.** We use the red and blue leads of [15, Definition 1]: a lead is *red* if it lies in Π_x for some input x, and *blue* otherwise. The first lemma uses only the behaviour of M, so we state it for networks in which the maximum alone is delivered to a fixed output. It is also applied to pruned networks in Lemma 3.10.

Lemma 3.1 (the tree of maximum paths)**.** *Let T be a comparator network on $n \geq 2$ inputs in which the largest value ends on the same output o_M whichever input carries it.*

- *(a) Input leads are red, each* lo *output is blue, and* $\mathrm{hi}(g)$ *is red if and only if g has a red input lead.*
- *(b) Join each red lead $\ell \neq o_M$ to* $\mathrm{hi}(g)$*, where g is the comparator that ℓ enters. Then the red leads form a rooted tree with root o_M whose leaves are the n input leads.*
- *(c) Call a comparator a* branch node *if both of its input leads are red, a* pass-through *if exactly one is, and* blue *if neither is. There are exactly $n-1$ branch nodes. Contracting the vertices with one child in the tree of* (b) *gives a binary tree B whose leaves are the inputs and whose internal vertices are the branch nodes, and each pass-through lies on an edge of B or on the edge from the root of B to o_M.*

Proof. (a) The input lead x lies in Π_x. The maximum leaves each comparator on hi, so no lo output is red. If an input lead of g lies in Π_x, the maximum enters g on it and leaves on $\mathrm{hi}(g)$, so $\mathrm{hi}(g) \in \Pi_x$. Conversely, if $\mathrm{hi}(g) \in \Pi_x$, the maximum entered g on an input lead, which lies in Π_x.

(b) Let $\ell \in \Pi_x$. If ℓ enters a comparator g, then $\mathrm{hi}(g)$ is the next lead of Π_x. Otherwise ℓ is an output that carries the maximum, so $\ell = o_M$. Since each lead enters at most one comparator, each red lead other than o_M has exactly one successor, and since the circuit is acyclic, following successors from any red lead ends at o_M. A red lead that is not an input lead is $\mathrm{hi}(g)$ for a comparator g with a red input lead, by (a), and that input lead is a predecessor. An input lead is the output of no comparator, so it has no predecessor. Thus the leaves are the n input leads.

(c) A vertex $\mathrm{hi}(g)$ of the tree has one child for each red input lead of g, so it has one or two. A rooted tree with n leaves in which each vertex other than a leaf has one or two children has exactly $n-1$ vertices with two children. Contracting the vertices with one child gives B. A pass-through g gives the vertex $\mathrm{hi}(g)$, which has one child, so it lies on an edge of B or above the root of B. □

From now on in this section, T is a top-two correct network on $N \geq 3$ inputs and B is its tree. For a vertex u of B (an input or a branch node), $\mathrm{seg}(u)$ denotes the edge from u to its parent, or from u to o_M if u is the root, and $k(u)$ denotes the number of pass-throughs on $\mathrm{seg}(u)$. We number them $(u, 1), \ldots, (u, k(u))$ from below. The branch nodes and the pass-throughs are the *red comparators*. For a red comparator v, let $u(v) = v$ if v is a branch node and $u(v) = u$

if $v = (u, i)$, and let $A(v)$ be the set of red comparators above v on the path to o_M, with v excluded. For an input x, the comparators on the path from x to o_M are those that M passes when it starts on x, so

$$\delta(x) = k(x) + \sum_u (1 + k(u)),$$

where u runs over the proper ancestors of x in B. For a branch node c and an input y below c, let $\mathrm{pre}(y, c) = k(y) + \sum_u (1 + k(u))$ over the vertices u strictly between y and c, the number of comparators on the path from y up to c, with c excluded. For a pass-through $q = (w, i)$ and a branch node e that is a proper ancestor of w, let

$$\mathrm{cl}(q, e) = 1 + \big(k(w) - i\big) + \sum_z (1 + k(z)),$$

where z runs over the vertices strictly between w and e. This is the number of red comparators from q up to e, with e excluded. A pass-through $q = (w, i)$ lies on the path from an input x to o_M exactly when w is x or an ancestor of x.

Remark 3.2. The depth of x in B is at most $\delta(x)$, and some leaf of B has depth at least $\lceil \log_2 N \rceil$, by Lemma 2.5(a). Together with the pruning of one path [8, Lemma 16] this gives the one-value bound $S(N) \geq S(N-1) + \lceil \log_2 N \rceil$ of Van Voorhis [22].

3.2. **The runs.**

Lemma 3.3 (the second maximum before the meeting)**.** *Let $x \neq y$ and $c = \mathrm{lca}_B(x, y)$.*

(a) In the run (x, y), the value m passes the comparators on the path from y up to c without meeting M, meets M at c and leaves c on $\mathrm{lo}(c)$.

(b) After c, m can meet M only at comparators of $A(c)$, and the leads of m after c depend only on c.

(c) $|W(x, y)| = \delta(x) + \mathrm{pre}(y, c) + \tau(c)$, where $\tau(c)$ is the number of comparators that m enters after leaving c and that do not belong to $A(c)$.

Proof. In the run (x, y) the leads that carry M are those of Π_x. When m enters a comparator g, the lead that carries m is not in Π_x, so m leaves on $\mathrm{lo}(g)$ exactly when the other input lead of g lies in Π_x, that is, when g is on the path from x to o_M.

(a) In the tree of Lemma 3.1(b) the paths from x and y to o_M meet at c and coincide above it. While m is on the path from y below c, it enters either c or a comparator below c on the side of y. Such a comparator is not on the path from x, so m leaves it on hi, which is the next lead on the path from y. The other input lead of c lies on the path from x, so m leaves c on $\mathrm{lo}(c)$.

(b) After c the value m is on leads that come after c in the circuit. The comparators on the path from x below c come before c, so m does not enter them. Hence the only comparators where m can meet M are those of $A(c)$, and by the first paragraph the leads of m after c are determined by $\mathrm{lo}(c)$ and $A(c)$.

(c) The run meets the $\delta(x)$ comparators on the path from x, the $\mathrm{pre}(y, c)$ comparators on the path from y below c, which are not on the path from x, and the comparators that m enters after c. By (b), those of the latter that lie on the path from x belong to $A(c)$. Since the circuit is acyclic, m enters no comparator twice. □

The same argument applies to every meeting. If m leaves a red comparator v on $\mathrm{lo}(v)$ in some run, then M was at v, the comparators that M passes after v are those of $A(v)$, and the leads of m after v depend only on v. We write $\tau(v)$ for the number of comparators that m enters after leaving v and that do not belong to $A(v)$. The *walk* from $\mathrm{lo}(v)$ starts at $\mathrm{lo}(v)$ and, while the current lead enters a blue comparator g, moves to $\mathrm{hi}(g)$. It stops at a lead $\rho(v)$ that is an output or enters a comparator that is not blue, and $d(v)$ denotes the number of blue comparators it

passes. For a pass-through q and a red comparator v such that $u(q)$ and $u(v)$ are incomparable in B, let $e(q, v) = \mathrm{lca}_B(u(q), u(v))$.

Lemma 3.4 (after a meeting)**.** *Let v be a red comparator, and suppose that m leaves v on $\mathrm{lo}(v)$ in some run. In that run m follows the walk from $\mathrm{lo}(v)$ and passes its $d(v)$ blue comparators, and one of the following holds.*

(i) $\rho(v)$ is an output. Then $\rho(v) = o_m$ and $\tau(v) = d(v)$.
(ii) $\rho(v)$ is the blue input lead of a pass-through $q \in A(v)$. Then m meets M at q, leaves on $\mathrm{lo}(q)$, and $\tau(v) = d(v) + \tau(q)$.
(iii) $\rho(v)$ is the blue input lead of a pass-through $q \notin A(v)$. Then $u(q)$ and $u(v)$ are incomparable in B, the vertex $e = e(q, v)$ lies in $A(v)$, the value m passes the $\mathrm{cl}(q, e)$ red comparators from q up to e, none of which lies in $A(v)$, meets M at e and leaves on $\mathrm{lo}(e)$, and $\tau(v) = d(v) + \mathrm{cl}(q, e) + \tau(e)$.

Proof. The leads of the walk are blue. The other input lead of a blue comparator is blue, so it does not carry M, and m leaves on hi, which is blue by Lemma 3.1(a). Therefore m follows the walk.

If $\rho(v)$ is an output, m ends on it, so $\rho(v) = o_m$ because T is top-two correct, and $\tau(v) = d(v)$. Otherwise $\rho(v)$ enters a comparator q that is not blue and has the blue input lead $\rho(v)$, so q is a pass-through. Since q comes after v in the circuit, M is at q exactly when $q \in A(v)$.

In case (ii), M is on the red input lead of q, so m leaves on $\mathrm{lo}(q)$, and q itself is not counted in $\tau(v)$. This gives $\tau(v) = d(v) + \tau(q)$.

In case (iii), m leaves q on the red lead $\mathrm{hi}(q)$. If $u(q)$ were an ancestor of $u(v)$, or equal to $u(v)$ with q above v, then q would lie in $A(v)$. If $u(q)$ were a proper descendant of $u(v)$, or equal to $u(v)$ with q below v, then q would come before v in the circuit. Hence $u(q)$ and $u(v)$ are incomparable, and e is a proper ancestor of $u(v)$, hence $e \in A(v)$. At each pass-through above q on $\mathrm{seg}(u(q))$, the value m is on the red input lead and the other input lead is blue, so m leaves on hi. A branch node z strictly between $u(q)$ and e is not an ancestor of $u(v)$, so $z \notin A(v)$. It comes after v in the circuit, so it is not on the path of M, and m leaves it on hi and continues up $\mathrm{seg}(z)$, whose pass-throughs are not in $A(v)$ either. At e the value M arrives on the input lead on the side of $u(v)$ and m on the other one, so m leaves on $\mathrm{lo}(e)$. The comparators passed from q up to e, with e excluded, are the $\mathrm{cl}(q, e)$ red comparators counted above, and none of them is in $A(v)$, which gives the formula for $\tau(v)$. □

Each step of this recursion moves to a comparator of $A(v)$, so it ends, in case (i).

3.3. **Reduced networks and the forest of the second maximum.** The blue input lead of a pass-through is its *port*. A blue lead is *relevant* if it carries m in some run. We call T *reduced* if

(R1) the port of each pass-through is relevant, and
(R2) both input leads of each blue comparator are relevant.

Lemma 3.5 (reduction)**.** *For each top-two correct network T there is a reduced top-two correct network T^* on the same inputs such that each run meets at most as many comparators in T^* as in T.*

Proof. Suppose a pass-through q violates (R1). Delete q, and join its red input lead to $\mathrm{hi}(q)$ and its port to $\mathrm{lo}(q)$. The port carries neither M, being blue, nor m, and $\mathrm{lo}(q)$ carries neither: it would carry m only if m entered on the port. A value M or m on the red input lead leaves q on $\mathrm{hi}(q)$, because the port does not carry M. Thus in each run M and m follow the same leads as before, with q skipped. Suppose a blue comparator g has an input lead that is not relevant. Delete g, and join its other input lead to $\mathrm{hi}(g)$ and that input lead to $\mathrm{lo}(g)$. The value M never enters g, and m leaves g on $\mathrm{hi}(g)$ whenever it enters, so again M and m follow the same leads,

with g skipped. Each deletion keeps the network top-two correct, increases no $|W(x,y)|$, and does not change which leads carry m, up to the joins. Repeat while (R1) or (R2) fails. This ends because each step removes a comparator. □

Lemma 3.6 (the forest of the second maximum)**.** *Let T be reduced.*

(a) A relevant lead enters a blue comparator, or is the port of a pass-through, or is o_m.
(b) Join each relevant lead that enters a blue comparator g to $\mathrm{hi}(g)$*. Then the relevant leads form a forest in which each vertex has zero or two children. Its roots are o_m and the ports. Its leaves are the leads* $\mathrm{lo}(c)$ *of all branch nodes c and the leads* $\mathrm{lo}(q)$ *of those pass-throughs q for which* $\mathrm{lo}(q)$ *is relevant.*
(c) A blue lead that is not relevant enters no comparator.
(d) If $\mathrm{lo}(v)$ *is relevant for a red comparator v, then $\rho(v)$ is the root of the tree that contains* $\mathrm{lo}(v)$*, and $d(v)$ is the depth of* $\mathrm{lo}(v)$ *in that tree.*

Proof. (a) A relevant lead is blue, so it does not enter a branch node. If it enters a pass-through, it is the port. If it enters no comparator, it is an output that carries m in some run, so it is o_m.

(b) If a relevant lead enters a blue comparator g, then $\mathrm{hi}(g)$ carries m whenever that lead does, so $\mathrm{hi}(g)$ is relevant. Each lead enters at most one comparator and the circuit is acyclic, so the joins form a forest. Its roots are the relevant leads that enter no blue comparator, which by (a) and (R1) are o_m and all ports. The lead o_m is relevant because m ends there. The children of a vertex $\mathrm{hi}(g)$ are the two input leads of g, both relevant by (R2). A relevant lead that is not $\mathrm{hi}(g)$ for a blue comparator g is neither an input lead nor the hi output of a red comparator, since those are red. It is therefore $\mathrm{lo}(g)$ for a comparator g, which is not blue because m leaves blue comparators on hi. Each lead $\mathrm{lo}(c)$ of a branch node c is relevant, since m leaves c on $\mathrm{lo}(c)$ in the runs (x,y) with $\mathrm{lca}_B(x,y) = c$.

(c) A blue lead does not enter a branch node. By (R2) an irrelevant one enters no blue comparator, and by (R1) it is the port of no pass-through.

(d) The walk from $\mathrm{lo}(v)$ follows the joins of (b). □

3.4. **Configurations.**

Definition 3.7. A *configuration* Γ on N inputs consists of

(i) a binary tree B whose leaves are the N inputs, together with an edge from its root to o_M;
(ii) a number $k(u) \geq 0$ of pass-throughs $(u,1), \ldots, (u,k(u))$ on each edge $\mathrm{seg}(u)$, numbered from below; the *red nodes* are the internal vertices of B, called branch nodes, and the pass-throughs;
(iii) for each red node v, either the mark `dead`, allowed for pass-throughs only, or a *source* $(\rho(v), d(v))$ consisting of a *root* $\rho(v)$, which is o_m or a pass-through, and a *depth* $d(v) \geq 0$; the red nodes that are not `dead` are *live*;

subject to two conditions:

(K) for each $r \in \{o_m\} \cup \{\text{pass-throughs}\}$, the live red nodes with root r form a nonempty set whose depths satisfy $\sum 2^{-d} = 1$;
(G) the directed graph $G(\Gamma)$ on the red nodes, with an arc from each red node to the next red node above it and an arc from each live red node v to its root whenever that root is a pass-through, is acyclic.

A root that is a pass-through q stands for the port of q. We call the pair (B,k) of (i) and (ii) the *skeleton* of Γ. The quantities δ, pre, $u(\cdot)$, $A(\cdot)$, cl and $e(\cdot,\cdot)$ are defined from the skeleton as

in Sections 3.1 and 3.2. For each live red node v, define $\tau_\Gamma(v)$ by the recursion of Lemma 3.4,

$$\tau_\Gamma(v) = \begin{cases} d(v) & \text{if } \rho(v) = o_m, \\ d(v) + \tau_\Gamma(q) & \text{if } \rho(v) = q \in A(v), \\ d(v) + \mathrm{cl}(q, e(q,v)) + \tau_\Gamma(e(q,v)) & \text{if } \rho(v) = q \notin A(v), \end{cases}$$

and put $\tau_\Gamma(v) = \infty$ if the recursion reaches a dead node. The *cost* of the run (x, y) is $\mathrm{cost}_\Gamma(x, y) = \delta(x) + \mathrm{pre}(y, c) + \tau_\Gamma(c)$ with $c = \mathrm{lca}_B(x, y)$, and Γ is *feasible at level* L if $\mathrm{cost}_\Gamma(x, y) \le L$ for all $x \ne y$. For an internal vertex c of B with children c_1 and c_2 and an input x below c_1, put $\sigma(c, x) = L - \delta(x) - \max_{y \text{ below } c_2} \mathrm{pre}(y, c)$.

In the third case of the recursion, $u(q)$ and $u(v)$ are incomparable by (G). Otherwise q would lie at or below v on the path to o_M, as in the proof of Lemma 3.4, and the path from q to v in $G(\Gamma)$ together with the arc from v to q would form a cycle. Each step of the recursion moves strictly up, so τ_Γ is well defined.

Theorem 3.8 (reduction to configurations)**.** *Let $N \ge 3$ and $L \ge 0$.*

(a) If some top-two correct network on N inputs has $p_2 \le L$, then so does a reduced one, T^*, and the configuration of T^* given by Lemmas 3.1 and 3.6 is feasible at level L and gives each live red node a depth at most L.

(b) Conversely, each configuration Γ feasible at level L is realized by a top-two correct network in which the run (x, y) meets exactly $\mathrm{cost}_\Gamma(x, y)$ comparators, for all $x \ne y$.

Proof. (a) By Lemma 3.5 we may assume that the network T is reduced. Lemma 3.1 gives B and the numbers $k(u)$. By Lemma 3.6, give each branch node c, and each pass-through q for which $\mathrm{lo}(q)$ is relevant, the source formed by the root and the depth of its lo output in the forest, and mark the other pass-throughs dead. Condition (K) holds by Lemma 2.5(a); each root is the root of some live red node, since the tree of the forest at that root has a leaf. An arc of $G(\Gamma)$ is a directed path in the circuit: from a red comparator up to the next one, or from a live red node v through $\mathrm{lo}(v)$ and the blue comparators of its tree into the pass-through that is its root. Consequently, a cycle in $G(\Gamma)$ would give a cycle in the circuit, and (G) holds. By Lemmas 3.3, 3.4 and 3.6(d), $|W(x, y)| = \mathrm{cost}_\Gamma(x, y)$ for all $x \ne y$. No recursion reaches a dead node, because m leaves each node that the recursion reaches on its lo output in the run. Hence Γ is feasible at level L. For a live red node v, the value m is on $\mathrm{lo}(v)$ in some run (x, y), and that run passes the $d(v)$ blue comparators below $\rho(v)$ after $\mathrm{lo}(v)$, so $d(v) \le |W(x, y)| \le L$.

(b) Build the circuit as follows. The red part is the tree B with $k(u)$ comparators on each edge $\mathrm{seg}(u)$, each branch node taking the leads from its two children and each pass-through taking the lead from below on one input lead. Its top lead is o_M. For each root r, take a binary tree of blue comparators whose leaves are the lo outputs of the live red nodes with root r, at the prescribed depths (Lemma 2.5(b)). Each internal vertex of this tree is a comparator with the leads at its children as input leads. The lead at its root is the port of r, or the output o_m if $r = o_m$. The lo outputs of dead nodes and of blue comparators are outputs. A directed cycle in this circuit would give a cycle in $G(\Gamma)$, after each tree of blue comparators is contracted, so by (G) the circuit is acyclic. The maximum, from each input, stays in the red part and ends on o_M, so the red comparators of this network are those of Γ. The proofs of Lemmas 3.3 and 3.4 use top-two correctness only to identify the output in case (i), and here each walk from a red node ends at its root by construction. Since Γ is feasible, the recursion from each $\mathrm{lo}(c)$ avoids dead nodes and ends at o_m. Thus m always ends on o_m, the network is top-two correct, and $|W(x, y)| = \mathrm{cost}_\Gamma(x, y)$. □

3.5. **Isomorphic configurations.** An *isomorphism* from a binary tree B onto a binary tree B' is a bijection h between their vertices that preserves the parent relation; it maps the root to the root and inputs to inputs. If $B' = B$, it is an *automorphism* of B. The isomorphism classes of binary trees are the *tree shapes*. For a configuration Γ with skeleton (B, k), let $h\Gamma$ denote the data consisting of the skeleton $(B', k \circ h^{-1})$, in which the pass-through $(h(u), i)$ is the image of the pass-through (u, i), and of the assignment that gives the red node $h(v)$ the root $h(\rho(v))$ and the depth $d(v)$ if v is live, and the mark dead otherwise. Here $h(o_m) = o_m$. An automorphism h of B with $k \circ h^{-1} = k$ is an automorphism of the skeleton (B, k).

Lemma 3.9. *Let Γ be a configuration with skeleton (B, k), and let h be an isomorphism from B onto a binary tree B'. Then $h\Gamma$ is a configuration, h carries δ,* pre *and σ of Γ to those of $h\Gamma$, and $\mathrm{cost}_{h\Gamma}(h(x), h(y)) = \mathrm{cost}_\Gamma(x, y)$ for all inputs $x \neq y$. In particular, $h\Gamma$ is feasible at level L if and only if Γ is.*

Proof. The quantities δ, pre, σ, lca_B, $A(\cdot)$, cl and $e(\cdot, \cdot)$ are defined from the parent relation, the numbers k and the positions of the pass-throughs on their edges, and h carries each of them for Γ to the same quantity for $h\Gamma$. For instance, δ takes the same value at x for Γ as at $h(x)$ for $h\Gamma$, and $e(h(q), h(v)) = h(e(q, v))$. The red nodes with root $h(r)$ in $h\Gamma$ are the images of those with root r in Γ, with the same depths, so (K) holds for $h\Gamma$. The graph $G(h\Gamma)$ is the image of $G(\Gamma)$ under h, so (G) holds. Induction along the recursion of Definition 3.7 gives $\tau_{h\Gamma}(h(v)) = \tau_\Gamma(v)$ for every live red node v, and the costs follow. □

The *decorated canonical string* of the subtree at a vertex u is $k(u)$L if u is an input, and $k(u)(s,t)$ otherwise, where s and t are the decorated canonical strings of the two children of u, with s preceding or equal to t lexicographically. Two subtrees have the same decorated canonical string if and only if some bijection between them preserves the parent relation and the numbers k.

3.6. **Bounds for the search.**

Lemma 3.10 (fixed-input bound)**.** *Let T be top-two correct on $N \geq 3$ inputs, and let x be an input. Then $\max_{y \neq x} |W(x, y)| \geq \delta(x) + \lceil \log_2(N-1) \rceil$. In particular, if $p_2(T) \leq L$, then $\delta(x) \leq H_L$ for each input x, where $H_L = L - \lceil \log_2(N-1) \rceil$, and B has height at most H_L.*

Proof. We delete the comparators on the path from x to o_M, join the other input lead of each to its lo output, and call the result T_x. It is the network T/x of Section 2.2, on the $N-1$ inputs other than x. In the run (x, y) of T, the value m is never on a lead of Π_x. At a comparator on the path from x it leaves on lo, where the join takes it, and at every other comparator it leaves on hi, as the maximum of T_x does. Hence the leads of m in the run (x, y) are the leads of the path of the maximum of T_x from y, up to the joins, and this path ends on o_m for every $y \neq x$. By Lemma 3.1(c) applied to T_x, the tree of T_x has $N-1$ leaves, so by Lemma 2.5(a) some leaf y has depth at least $\lceil \log_2(N-1) \rceil$ in it. The path of y in T_x then passes at least $\lceil \log_2(N-1) \rceil$ comparators of T_x. In the run (x, y) the value m enters them, and they are not on the path from x. Therefore $|W(x, y)| \geq \delta(x) + \lceil \log_2(N-1) \rceil$. The depth of x in B is at most $\delta(x)$. □

Corollary 3.11. *$P(2, N) \geq \nu(N) := \lceil \log_2 N \rceil + \lceil \log_2(N-1) \rceil$ for $N \geq 3$.*

Proof. Some leaf x of B has depth at least $\lceil \log_2 N \rceil$ by Lemma 2.5(a), so $\delta(x) \geq \lceil \log_2 N \rceil$, and Lemma 3.10 applies. □

Corollary 3.11 amounts to applying the one-value bound twice. For $3 \leq N \leq 32$ it equals $\lceil \log_2 F(N) \rceil$ when $N \in \{3, 4, 6, 10, 11, 18, 19, 20\}$, and it is smaller by one or two otherwise (Table 1).

Lemma 3.12 (budgets)**.** *Let Γ be a configuration feasible at level L, let c be an internal vertex of B with children c_1 and c_2, and let x lie below c_1. Then*

$$d(c) + \chi(c) \;\le\; \tau_\Gamma(c) \;\le\; \sigma(c, x),$$

where $\chi(c) = \mathrm{cl}(\rho(c), e(\rho(c), c))$ if $\rho(c)$ is a pass-through outside $A(c)$, and $\chi(c) = 0$ otherwise.

Proof. The upper bound is $\mathrm{cost}_\Gamma(x, y) \le L$ for a y below c_2 that maximizes $\mathrm{pre}(y, c)$. The lower bound follows from the recursion, since $\tau_\Gamma \ge 0$. □

4. Determining $P(2, N)$

We now describe the search for a configuration feasible at level L, for given $N \ge 3$ and L, prove that each of its tests is valid, and report the searches and their independent checks. Put $H_L = L - \lceil \log_2(N-1) \rceil$.

4.1. **The basic search.**

(S1) Enumerate the binary trees B with N leaves and height at most H_L, up to isomorphism. A tree is encoded by a canonical string: `L` for a leaf, and `(`st`)` for a tree with subtrees encoded by s and t, where s precedes or equals t lexicographically.

(S2) For each B, enumerate the numbers $k(u) \in \{0, \dots, H_L\}$ with $\delta(x) \le H_L$ for all inputs x, and keep one skeleton (B, k) from each class under the automorphisms of B, using the decorated canonical string of Section 3.5.

(S3) Compute $\sigma(c, x)$ for each internal vertex c and each input x below c, and discard the skeleton if some $\sigma(c, x)$ is negative.

(S4) Search depth first over the red nodes: the internal vertices of B in increasing order of $\min_x \sigma(c, x)$, then the pass-throughs. An internal vertex receives a root and a depth in $\{0, \dots, L\}$, and a pass-through receives the mark dead or a root and a depth in $\{0, \dots, L\}$. A partial assignment is extended only if
   (a) at each root, the depths assigned so far satisfy $\sum 2^{-d} \le 1$, a depth 0 occurs only as the only member, and a root whose sum has reached 1 receives no further member;
   (b) an arc from a red node v to a pass-through q is added only if the graph formed by the arcs from each red node to the next red node above it and the arcs to roots present so far has no path from q to v;
   (c) each internal vertex c satisfies $d(c) + \chi(c) \le \sigma(c, x)$ for each input x below c.

(S5) At a complete assignment, check (K) and evaluate $\mathrm{cost}_\Gamma(x, y)$ for all $x \ne y$. The configuration is feasible if no recursion reaches a dead node and all costs are at most L.

(S6) Build the network of each feasible configuration as in Theorem 3.8(b), simulate all $N(N-1)$ runs, and compare the largest simulated cost with the largest value of cost_Γ.

Proposition 4.1. *If the search* (S1)–(S5) *finds no feasible configuration for (N, L), then $P(2, N) > L$.*

Proof. Suppose a top-two correct network on N inputs has $p_2 \le L$. Theorem 3.8(a) gives a reduced network T^* with $p_2(T^*) \le L$ whose configuration Γ is feasible at level L and has depths at most L. Lemma 3.10, applied to T^*, shows that the tree of Γ has height at most H_L and that $\delta(x) \le H_L$ for each x; in particular $k(u) \le \delta(x) \le H_L$ for each input x below u. Let B' be the tree listed by (S1) with the shape of the tree of Γ, and let h_0 be an isomorphism onto it. By Lemma 3.9, h_0 carries δ, so (S2) enumerates the skeleton of $h_0\Gamma$ and keeps the skeleton of $h\Gamma$, where $h = \alpha h_0$ for some automorphism α of B'. By the same lemma, $h\Gamma$ is feasible at level L, has depths at most L and satisfies $\delta(x) \le H_L$. Replace Γ by $h\Gamma$. Each $\sigma(c, x)$ is at least $\tau_\Gamma(c) \ge 0$ by Lemma 3.12, so (S3) keeps the skeleton. Condition (K) implies (a) for each partial

assignment, (G) implies (b), and Lemma 3.12 gives (c). So the search reaches the assignment of Γ, and (S5) reports it feasible. □

The programs decide whether a pass-through q lies in $A(v)$, for a red node v, by testing whether $u(q)$ is equal to or an ancestor of the input x at hand, for x below v. For assignments that pass (b) this gives the right answer for all such x, because a pass-through at or below v on the side of x would close a cycle with the arc from v to q.

4.2. **Three further tests.** We add the following three tests to (S1)–(S5). Each discards only partial data that admit no configuration feasible at level L, so Proposition 4.1 holds with them.

(P1) *Path test.* After (S3), discard the skeleton if, for some input x such that each pass-through lies on the path from x to o_M, the internal vertices c above x satisfy $\sum_c 2^{-\sigma(c,x)} > 1$.

(P2) *Capacity test.* During (S4), let U be the set of internal vertices not yet assigned. Call a root open if its sum is below 1 and it has no member of depth 0. For $c \in U$ and an open root r that c may join under (b), let $\mu(c,r)$ be $\min_x \sigma(c,x)$ minus the value that $\chi(c)$ would take if $\rho(c) = r$, capped at L. The programs also evaluate a value of μ for roots that (b) forbids, and including such roots in the conditions below can only weaken the test. Prune if some $c \in U$ has $\mu(c,r) < 0$ for each open root r, or if

$$\sum_{c \in U} 2^{-\max_r \mu(c,r)} > \sum_{r \text{ open}} \Big(1 - \sum_{v \text{ assigned to } r} 2^{-d(v)}\Big).$$

(P3) *Partial costs.* During (S4), define $\tau^-(v)$ by the recursion of Definition 3.7 with $\tau^-(v) = 0$ for red nodes that are not yet assigned or are dead. Prune if $\delta(x) + \mathrm{pre}(y,c) + \tau^-(c) > L$ for some $x \neq y$ with $c = \mathrm{lca}_B(x,y)$.

Lemma 4.2. *Let Γ be feasible at level L, and let x be an input such that each pass-through of Γ lies on the path from x to o_M. Then $\sum_c 2^{-\tau_\Gamma(c)} \le 1$ over the internal vertices c above x, so the path test* (P1) *is valid.*

Proof. Let T be the network of Γ given by Theorem 3.8(b), let T_x be as in the proof of Lemma 3.10, and let R_x be the tree of maximum paths of T_x given by Lemma 3.1(b), with root o_m. For an internal vertex c above x, the lead $\mathrm{lo}(c)$ lies in R_x, since it carries the maximum of T_x when that maximum starts on an input y with $\mathrm{lca}_B(x,y) = c$. The comparators of T_x on the path in R_x from $\mathrm{lo}(c)$ to o_m are the comparators that m enters after leaving c and that are not on the path from x, so there are $\tau_\Gamma(c)$ of them.

We claim that no lead $\mathrm{lo}(c)$ lies on the path in R_x from another lead $\mathrm{lo}(c')$ to o_m. Suppose it did. Then in some run m leaves c', later enters c, and leaves c on $\mathrm{lo}(c)$, so c lies above c' and m arrives at c on the input lead that is not on the path from x. That lead is red. By Lemma 3.4, m is on a red lead only while it climbs from a pass-through q to the vertex $e(q,v)$, where v is the comparator of the previous meeting. Here $e(q,v) = c$, and v lies on the path from x below c, so $u(q)$ lies below the child of c that is not on the path from x. Then q is not on the path from x, contrary to the hypothesis.

So the leads $\mathrm{lo}(c)$ form an antichain in R_x. Give the root of R_x weight 1, and let each vertex pass its weight to its child or split it equally between its two children. The leaves below a vertex then carry total weight equal to its weight, so the weights of an antichain sum to at most 1. The weight of $\mathrm{lo}(c)$ is 2^{-j}, where j is the number of vertices with two children strictly above $\mathrm{lo}(c)$. Such a vertex is $\mathrm{hi}(g)$ for a comparator g of T_x on the path from $\mathrm{lo}(c)$, so $j \le \tau_\Gamma(c)$. Hence $\sum_c 2^{-\tau_\Gamma(c)} \le 1$. Since $\tau_\Gamma(c) \le \sigma(c,x)$ by Lemma 3.12, a configuration with $\sum_c 2^{-\sigma(c,x)} > 1$ is not feasible. □

The hypothesis of Lemma 4.2 cannot be dropped, however. Take the sorting network $(1,0),(2,0),(2,1)$ on 3 inputs, written with the conventions of Section 2.1; it is the network of [15, Proposition 3]. Its branch nodes are $c_1 = (1,0)$ and $c_2 = (2,1)$, and $(2,0)$ is a pass-through on the edge of input 2. For $x = 0$ the value m leaves c_1, enters $(2,0)$ through its port, climbs to c_2 and meets M again there. So $\tau(c_1) = 1$ and $\tau(c_2) = 0$, and $\sum_c 2^{-\tau(c)} = 3/2$. At $L = 3$ the path test would discard this skeleton, although the network has $p_2 = 3$. A missing antichain step of this kind is what [15] found in Van Voorhis's argument.

Lemma 4.3. *The tests* (P2) *and* (P3) *are valid.*

Proof. (P2) In a feasible completion each $c \in U$ is assigned to a root that is open at this node, since by (K) a root whose sum has reached 1 takes no further member, and by Lemma 3.12 its depth is at most $\mu(c,r)$ for its root r. It therefore adds at least $2^{-\max_r \mu(c,r)}$ to the sum at its root. By (K) the total added at the open roots equals their remaining capacity, and the pass-throughs add nonnegative amounts.

(P3) In a completion the terms fixed by the assigned red nodes do not change, and the other terms are nonnegative or infinite. Induction along the recursion gives $\tau^-(v) \le \tau_\Gamma(v)$ for every live red node v. □

4.3. **A symmetry reduction.** Let v be an internal vertex of B whose children u_1 and u_2 have the same decorated canonical string (Section 3.5). Then some bijection from the subtree at u_1 onto the subtree at u_2 preserves the parent relation and the numbers k. Let g_v exchange the two subtrees by such a bijection and fix every other vertex. Then g_v is an automorphism of the skeleton (B,k), and $g_v^{-1} = g_v$. In (S4) the red nodes are visited in a fixed order $s_1,\dots,s_n$. We order the possible values of a red node as follows: the mark dead comes first, then the root o_m, then the pass-throughs in a fixed order, and values with the same root are ordered by depth. Assignments are compared lexicographically through $(a(s_1),\dots,a(s_n))$. For an automorphism g of the skeleton and a partial assignment a, ga is defined as for configurations in Section 3.5, and a *class* of assignments is an orbit of the group generated by the automorphisms g_v.

Symmetry test. Let a be a partial assignment of $s_1,\dots,s_t$, and let $g = g_v$ for some v as above. Since $g^{-1} = g$, the value $(ga)(s_j)$ is the value $a(g(s_j))$ with its root mapped by g, and it is defined when $g(s_j)$ is among $s_1,\dots,s_t$. Prune a if for some such g and some $j \le t$ the values $(ga)(s_i)$ with $i \le j$ are defined, $(ga)(s_i) = a(s_i)$ for $i < j$, and $(ga)(s_j) < a(s_j)$. Then every completion b of a satisfies $gb < b$. The test may keep several members of a class of assignments, but by the next lemma it keeps at least one.

Lemma 4.4. *If some configuration with skeleton (B,k) is feasible at level L and has depths at most L, then the search* (S4)–(S5) *with the tests* (P2), (P3) *and the symmetry test finds one.*

Proof. Let Γ be such a configuration. Among the configurations $h\Gamma$, where h ranges over the group generated by the automorphisms g_v, let Γ^* be the lexicographically least. It has the same depths as Γ, and it is feasible by Lemma 3.9. Moreover $\Gamma^* \le g_v\Gamma^*$ for every v, so the symmetry test prunes no partial assignment of Γ^*. Neither do the tests of (S4), as in the proof of Proposition 4.1, nor (P2) and (P3), by Lemma 4.3. So the search reaches Γ^*, and (S5) reports it feasible. □

4.4. **Programs.** We call (S1)–(S6) the *basic search*. The *refined search* adds (P1)–(P3) and can split the tree shapes among s processes by their position in the list of (S1) modulo s. The supplement [18] gives the program for each.

The search for $N = 28$ needs three further features; the *extended search* adds them to the refined search. First, to keep one skeleton from each class in (S2) without a table, it keeps the skeleton in which, at each vertex whose two subtrees have the same shape, the decorated

canonical string of the left subtree precedes or equals that of the right one; left and right refer to the order of the two subtrees in the canonical string of B. Each class of skeletons under the automorphisms of B has exactly one such member: swapping subtrees bottom-up brings any skeleton into this form, and induction on the height shows that two skeletons in this form with the same decorated canonical string coincide. Since (S3), (P1) and the bound $\delta(x) \le H_L$ are invariant under the automorphisms of B (Lemma 3.9), this keeps one skeleton from each class that the other programs examine. Second, it applies this test and (S3) while the numbers $k(u)$ are enumerated, with the inputs first, in the order of the canonical string, and then the internal vertices in postorder, so that each vertex comes after its descendants. Once the subtree of an internal vertex c is complete, the values $\mathrm{pre}(y, c)$ are final and $\delta(x)$ can only grow, so a negative $\sigma(c, x)$ stays negative; and the comparison of two sibling subtrees is final once both are complete. Third, the search for one tree shape can be split among several processes. Below a fixed level of the enumeration of the numbers $k(u)$, or a fixed depth of the depth-first search (the number of red nodes assigned), the subtrees are dealt out to the processes in turn, so that their shares partition them. All processes walk the levels above in the same order with the same tests, so the shares together cover the search, and a share can be split again at a deeper level. A complete assignment lies at depth equal to the number of red nodes, which at $N = 28$ is at least 27 and exceeds every split depth we used, so it is reached in exactly one share. The extended search *with the symmetry test* also applies the test of Section 4.3, and it can split the depth-first search at more levels. The refined search and the extended search, with or without the symmetry test, are the *searches with (P1)–(P3)*.

The programs compute Kraft sums in double precision. Their terms are powers of two with exponents between $-L$ and 0, so the sums are exact.

4.5. **Results.** We ran the search at $L = \lceil \log_2 F(N) \rceil - 1$ for $3 \le N \le 13$ and for $N = 17$, 21 and 28, the values of N between 14 and 32 at which $\lceil \log_2 F(N) \rceil$ increases.

Theorem 4.5. *Let $N \in \{3, 4, \ldots, 13\} \cup \{17, 21, 28\}$ and $L = \lceil \log_2 F(N) \rceil - 1$. The search of Sections 4.1 to 4.4 finds no configuration on N inputs that is feasible at level L. Hence $P(2, N) \ge \lceil \log_2 F(N) \rceil$.*[†]

Proof. The searches report no feasible configuration (Table 2). Proposition 4.1 applies; for the runs of the searches with (P1)–(P3), which examine one skeleton from each class (Section 4.4), it is combined with Lemmas 4.2 and 4.3, and for the runs with the symmetry test also with Lemma 4.4. For a run split into shares, the shares together cover the search (Section 4.4), and at $N = 28$ a script in the supplement [18] checks that every one of the 62,641 shapes is covered and that the shares of each divided search are all present. □

For $N \in \{3, 4, 6, 10, 11\}$ no tree has height at most H_L, as Corollary 3.11 already shows. At $N = 13$ and $L = 8$, four tree shapes have height at most $H_L = 4$, and 25 skeletons pass (S3). The depth-first search abandons each of them before it completes an assignment, after 456 nodes in total.

At $N = 28$ the shapes were searched with (P1)–(P3), in many independent processes. The searches for the 119 shapes that take far longer than the rest were divided into shares (Section 4.4), and 17 of these shapes were searched with the symmetry test; the extended search without the test searched all the others. No part reports a feasible configuration. The script of the proof of Theorem 4.5 also recomputes the totals in Table 2, in which each skeleton is counted once, the nodes above a split of the depth-first search once per share, and for the 17 shapes searched with the symmetry test the counts after that test. Each search with $N \le 21$ took under five seconds; the run at $N = 28$ took of the order of 10^4 core-hours.

N	L	H_L	search	shapes	skeletons	search nodes	complete
3	2	1	basic	0	0	0	0
4	3	1	basic	0	0	0	0
5	5	3	basic	2	11	58	3
6	5	2	basic	0	0	0	0
7	6	3	basic	1	2	24	0
8	6	3	basic	1	1	6	0
9	7	4	basic	8	309	1,273	2
10	7	3	basic	0	0	0	0
11	7	3	basic	0	0	0	0
12	8	4	basic	7	67	130,268	57,650
13	8	4	basic	4	25	456	0
17	9	5	basic	187	262,776	567,225	0
21	10	5	refined	177	62,879	453,320	0
28	11	6	see text	62,641	262,417,641,559	30,972,138,194,473	12,416,681,168,846

TABLE 2. The searches at $L = \lceil \log_2 F(N) \rceil - 1$. None finds a feasible configuration. "Skeletons" counts the skeletons that pass (S3), and for the searches with (P1)–(P3) also (P1). "Search nodes" counts the nodes of the depth-first search, and "complete" the complete assignments it reaches.

Proof of Theorem 1.2. For $N \in \{3, \dots, 13\} \cup \{17, 21, 28\}$ the lower bound is Theorem 4.5. For each other $N \leq 32$, let N' be the largest element of $\{13, 17, 21, 28\}$ below N. By Table 1 the values $\lceil \log_2 F(N) \rceil$ and $\lceil \log_2 F(N') \rceil$ are equal, and Lemma 2.4 gives $P(2, N) \geq P(2, N')$. The upper bound is Proposition 2.6. □

4.6. **Independent checks.** We checked the computation in four ways. First, programs that share no code with the basic search and do not use the model of Section 3 determined $P(2, N)$ for $N \leq 13$ again: method A searches comparator networks directly, with a pruning rule based on a Kraft inequality over the runs, and methods B and C search a relaxed tree model with its own structure theorem; notes in the supplement [18] prove the Kraft inequality and the structure theorem. Second, an integer program (HiGHS [10]) for each skeleton and millions of networks mapped into the model test the basic search and its model. Third, a search over standard networks that does not use the model settles $N = 12$ and $N = 13$ at $L = 8$; its pruning rules were tested, but unlike those of methods A, B and C they have no written proofs. Fourth, the searches with (P1)–(P3) agree with each other and with the basic search where their runs overlap, split runs agree with unsplit ones, and a separate run at $N = 28$ by other variants of the search found no feasible configuration in all shapes except the 119 slowest; Section 6 and Table 3 of [20] list these checks with their scope and outcome. For the 119 slowest shapes, and in particular for the 17 searched with the symmetry test, the result therefore rests on a single computation, made by programs that were checked as described above.

5. Partial sorting networks and cores

From now on we apply the two-value argument to the suffix of a network. This section introduces the sets of 0-1 vectors that prefixes of a network produce, the networks that sort them, and the cores of pairs of channels.

5.1. **Partial sorting networks.** For a set of channels A, a vector is an element $v \in \{0, 1\}^A$, e_c denotes the vector with a single 1 on channel c, and $\mathbf{1}$ the vector of all 1s. A network T on A *sorts* a set $X \subseteq \{0, 1\}^A$ if its outputs can be listed as $o_0, \dots, o_{|A|-1}$ so that, for each input

vector in X, the output values read in this order are nondecreasing. Networks that sort a set are the partial sorting networks of [8, Definition 1], except that partial sorting networks may also contain exchanges of channels, which the circuit model absorbs into the wiring. Let $s(X)$ be the least size of a network that sorts X. A permutation π of the channels acts on vectors by moving coordinates, and $\bar{X} = \{\mathbf{1} - v : v \in X\}$ is the complement of X. Sets related by a permutation of the channels, possibly combined with complementation, are *similar*.

Lemma 5.1. *(a)* $s(\{0,1\}^n) = S(n)$.
(b) If $X \subseteq X'$ *then* $s(X) \le s(X')$.
(c) Similar sets have the same value of s.

Proof. (a) A network sorts every input sequence if and only if it sorts all 0-1 sequences [12, Section 5.3.4], and a network in the circuit model that sorts can be untangled into a standard sorting network of the same size [12, Exercise 5.3.4–16]. (b) and (c) are Lemma 2 and Corollary 6 of [8]. □

A standard comparator (i, j) maps a set X to the set $X(i, j)$ of its images. It is *useful* for X if X contains a vector v with $(v_i, v_j) = (0, 1)$ and a vector with $(v_i, v_j) = (1, 0)$. If no vector of X has $(v_i, v_j) = (0, 1)$, the comparator leaves X unchanged. If some vector does but none has $(1, 0)$, it acts on X as the exchange of channels i and j.

5.2. **Pruning and cores.**

Definition 5.2. Let $X \subseteq \{0,1\}^A$ and $\epsilon \in \{0,1\}$, the *polarity*. A channel $c \in A$ is *extremal* in X for polarity 0 if $e_c \in X$, and for polarity 1 if $\mathbf{1} - e_c \in X$. For polarity 0 and an extremal channel c, let R be the set of vectors $u \in X$ that can be reached from e_c by a sequence of vectors of X in which each vector arises from the previous one by changing one coordinate from 0 to 1. The *pruned set* $X/c \subseteq \{0,1\}^{A\setminus\{c\}}$ consists of the restrictions of the vectors of R to $A \setminus \{c\}$. For polarity 1, X/c is the complement of $\bar{X}/c$, the latter taken for polarity 0.

The vectors of R have a 1 on channel c, so X/c is contained in Harder's pruned set $\{u|_{A\setminus\{c\}} : u \in X,\ u_c = 1\}$ [8, Definition 18].

Lemma 5.3 (pruning a channel)**.** *Let* T *sort* $X \subseteq \{0,1\}^A$ *with outputs* $o_0, \ldots, o_{|A|-1}$*, and let* c *be extremal in* X *for polarity* 0*. Then* Π_c *ends on* $o_{|A|-1}$*, and* T/c *sorts* X/c *with its outputs listed as* $o_0, \ldots, o_{|A|-2}$.

Proof. On input e_c the single 1 follows Π_c, and since the output is sorted, Π_c ends on $o_{|A|-1}$. By [8, Lemma 20], T/c sorts $\{u|_{A\setminus\{c\}} : u \in X,\ u_c = 1\}$, which contains X/c. □

A *maximal chain* of $\{0,1\}^A$ is a sequence $u_0, \ldots, u_{|A|}$ that starts at the zero vector and in which each vector arises from the previous one by changing one coordinate from 0 to 1. Harder calls a set *well-behaved* if it is a union of maximal chains [8, Definitions 27 and 28]; for such sets, sortnetopt [9] computes $s(X)$ [8, Corollary 56].

Lemma 5.4 (well-behaved sets)**.** *Let* $X \subseteq \{0,1\}^A$ *be well-behaved and let* c *be extremal in* X *for polarity* 0*. Then* X/c *is the union of the maximal chains of* $\{0,1\}^{A\setminus\{c\}}$ *that are contained in Harder's pruned set. In particular,* X/c *is well-behaved. For polarity* 1*, too,* X/c *is well-behaved. Consequently, the well-behaved sets are closed under comparators, similarity and pruning an extremal channel of either polarity, and every set obtained from* $\{0,1\}^n$ *by these operations is well-behaved.*

Proof. Let $u \in X/c$ be the restriction of $\bar{u} \in R$. Some maximal chain of $\{0,1\}^A$ inside X passes through $\bar{u}$. Its part above $\bar{u}$, preceded by a sequence from e_c to $\bar{u}$ as in Definition 5.2, runs from

e_c to $\mathbf{1}$ inside X, and its restriction to $A \setminus \{c\}$ is a maximal chain through u inside Harder's pruned set. Conversely, setting coordinate c to 1 in the vectors of a maximal chain of $\{0,1\}^{A\setminus\{c\}}$ inside Harder's pruned set gives a sequence in X from e_c as in Definition 5.2, so the chain lies in X/c. Polarity 1 follows by complementation. The set $\{0,1\}^n$ is well-behaved, and comparators, permutations of the channels and complementation map well-behaved sets to well-behaved sets [8, Lemma 35 and Corollary 36]. □

Definition 5.5. Let $X \subseteq \{0,1\}^n$ and let ϵ be a polarity. A *scenario* of (X,ϵ) is a pair (x,y) of channels such that x is extremal in X and y is extremal in X/x, both for polarity ϵ. Its *core* is the set $Y_{xy} = (X/x)/y \subseteq \{0,1\}^{[n]\setminus\{x,y\}}$, where $[n] = \{0,\dots,n-1\}$.

Remark 5.6. A channel is extremal in X for polarity 1 exactly when it is extremal in $\bar{X}$ for polarity 0, and X/c for polarity 1 is the complement of $\bar{X}/c$ for polarity 0. Hence the scenarios of $(X,1)$ are those of $(\bar{X},0)$, and their cores are the complements of the cores of $(\bar{X},0)$. By Lemma 5.1(c) they have the same values of s.

Lemma 5.7 (two values)**.** *Let T sort $X \subseteq \{0,1\}^n$ with outputs $o_0,\dots,o_{n-1}$, and let (x,y) be a scenario of $(X,0)$. In the run (x,y) of T, the value M ends on o_{n-1} and m ends on o_{n-2}, and $|T| - |W(x,y)| \geq s(Y_{xy})$.*

Proof. By Lemma 5.3, Π_x ends on o_{n-1} and T/x sorts X/x with outputs $o_0,\dots,o_{n-2}$. Since y is extremal in X/x, the same lemma applied to T/x shows that the path of the maximum of T/x from y ends on o_{n-2} and that $(T/x)/y$ sorts Y_{xy}. By Lemma 2.1 this path is the path of m, and $(T/x)/y$ has $|T| - |W(x,y)|$ comparators. □

5.3. **The trivial bound.** The bound of the next lemma uses only the comparators that produce X.

Lemma 5.8 (trivial bound)**.** *Let P be a sequence of k comparators on n channels, and let X be the set of outputs of P on $\{0,1\}^n$. For channels $i \neq j$, let x and y be the channels on which M and m end in the run (i,j) of P, and let $w = |W(i,j)|$. Then (x,y) is a scenario of $(X,0)$, and*

$$s(Y_{xy}) \;\geq\; S(n-2) - (k-w).$$

The same holds for polarity 1 with the run that has the smallest value on i and the second smallest on j.

Proof. Let $P_{ij} = (P/i)/j$. By Lemma 2.1 it has $k-w$ comparators. Its input channels are $[n]\setminus\{i,j\}$, and its outputs are the channels other than x and y. On an input vector with a 1 on channel i, the leads of Π_i carry 1, because comparator networks are monotone. Hence P and P/i agree on the remaining leads [8, proof of Lemma 16]. Applying the same argument to P/i and channel j shows, for every $z \in \{0,1\}^{[n]\setminus\{i,j\}}$, that $P(e_i+e_j+z)$ has 1s on x and y and agrees with $P_{ij}(z)$ on the other channels.

Since the map P is monotone and preserves the number of 1s, a vector u' that arises from u by changing one coordinate from 0 to 1 is mapped to a vector that arises from $P(u)$ in the same way. Starting from e_i, adding the 1 on j and then the 1s of z one at a time gives vectors whose images form a sequence in X from $P(e_i) = e_x$ through $P(e_i+e_j) = e_x+e_y$ to $P(e_i+e_j+z)$, each arising from the previous one by changing one coordinate from 0 to 1. Hence x is extremal in X, y is extremal in X/x, and $P_{ij}(z) \in Y_{xy}$ for all z. If a network U sorts Y_{xy}, then P_{ij} followed by U sorts $\{0,1\}^{[n]\setminus\{i,j\}}$, so $s(Y_{xy}) + k - w \geq S(n-2)$ by Lemma 5.1(a). For polarity 1, apply this to the sequence $\bar{P}$ obtained from P by replacing each comparator (a,b) with (b,a). Since it maps $\mathbf{1}-u$ to $\mathbf{1}-P(u)$, its set of outputs is $\bar{X}$, and its run with the largest values on i and j is the run of P with the smallest values there. By Remark 5.6, the scenarios and cores of $(\bar{X},0)$ give those of $(X,1)$. □

6. The weighted two-value bound

Definition 6.1. A *weighted instance* I on a set of n channels consists of a budget $L \geq 0$ and a finite list of *scenarios* (x, y, v), where $x \neq y$ are channels and $v \geq 0$ is an integer, the *weight*. Let E be the set of pairs (x, y) that occur in scenarios, and for $(x, y) \in E$ let $V(x, y)$ be the largest weight of a scenario on (x, y) and $b(x, y) = L - V(x, y)$. The instance is *feasible* if there are a comparator network T on the n channels and two distinct outputs o_M and o_m of T such that, for each $(x, y) \in E$, the run (x, y) of T ends with M on o_M and m on o_m, and $|W(x, y)| \leq b(x, y)$.

For a set $X \subseteq \{0,1\}^n$, a polarity ϵ, integers v_t indexed by the scenarios (x_t, y_t) of (X, ϵ), and a budget L, let $I(X, \epsilon; v; L)$ denote the weighted instance with budget L and scenarios (x_t, y_t, v_t).

Proposition 6.2 (weighted two-value bound)**.** *Let $X \subseteq \{0,1\}^n$, let ϵ be a polarity, and let v_t be integers with $0 \leq v_t \leq s(Y_t)$ for each scenario (x_t, y_t) of (X, ϵ) with core Y_t. If $s(X) \leq L$, then $I(X, \epsilon; v; L)$ is feasible. Hence, if $I(X, \epsilon; v; L)$ is infeasible, then $s(X) \geq L + 1$.*

Proof. For polarity 0, let T sort X with $s(X)$ comparators and outputs $o_0, \ldots, o_{n-1}$. By Lemma 5.7, each run (x_t, y_t) ends with M on o_{n-1} and m on o_{n-2}, and $|W(x_t, y_t)| \leq s(X) - s(Y_t) \leq L - v_t$. With $o_M = o_{n-1}$ and $o_m = o_{n-2}$, the network T shows that the instance is feasible. For polarity 1, Remark 5.6 shows that $I(X, 1; v; L) = I(\bar{X}, 0; v; L)$ and that the cores of $(\bar{X}, 0)$ have the same values of s, and $s(\bar{X}) = s(X)$. □

For $X = \{0,1\}^n$ and polarity 0, every pair of distinct channels is a scenario, and every core is the full set on the other $n-2$ channels. With all weights equal to $S(n-2)$, the instance with budget L is feasible exactly when some top-two correct network has $p_2 \leq L - S(n-2)$ (Definition 2.2). The weighted bound thus generalizes the two-value bound $S(n) \geq S(n-2) + P(2, n)$ of Lemma 2.3. It is also the two-value counterpart of Harder's Huffman bound [8, Theorem 26], which gives each extremal channel i of X its own cost $s(X/i)$. As in Harder's use of that bound, X is the set produced by a prefix of a network, but our weights are lower bounds on the costs of the cores (Section 9) and different scenarios have different budgets.

7. From feasible networks to configurations

Sections 7 and 8 do for weighted instances what Sections 3 and 4 do for all pairs of inputs: Section 7 extends the structure theory to an arbitrary set of scenarios, each with its own budget, and Section 8 gives a search for this setting. We keep the case of all pairs separate because Theorem 1.2 needs nothing more, its search is simpler, and Sections 3 and 4 can be read on their own. Here we restate only the definitions and results that change. We reuse the tree of maximum paths, its pass-throughs and its notation (Section 3.1), the walk of Section 3.2, the ports of Section 3.3, the proofs of Lemmas 3.1 and 3.3(a), and Definition 3.7, as adapted in Definition 7.8. In this section we show that a feasible weighted instance has a *valid configuration*, a finite object; the search of Section 8 enumerates such objects.

Throughout, I is a weighted instance with $E \neq \emptyset$, and T with outputs o_M and o_m shows that I is feasible. Let $P' = \{x : (x, y) \in E\}$. The channels $y \notin P'$ with $(x, y) \in E$ for some x are the *blue starts*. For a channel y let $E^y = \{x : (x, y) \in E\}$, and for $x \in P'$ let $E_x = \{y : (x, y) \in E\}$.

7.1. The tree of maximum paths.

A lead of T is *red* if it lies in Π_x for some $x \in P'$, and *blue* otherwise. With this meaning of red, branch nodes, pass-throughs, blue comparators and ports are defined as in Sections 3.1 and 3.3.

Lemma 7.1. *(a) The input lead of a channel x is red exactly when $x \in P'$. Each* lo *output is blue, and* hi(g) *is red if and only if g has a red input lead.*

(b) *Join each red lead $\ell \neq o_M$ to* $\mathrm{hi}(g)$*, where g is the comparator that ℓ enters. Then the red leads form a rooted tree with root o_M whose leaves are the input leads of the channels in P'.*
(c) *There are exactly $|P'| - 1$ branch nodes. Contracting the vertices with one child in the tree of* (b) *gives a binary tree B whose leaves are the channels of P' and whose internal vertices are the branch nodes. Each pass-through lies on an edge of B or on the edge from the root of B to o_M.*

Proof. The proof of Lemma 3.1 applies to the paths Π_x with $x \in P'$. Each of them ends on o_M, because some scenario has M on x. The input lead of a channel outside P' lies on none of them, since a path Π_x starts at the input lead of x and input leads are outputs of no comparator. □

For this tree we use the notation of Section 3.1: the segments $\mathrm{seg}(u)$, the numbers $k(u)$, the pass-throughs $(u,1),\dots,(u,k(u))$, the red comparators, $u(v)$, $A(v)$ and $\mathrm{cl}(q,e)$, and, for channels $x, y \in P'$, the numbers $\delta(x)$ and $\mathrm{pre}(y,c)$, given by the same formulas. The red comparators on the path from $x \in P'$ to o_M are the comparators that M enters in the runs from x, and a pass-through $q = (w,i)$ lies on this path exactly when w is x or an ancestor of x.

7.2. **The runs.** For a blue lead ℓ, the *walk* from ℓ is defined as in Section 3.2: it starts at ℓ and, while the current lead enters a blue comparator g, moves to $\mathrm{hi}(g)$. It stops at a lead $\rho(\ell)$ that is an output or enters a comparator that is not blue, and $d(\ell)$ denotes the number of blue comparators that it passes. Since a blue lead does not enter a branch node, $\rho(\ell)$ is the port of a pass-through whenever it is not an output.

Lemma 7.2. *Suppose that m is on a blue lead ℓ in the run (x,y), where $(x,y) \in E$. Then m follows the walk from ℓ, none of the $d(\ell)$ blue comparators on it is a meeting, and one of the following holds.*

(i) $\rho(\ell)$ is an output. Then $\rho(\ell) = o_m$, and m ends there.
(ii) $\rho(\ell)$ is the port of a pass-through q that lies on Π_x. Then q is a meeting, and m leaves q on $\mathrm{lo}(q)$.
(iii) $\rho(\ell)$ is the port of a pass-through $q = (w,i)$ that does not lie on Π_x. Then $e = \mathrm{lca}_B(w,x)$ is a proper ancestor of w, m passes the $\mathrm{cl}(q,e)$ red comparators from q up to e, none of which is a meeting, meets M at e, and leaves e on $\mathrm{lo}(e)$.

Proof. The other input lead of a blue comparator is blue and does not carry M. The value m therefore leaves on hi, which is blue by Lemma 7.1(a), and it follows the walk. In case (i), m ends on $\rho(\ell)$, which is o_m because $(x,y) \in E$. In case (ii), M is on the red input lead of q. In case (iii), M is not at q. Then m leaves on the red lead $\mathrm{hi}(q)$, and w is neither x nor an ancestor of x. From then on m climbs the tree of Lemma 7.1(b). At a pass-through on the way, the other input lead is the port, which never carries M. At a branch node z strictly between w and e, the other input lead comes from the subtree of z that does not contain w. Since z is not an ancestor of x, that subtree does not contain x, and the lead does not carry M. Hence m leaves each of these comparators on hi. At e the value M arrives from the side of x and m from the side of w, and m leaves on $\mathrm{lo}(e)$. □

Lemma 7.3. *Let $(x,y) \in E$.*

(a) If $y \in P'$ and $c = \mathrm{lca}_B(x,y)$, then m passes the $\mathrm{pre}(y,c)$ red comparators below c on the path from y, none of which is a meeting, meets M at c and leaves c on $\mathrm{lo}(c)$.
(b) If m leaves a red comparator v on $\mathrm{lo}(v)$, then from there on, the comparators that m enters, and which of them are meetings, depend only on v. We write $\tau(v)$ for the number of those comparators that are not meetings.

(c) $|W(x,y)| = \delta(x)+\mathrm{pre}(y,c)+\tau(c)$ *if* $y \in P'$ *and* $c = \mathrm{lca}_B(x,y)$*, and* $|W(x,y)| = \delta(x)+r_x(y)$ *if* y *is a blue start, where* $r_x(y)$ *is the number of comparators that* m *enters in the run* (x,y) *and that are not meetings.*

Proof. (a) The proof of Lemma 3.3(a) applies. (b) After v the value M passes the comparators of $A(v)$. The comparators of Π_x below v come before v in the circuit, and m enters only comparators after v. Whether m leaves a comparator on lo therefore depends only on its current lead and on $A(v)$. (c) follows from (a), (b) and Lemma 2.1. □

For a red comparator v that m leaves on $\mathrm{lo}(v)$ in some run of a scenario, we call v *live*. Then $\mathrm{lo}(v)$ is blue, and Lemma 7.2 applies to it. If $\rho(\mathrm{lo}(v))$ is the port of a pass-through q, then q comes after v in the circuit. The comparators of Π_x that come after v are those of $A(v)$, so q lies on Π_x exactly when $q \in A(v)$. If $q \notin A(v)$, then $u(q)$ is not $u(v)$ or a descendant of $u(v)$: otherwise q would lie in $A(v)$, be v or come before v. As x is $u(v)$ or lies below it, the vertex $\mathrm{lca}_B(u(q),x)$ of Lemma 7.2(iii) is then $\mathrm{lca}_B(u(q),u(v))$, a proper ancestor of $u(v)$. This gives the recursion

$$\tau(v) = \begin{cases} d(v) & \text{if } \rho(v) = o_m, \\ d(v)+\tau(q) & \text{if } \rho(v) \text{ is the port of } q \in A(v), \\ d(v)+\mathrm{cl}(q,e)+\tau(e) & \text{if } \rho(v) \text{ is the port of } q \notin A(v), \text{ with } e = \mathrm{lca}_B(u(q),u(v)), \end{cases} \tag{2}$$

where $\rho(v) = \rho(\mathrm{lo}(v))$ and $d(v) = d(\mathrm{lo}(v))$. For a blue start y and $x \in E^y$, Lemma 7.2 applied to the input lead y gives $r_x(y) = d(y)$ if $\rho(y) = o_m$; $r_x(y) = d(y)+\tau(q)$ if $\rho(y)$ is the port of $q = (w,i)$ and x lies below w or equals it; and $r_x(y) = d(y)+\mathrm{cl}(q,e)+\tau(e)$ with $e = \mathrm{lca}_B(w,x)$ otherwise. Here $\rho(y) = \rho(\ell)$ and $d(y) = d(\ell)$ for the input lead ℓ of y.

7.3. **Three properties of feasible networks.** A pass-through q has a *climbing run* if, in the run of some scenario, m enters the port of q while M is not at q.

Lemma 7.4. *If* I *is feasible, then some network* T *with outputs* o_M *and* o_m *shows this and has a climbing run for each of its pass-throughs. In such a network no pass-through lies on* seg(*root of* B).

Proof. Let q be a pass-through without a climbing run, with red input lead a and port b. Delete q, and join a to $\mathrm{hi}(q)$ and b to $\mathrm{lo}(q)$. In the run of a scenario, if M is at q then it is on a and leaves on $\mathrm{hi}(q)$. If m enters q through b, then M is at q, because there is no climbing run, and m leaves on $\mathrm{lo}(q)$. If m enters on a, then M is not at q, since M is not on a and never on the blue lead b, and m leaves on $\mathrm{hi}(q)$. In every scenario run, M and m therefore follow the same leads as before, with q skipped, and the new network shows that I is feasible. Since the runs are the same, the other comparators keep their kinds and the remaining pass-throughs keep their climbing runs. Repeating this ends, since each step removes a comparator. A pass-through on the segment of the root has no climbing run, because every path Π_x with $x \in P'$ passes it. □

The *source leads* are the leads $\mathrm{lo}(v)$ of the live red comparators v and the input leads of the blue starts.

Lemma 7.5 (Kraft sums at a lead)**.** *For each lead* r*, the source leads* s *with* $\rho(s) = r$ *satisfy* $\sum_s 2^{-d(s)} \le 1$.

Proof. Each lead on a walk, except its first lead, is $\mathrm{hi}(g)$ for a blue comparator g. A source lead is the lo output of a red comparator or an input lead, so it lies on no walk except its own, as the first lead. The walks to r form a tree with root r whose leaves are the source leads s with $\rho(s) = r$, and the depth of s in it is $d(s)$. A vertex $\mathrm{hi}(g)$ of this tree has at most two children, the input leads of g. Give the root weight 1, and let each vertex pass its weight to its child or

split it equally between its two children. The weight of a leaf is $2^{-j} \geq 2^{-d(s)}$, where j is the number of vertices with two children above it, and the weights of the leaves sum to 1. □

Lemma 7.6. *If each pass-through has a climbing run, then each port is $\rho(s)$ for some source lead s.*

Proof. Let q be a pass-through and consider a climbing run of q. Before m enters the port of q, it is on blue leads. Let s be the last lead before the port at which m either started or arrived from a red comparator v on $\mathrm{lo}(v)$. In the first case s is the input lead of a blue start, because input leads of P' are red. In the second case v is live. In both cases s is a source lead. After s, the value m passes only blue comparators until it reaches the port: it never enters a pass-through through its port on the way, since a meeting there would give a later lo output, and so would the meeting that ends a climb (Lemma 7.2). The walk from s thus ends at the port of q. □

Lemma 7.7 (cap)**.** *For each $x \in P'$, $2^{\delta(x)} \sum_{y \in E_x} 2^{-b(x,y)} \leq 1$.*

Proof. Fix x. In the runs (x, y) with $y \in E_x$ the value M follows Π_x, and the next lead of m depends only on its current lead. The leads of m in these runs therefore form a tree with root o_m whose leaves are the input leads of the channels in E_x, since an input lead is the output of no comparator. Call a comparator *binary* if both of its input leads lie in this tree. A binary comparator does not lie on Π_x, since one of its input leads would then carry M in all these runs. In particular, it is not a meeting. With weights passed down from the root as in the proof of Lemma 7.5, the leaf y receives 2^{-j_y}, where j_y is the number of binary comparators that m enters in the run (x, y), and these weights sum to 1. By Lemma 2.1, $j_y \leq |W(x, y)| - \delta(x) \leq b(x, y) - \delta(x)$. Hence $\sum_y 2^{-(b(x,y)-\delta(x))} \leq 1$. □

7.4. **Configurations.** The following definition adapts Definition 3.7 to weighted instances. Besides adding the blue starts, it excludes pass-throughs on seg(root), as Lemma 7.4 permits, lets branch nodes be dead, and replaces the equality (K) by the inequality (V1), which suffices by Lemma 2.5(c), together with (V2). Below, a source also stands for the live red node or blue start that carries it.

Definition 7.8. A *configuration* Γ for I consists of

(i) a binary tree B whose leaves are the channels of P', each once, with an edge seg(root) from its root to o_M;
(ii) numbers $k(u) \geq 0$ for the vertices u of B, with $k(\mathrm{root}) = 0$, which give the pass-throughs (u, i) with $1 \leq i \leq k(u)$ and the red nodes as in Definition 3.7;
(iii) for each red node v, either the mark dead or a source $(\rho(v), d(v))$ as in Definition 3.7, where branch nodes may now be dead as well; the red nodes that are not dead are live;
(iv) for each blue start y, a source $(\rho(y), d(y))$.

The quantities δ, pre, cl, $u(\cdot)$ and $A(\cdot)$ are defined from (B, k) as above. A root that is a pass-through q stands for the port of q. For a live red node v whose root is a pass-through q, the *successor* of v is q if $q \in A(v)$, and $\mathrm{lca}_B(u(q), u(v))$ otherwise. For a blue start y whose root is a pass-through $q = (w, i)$ and for $x \in E^y$, the successor of y for x is q if x is w or lies below w, and $\mathrm{lca}_B(w, x)$ otherwise. The values $\tau(v)$ of the live red nodes are defined by the recursion (2), and the values $r_x(y)$ of the blue starts by the formulas that follow it; a value is ∞ if the recursion reaches a dead node. The *LCA nodes* are the branch nodes $\mathrm{lca}_B(x, y)$ with $(x, y) \in E$ and $y \in P'$. For an LCA node c let

$$\beta(c) = \min\{b(x, y) - \delta(x) - \mathrm{pre}(y, c) : \ (x, y) \in E, \ y \in P', \ \mathrm{lca}_B(x, y) = c\}.$$

Definition 7.9. A configuration is *valid* if it satisfies the following conditions.

(V1) For each root r, the sources with root r, of live red nodes and of blue starts, satisfy $\sum 2^{-d} \le 1$.
(V2) Each pass-through is the root of some source.
(V3) The directed graph on the red nodes with an arc from each red node to the next red node above it, and an arc from each live red node v to its root whenever that root is a pass-through, is acyclic.
(V4) Each LCA node is live, the successor of each live red node is live, and the successor of each blue start for each $x \in E^y$ is live.
(V5) $\tau(c) \le \beta(c)$ for each LCA node c, and $\delta(x) + r_x(y) \le b(x,y)$ for each blue start y and each $x \in E^y$.
(V6) $2^{\delta(x)} \sum_{y \in E_x} 2^{-b(x,y)} \le 1$ for each $x \in P'$.

By (V3) the successor of a live red node v lies in $A(v)$. If the root q of v is not in $A(v)$, then q is neither v nor below v, since the arc from v to q would close a cycle with the path from q up to v. As in the paragraph before (2), $u(q)$ is then not $u(v)$ or a descendant of $u(v)$, so $\mathrm{lca}_B(u(q), u(v))$ is a proper ancestor of $u(v)$. Each step of the recursion therefore moves strictly up, and τ is well defined. For $(x,y) \in E$ the *cost* is $\delta(x) + \mathrm{pre}(y,c) + \tau(c)$ with $c = \mathrm{lca}_B(x,y)$ if $y \in P'$, and $\delta(x) + r_x(y)$ if y is a blue start. Condition (V5) says that every cost is at most $b(x,y)$.

Theorem 7.10 (soundness)**.** *If I is feasible and $E \neq \emptyset$, then I has a valid configuration.*

Proof. By Lemma 7.4 we may assume that every pass-through of T has a climbing run, and then $k(\mathrm{root}) = 0$. Lemma 7.1 gives B and the numbers $k(u)$. Give each live red comparator v the source $(\rho(\mathrm{lo}(v)), d(\mathrm{lo}(v)))$, mark the other red comparators `dead`, and give each blue start y the source $(\rho(\ell), d(\ell))$ of its input lead ℓ. By Lemma 7.2(i) and the discussion before (2), each root is o_m or the port of a pass-through, and the successors in the configuration are the comparators at which m meets M next in the network. These are live, which gives (V4) together with Lemma 7.3(a). The values τ and r_x of the configuration are those of the network, by (2). By Lemma 7.3(c) the costs are then the numbers $|W(x,y)|$, and (V5) holds because T shows that I is feasible. (V1), (V2) and (V6) are Lemmas 7.5, 7.6 and 7.7. Each arc of the graph of (V3) is a directed path in the circuit: from a red comparator to the next one above it, or from v through $\mathrm{lo}(v)$ and the blue comparators of its walk into the pass-through. A cycle in this graph would give a cycle in the circuit. □

Remark 7.11. Conversely, every valid configuration is realized by a network whose runs cost at most the costs of the configuration: build the red tree with its pass-throughs, and for each root a tree of blue comparators on the lo outputs of its live red nodes and the input leads of its blue starts, at depths at most the given ones, by Lemma 2.5(c). For budgets at most 56, the program of Section 8 uses this construction to confirm each feasible answer by simulating every run. No part of the proof of Theorem 1.1 uses this converse.

8. Deciding weighted instances

The program `lb2cr` decides a weighted instance by searching for a valid configuration. If $E = \emptyset$ it answers that the instance is feasible, and if some $b(x,y)$ is negative it answers that it is infeasible. Otherwise it searches the configurations in a normal form, discards partial data that violate necessary conditions, and answers *infeasible* only if the search finishes without finding one. By Theorem 7.10, the instance is then infeasible. A third answer, *unknown*, is given when a limit on the work is reached; the verification of Section 11 treats it as a failure. This section states the normal form and the necessary conditions, proves each of them, and describes the search. We fix an instance I with $E \neq \emptyset$ and $b(x,y) \ge 0$ for all $(x,y) \in E$.

8.1. **Normal forms.** A pass-through $q = (w, i)$ is *readable* if some red node lies below it on the path to o_M, or some blue start y has an $x \in E^y$ that is w or lies below w. A branch node e is readable if some pass-through lies on the segment of a proper descendant of e. By the recursion (2) and the formulas for $r_x(y)$, the value $\tau(q)$ of a pass-through enters only the values of sources that meet at q, which lie below q, and of blue starts with x equal to w or below w, while $\tau(e)$ of a branch node enters only the values of sources whose root is a pass-through on the segment of a proper descendant of e. Only readable nodes can thus be successors. For a source s whose successor is s', we call s a *predecessor* of s'; a blue start y is a predecessor of each of its successors for the $x \in E^y$. Let

$$B_{\max} = \max\Big(\{\beta(c) : c \text{ an LCA node}\} \cup \{b(x, y) - \delta(x) : y \text{ a blue start},\ x \in E^y\}\Big).$$

A configuration is in *normal form* if, besides being valid, it satisfies:

(N3) each live red node that is not readable is an LCA node, and an LCA node c that is not readable has the largest depth for which $\tau(c) \le \beta(c)$, given its root;
(N4) $\tau(v) \le B_{\max}$ for each live red node v that is not an LCA node;
(N5) each blue start y has the largest depth for which $\delta(x) + r_x(y) \le b(x, y)$ for all $x \in E^y$, given its root.

The numbering follows (N1), $k(\text{root}) = 0$, and (N2), which is (V2).

Lemma 8.1 (deletion)**.** *Let Γ be a configuration that satisfies all conditions of validity except possibly (V2). Then deleting, one at a time, pass-throughs that are the root of no source gives a valid configuration. If Γ violates (V2), the result has fewer pass-throughs than Γ.*

Proof. Let $q = (w, i)$ be the root of no source, and delete it: $k(w)$ drops by one and the pass-throughs above q on $\text{seg}(w)$ move down one place. No successor is q, since a pass-through is a successor only of sources whose root it is. Since membership in $A(\cdot)$ is unchanged for the other red nodes and B is unchanged, every other successor stays the same. The numbers δ, pre and cl count red nodes on paths and do not increase. By induction along the recursion, no value τ or r_x increases, and no $\beta(c)$ decreases. Conditions (V4), (V5) and (V6) therefore still hold. The source of q, if any, leaves the sum at its root, so (V1) holds. A path in the new graph of (V3) comes from a path in the old one, so (V3) holds. Only (V2) can fail, at the root of q, and then we repeat. This ends, since each step removes a pass-through. □

The formal configurations of the native Lean proof (Appendix A.3) also have a *reachability condition*: every live red node can be reached from an LCA node or a blue start by a chain of successors. Deletions keep this condition: the deleted pass-through is the successor of no source, so it is not reached and hence dead, and no other successor changes.

Lemma 8.2 (normal form)**.** *Let Γ be a valid configuration whose number of pass-throughs is the least among all valid configurations for I. Then some configuration with the same B and k is in normal form.*

Proof. Each step below keeps (B, k), only marks nodes dead or increases depths, and changes the values τ and r_x only where stated. This keeps (V1), (V3) and (V6). If it left a pass-through that is the root of no source, Lemma 8.1 would give a valid configuration with fewer pass-throughs, contrary to the choice of Γ. Condition (V2) is kept as well, and we check (V4) and (V5) at each step.

(N4) Let Z be the set of live red nodes v that are not LCA nodes and have $\tau(v) > B_{\max}$. A predecessor s of a node $v \in Z$ has $\tau(s) \ge \tau(v)$, or $r_x(s) \ge \tau(v)$ if s is a blue start. It is not an LCA node, since those have $\tau \le \beta \le B_{\max}$, and not a blue start, since $r_x(y) \le b(x, y) - \delta(x) \le B_{\max}$.

Thus Z contains every predecessor of its members. Mark the nodes of Z dead. No live node outside Z has its successor in Z, so (V4) holds, and no value outside Z changes, so (V5) holds.

(N3) A live red node v that is not readable is the successor of no source. If it is not an LCA node, mark it dead: no other value changes, and (V4) holds. If it is an LCA node, its value $\tau(v)$ enters only the condition $\tau(v) \le \beta(v)$ of (V5), and we raise its depth as far as that condition allows. This changes no other value. These changes do not affect (N4).

(N5) Since no source has a blue start as its successor, raising the depth of a blue start changes only its own values $r_x(y)$. We raise each depth as far as (V5) allows. □

Lemma 8.3. *Let Γ be as in Lemma 8.2. Then each pass-through $q = (w, i)$ of Γ has an* affordable climber*: an LCA node c such that q neither lies on the path from c to o_M nor below c, and $\beta(c) \ge \mathrm{cl}(q, \mathrm{lca}_B(w, c))$; or a blue start y and an $x \in E^y$ that is not w and not below w, with $b(x, y) - \delta(x) \ge \mathrm{cl}(q, \mathrm{lca}_B(w, x))$.*

Proof. A source with root q *climbs* through q if its successor is not q; for a blue start this means that its successor for some $x \in E^y$ is a branch node. First, some source climbs through q. Otherwise every source s with root q has successor q, and q is live by (V4). Give each such s the root $\rho(q)$ and the depth $d(s) + d(q)$. If $\rho(q)$ is o_m or a pass-through above q, then s meets there as q did. If $\rho(q)$ is a pass-through q' not above q, then q climbs to $e = \mathrm{lca}_B(u(q'), w)$, and since s lies below q, or x lies below w for a blue start, the climb from q' also ends at e for s. In each case the value of s is unchanged. The sum at $\rho(q)$ changes by $2^{-d(q)}(\mu - 1) \le 0$, where $\mu \le 1$ is the sum at q. Each new arc of (V3) replaces a path through q. Then mark q dead. Now q is the root of no source, and Lemma 8.1 gives a valid configuration with fewer pass-throughs, a contradiction.

Let U be the set of sources that climb through q, and let Z be the set of sources from which a chain of successors reaches a member of U. Suppose that Z contains an LCA node or a blue start c, whose chain reaches $t \in U$. If t is a red node, then $u(q) = w$ is not $u(t)$ or a descendant of $u(t)$ (see the remarks after Definition 7.9). If c is an LCA node, then c is t or lies below t, so the climb from q ends at the same branch node e for c as for t. Moreover q does not lie on the path from c to o_M: it would then lie above t, so that t meets at q, or at or below t, which is excluded by (V3). Nor does q lie below c, since it would then lie below t. If $c = y$ is a blue start, choose $x \in E^y$ as follows: if $t = y$, an x for which the successor of y is a branch node; otherwise an x to which the chain from y to t belongs. In the second case every node of the chain lies on the path from x to o_M, so x lies in the subtree of $u(t)$, and w is not x and not an ancestor of x, since q would otherwise lie in $A(t)$. In both cases x is not w and does not lie below w, and the climb from q ends at the same branch node e for c as for t. In all cases the value of c is at least that of t, which is at least $\mathrm{cl}(q, e)$, and the budget of c is at least its value. Hence c is an affordable climber. Otherwise Z contains no LCA node and no blue start. Mark the nodes of Z dead. No live node outside Z has its successor in Z, so (V4) holds, and no value outside Z changes. If a pass-through became the root of no source, Lemma 8.1 would contradict the choice of Γ. The result is therefore valid, with the same (B, k), and no source climbs through q, which is excluded by the first part. □

8.2. **Necessary conditions on the tree and the pass-throughs.** For $x \in P'$ let η_x be the largest integer a with $2^a \sum_{y \in E_x} 2^{-b(x,y)} \le 1$. As in Section 2.4, $D(u)$ denotes the depth of a vertex u of B, the root having depth 0.

Lemma 8.4. *Every valid configuration satisfies the following.*

(a) $D(x) \le \delta(x) \le \eta_x$ for each $x \in P'$. Moreover, if numbers $c(x) \ge D(x)$ are given for the leaves x of a subtree of B whose root has depth t, then $\sum_x 2^{-(c(x)-t)} \le 1$ over these leaves, and $c(x) - t \ge 1$ if the subtree has at least two leaves.

(b) $\delta(x)+\mathrm{pre}(y,c)\le b(x,y)$ *for each* $(x,y)\in E$ *with* $y\in P'$ *and* $c=\mathrm{lca}_B(x,y)$. *In particular* $D(x)+D(y)\le b(x,y)+D(c)+1$.
(c) $\beta(c)\ge 0$ *for each LCA node* c, *and* $\delta(x)\le b(x,y)$ *for each blue start* y *and* $x\in E^y$.

Proof. (a) Each proper ancestor of x contributes at least 1 to $\delta(x)$, and (V6) gives $\delta(x)\le\eta_x$. The leaves of a subtree satisfy Kraft's equality $\sum 2^{-(D(x)-t)}=1$ (Lemma 2.5(a)), and $D(x)-t\ge 1$ if there are at least two of them. (b) and (c) follow from (V5), since τ and r_x are nonnegative, and from $\mathrm{pre}(y,c)\ge D(y)-D(c)-1$. □

Since $\delta(x)=D(x)+\sum_u k(u)$ over x and its proper ancestors, the conditions of (b) and (c), and those of (a) other than the subtree condition, are linear inequalities in the numbers $k(u)$, with coefficients 0 and 1. For a segment w, let $k_{\max}(w)$ be the least right-hand side among the inequalities of (a) and (b) that contain $k(w)$.

8.3. **Credit bounds.** The following bounds compare the budgets with what the forest of sources can carry. Fix a valid configuration. Give each live red node v the *credit* $\kappa(v)=2^{-\tau(v)}$. For a blue start y fix $x^*(y)\in E^y$ with the least value of $b(x,y)-\delta(x)$, put $\beta(y)=b(x^*(y),y)-\delta(x^*(y))$, and let $\kappa(y)=2^{-r_{x^*(y)}(y)}$. The *parent* of a live red node is o_m if its root is o_m, and its successor otherwise. The parent of a blue start y is defined in the same way, with its successor for $x^*(y)$. Parents are live by (V4), and they lie above their children on the path to o_M, so the parents form a tree with root o_m, the *flow tree*. Put $\kappa(o_m)=1$. By (2), each child s of a node p satisfies

$$\kappa(s)=\kappa(p)\,2^{-d(s)-\gamma(s)}, \tag{3}$$

where $\gamma(s)=\mathrm{cl}(q,p)$ if s climbs to p through its root q, and $\gamma(s)=0$ otherwise. The LCA nodes and the blue starts are *required*. The LCA nodes are live by (V4), and the total credit R^* of the required nodes satisfies

$$R^*\ \ge\ \Omega:=\sum_c 2^{-\beta(c)}+\sum_y 2^{-\beta(y)} \tag{4}$$

by (V5), where c runs over the LCA nodes and y over the blue starts.

Lemma 8.5 (gain bound)**.** $R^*\le 1+\sum_w\bigl(1-2^{-k(w)}\bigr)$, *where* w *runs over the vertices of* B.

Proof. Sum (3) over all nodes of the flow tree other than o_m, and group the terms by the parent. The children of o_m contribute at most 1, by (V1). The children of a live pass-through q meet at q and contribute $\kappa(q)\mu_q$, where $\mu_q\le 1$ is their part of the sum at q. The nodes that climb contribute a total C. Since every live pass-through is a node of the flow tree that is not required,

$$R^*\ \le\ 1+\sum_q\kappa(q)\mu_q+C-\sum_q\kappa(q)\ \le\ 1+C.$$

A source that climbs through $q=(w,i)$ to p has $\gamma=\mathrm{cl}(q,p)\ge 1+k(w)-i$ and $\kappa(p)\le 1$, and the sources with root q have total sum at most 1. Hence $C\le\sum_{(w,i)}2^{-(1+k(w)-i)}=\sum_w(1-2^{-k(w)})$. □

The search fixes the numbers $k(w)$ one segment at a time. Suppose some segments are *fixed* and the others are *free*, with value 0 for now; a completion gives the free segments values $k'(w)\le k_{\max}(w)$. For each required node o let λ_o be the inequality of Lemma 8.4(b) or (c) for a pair that attains the minimum in the definition of $\beta(o)$, so that its right-hand side minus its left-hand side at the current k equals $\beta(o)$, and let $\Delta_w=\sum 2^{-\beta(o)}$ over the o whose λ_o contains $k(w)$. Write $\Omega(k)$ for the value of (4) at the current k.

Lemma 8.6 (lifted gain bound)**.** *If some completion has a valid configuration, then*

$$1-\Omega(k)+\sum_{w\ \mathit{fixed}}\bigl(1-2^{-k(w)}\bigr)+\sum_{w\ \mathit{free}}\max_{0\le k'\le k_{\max}(w)}\Bigl(\bigl(1-2^{-k'}\bigr)-\Delta_w\bigl(2^{k'}-1\bigr)\Bigr)\ \ge\ 0.$$

Proof. In a completion k', each $\beta(o)$ is at most the value of λ_o (its right-hand side minus its left-hand side), which is $\beta(o)$ at k minus the sum of $k'(w)$ over the free segments in λ_o; if the completion has a valid configuration, these values are nonnegative by Lemma 8.4. Since $2^{a+b}-1\ge(2^a-1)+(2^b-1)$ for $a,b\ge0$, this gives $\Omega(k')\ge\Omega(k)+\sum_{w\text{ free}}\Delta_w(2^{k'(w)}-1)$. Combine with (4) and Lemma 8.5 at k'. □

For a vertex u of B, the *region* of u is the set of red nodes v with $u(v)$ in the subtree of u. Define Ψ on the vertices of B from the leaves up: $\Psi(u)=1$ for a leaf, and for an internal vertex e with children a and b,

$$\Phi_e=[e\text{ is an LCA node}]+\Psi(b)S_a(e)+\Psi(a)S_b(e),\qquad \Psi(e)=\max\bigl(\Psi(a),\Psi(b),\Phi_e\bigr),$$

where $S_a(e)=\sum 2^{-\mathrm{cl}(q,e)}$ over the pass-throughs q in the region of a, and similarly $S_b(e)$.

Lemma 8.7 (amplification bound)**.** $R^*\le\Psi(\mathit{root})$*. If the root ϱ of B is an LCA node, put $\omega_{\min}=2^{-\beta(\varrho)}$ and $\Psi'=\max(\Psi(a),\Psi(b))$ for the children a,b of ϱ. Then $R^*\le\Phi_\varrho$ if $\Phi_\varrho\ge\Psi'$, and $R^*\le\omega_{\min}\Phi_\varrho+(1-\omega_{\min})\Psi'$ otherwise.*

Proof. For a node s of the flow tree other than o_m, let $\phi(s)$ be the total credit of the required nodes in the subtree of s, divided by $\kappa(s)$. We show $\phi(v)\le\Psi(u(v))$ for each live red node v, by induction from the bottom of the flow tree; note that Ψ does not decrease towards the root and that $\Psi\ge1$. A blue start has no children and has $\phi=1$. A live pass-through q is not required, and its children meet at q. They lie below q in the region of $u(q)$, and their part of the sum at q is at most 1, so $\phi(q)$ is at most the largest ϕ of a child, hence at most $\Psi(u(q))$. A live branch node e with children a and b has as its children in the flow tree the sources that climb to e. Such a source s climbs through a pass-through q in the region of one child, say a. Its successor e is $\mathrm{lca}_B(u(q),u(s))$, and $u(q)$ is not $u(s)$ or a descendant of $u(s)$ (see the remarks after Definition 7.9), so s lies in the region of b; or s is a blue start. Its value of ϕ is then at most $\Psi(b)$. Since the sources with root q have total sum at most 1,

$$\phi(e)\ \le\ [e\text{ is an LCA node}]+\Psi(b)S_a(e)+\Psi(a)S_b(e)=\Phi_e\le\Psi(e).$$

Now $R^*=\sum_s\kappa(s)\phi(s)=\sum_s2^{-d(s)}\phi(s)$ over the children s of o_m. By (V1) this is at most the largest $\phi(s)$, hence at most $\Psi(\text{root})$.

If the root ϱ of B is an LCA node, its root is o_m: since $k(\varrho)=0$, no pass-through lies above it, and a pass-through below it cannot be its root (remarks after Definition 7.9). So ϱ does not climb. Its term is $\omega\phi(\varrho)$ with $\omega=2^{-d(\varrho)}\ge2^{-\tau(\varrho)}\ge\omega_{\min}$ and $\phi(\varrho)\le\Phi_\varrho$. The other children of o_m lie in the regions of a and b or are blue starts, so their terms total at most $(1-\omega)\Psi'$. Being linear in $\omega\in[\omega_{\min},1]$, the bound $\omega\Phi_\varrho+(1-\omega)\Psi'$ is largest at an endpoint. □

With $H(e)=\sum_z(1+k(z))$ over the internal vertices z from e up to the root, the pass-throughs on the segment of a vertex w in the region of a contribute $(1-2^{-k(w)})2^{-(H(\mathrm{par}\,w)-H(e))}$ to $S_a(e)$, where $\mathrm{par}\,w$ is the parent of w. For a partial assignment of the numbers $k(w)$, the search evaluates the *lifted form* of this bound, the bound with $k_{\max}(w)$ pass-throughs on each free segment, with H and $\beta(\varrho)$ at the current k. In every completion, Ω is at least its current value and R^* is at most this bound: more pass-throughs give more terms, the current H gives the shortest climbs, and the current $\beta(\varrho)$ gives the smallest $\omega_{\min}$.

During the search over sources, the red nodes are assigned in an order in which each node comes after the nodes above it. At a given point, let $\mathrm{free}(r)$ be 1 minus the sum of 2^{-d} over the

sources already assigned to the root r. For a pass-through $q = (w, i)$ put $F_q = 2^{-(1+k(w)-i)}$, or $F_q = \max(\kappa(q), 2^{-(1+k(w)-i)})$ if q is assigned and live.

Lemma 8.8 (credit capacity)**.** *Every valid completion of the current partial assignment satisfies*

$$\sum_s 2^{-\beta(s)} \;\le\; \mathrm{free}(o_m) + \sum_q \mathrm{free}(q)\, F_q,$$

where s runs over the required nodes not yet assigned.

Proof. For an unassigned required node s, follow the parents in the flow tree as long as the step goes from a child that meets at a pass-through not yet assigned. Let p be the node where this stops. Then p is unassigned, and either its parent is o_m, or it meets at an assigned pass-through q, which is live by (V4), or it climbs through some pass-through q. By (3), $\kappa(p)$ is at most $2^{-d(p)}$, $2^{-d(p)}\kappa(q)$ or $2^{-d(p)}2^{-(1+k(w)-i)}$ in these three cases. Group the required nodes by this node p. In the subtree of p formed by the steps that meet at unassigned pass-throughs, the required nodes are leaves, since sources meet only at pass-throughs, while the required nodes are branch nodes or blue starts. The sums at the pass-throughs are at most 1, so the required nodes in this subtree have total credit at most $\kappa(p)$. Since the nodes p are unassigned, their sums $2^{-d(p)}$ at a root r total at most $\mathrm{free}(r)$. Finally $2^{-\beta(s)} \le \kappa(s)$ by (V5). □

8.4. **Symmetry.** Two channels are *interchangeable* if exchanging them maps the function V on E to itself. This is an equivalence relation, and the product G of the symmetric groups of its classes consists of automorphisms of I. A class that meets P' lies inside P', and $V(gx, gy) = V(x, y)$ for $g \in G$. Relabelling a valid configuration by an element of G gives a valid configuration, with the same number of pass-throughs, whose tree is the image of B, and the conditions (N3)–(N5) and the conclusion of Lemma 8.3 are preserved. Two trees whose leaves are labelled by the channels of P' lie in one G-orbit exactly when they are isomorphic as trees with leaves coloured by the classes. It therefore suffices to examine one labelled tree for each coloured isomorphism class. Blue starts in one class have the same sets E^y and the same budgets. Exchanging their sources therefore keeps a configuration valid and in normal form, and we may assume:

(N6) within each maximal run of consecutive blue starts of one class, in the order of the search, the roots do not decrease in a fixed order of the roots.

The program computes the classes from the scenarios themselves.

8.5. **The search.** The program proceeds as follows.

(1) Compute the numbers η_x. Answer infeasible if some $\eta_x < 0$ or $\sum_x 2^{-\eta_x} > 1$ (Lemma 8.4(a)).

(2) Enumerate the coloured binary trees on P', one for each isomorphism class, from the root down. Each leaf x has a cap on its depth, at first η_x. When the second subtree of a vertex c is enumerated, the depths of the leaves of the first subtree are known, and the cap of each leaf y of the second subtree is lowered to the least value of $b + D(c) + 1 - D(x)$, where x runs over the leaves of the first subtree and b over the budgets $b(x, y)$ and $b(y, x)$ of the pairs in E (Lemma 8.4(b), whose left side is symmetric in x and y). Because V is invariant under G, these budgets depend only on the classes of x and y, and the program uses the deepest leaf of each class. The subtree condition of Lemma 8.4(a) is applied with the current caps.

(3) For each tree, enumerate the numbers $k(w)$ on the segments other than that of the root, from the top down, under the linear inequalities of Lemma 8.4. At each node of this enumeration, including the first, discard the partial assignment if Lemma 8.6 or the lifted form of Lemma 8.7 fails.

(4) For each (B, k), compute δ, pre, β and the budgets of the blue starts, and discard (B, k) if one of them is negative, if some pass-through has no affordable climber (Lemma 8.3), or if Lemma 8.5 fails with R^* replaced by Ω. Then search the red nodes depth first, in an order in which each node comes after the nodes above it. A node that is not an LCA node may be `dead`. Otherwise it receives a root, o_m or a pass-through allowed by (V3), whose successor is live, and a depth. Its value τ is then determined, and it must satisfy $\tau \le \beta$ for an LCA node and $\tau \le B_{\max}$ otherwise. A node that is not readable receives only the choices allowed by (N3). Depths are tried from the largest allowed value down, and the loop stops when the sum at the root would exceed 1. The condition of Lemma 8.8 is tested whenever some red node is still unassigned. Condition (V3) is enforced in part during the search, by excluding roots at or below the node, and in full in step (6).
(5) Then search the blue starts depth first. Each receives a root whose successors are live and the largest depth allowed by (V5), subject to (N6) and to (V1).
(6) At a complete assignment, check (V2) and (V3), evaluate every cost again with a separate implementation of the recursion, build the network of Remark 7.11, and simulate the run of every scenario. Answer feasible. If the two evaluations disagree or a simulated run fails, the program stops with an error.

If the search ends without a complete assignment that passes step (6), the answer is infeasible.

Proposition 8.9. *If* `lb2cr` *answers infeasible for an instance whose budgets are all at most* 56*, then the instance is infeasible.*

Proof. Suppose the instance is feasible. Then $E \neq \emptyset$, since `lb2cr` answers feasible when $E = \emptyset$, and all $b(x, y) \ge 0$, since $|W(x, y)| \ge 0$. By Theorem 7.10 there is a valid configuration. Take one with the fewest pass-throughs, put it in normal form by Lemma 8.2, and relabel it by an element of G so that its tree is the one listed in step (2), then apply (N6). By Lemmas 8.3 to 8.8, no test of steps (1)–(5) discards it or one of its partial assignments, and each choice it makes is offered. The search then reaches it in step (6). □

The bound 56 comes from the arithmetic. Kraft sums are computed exactly as integers in units of 2^{-56}, and a depth above 56 gets weight 0, which only weakens each test. The credit bounds use exact 128-bit integers, except the amplification bound. That bound is computed in floating point from nonnegative dyadic terms, and the program prunes only if Ω exceeds it by a relative 10^{-9} plus 10^{-12}, far more than the rounding error. Rounding is toward the weaker test except in four places: the lifted gain bound maximizes over $k' \le 56$ only, the amplification bound would overflow once H exceeds about 1000 and caps $\beta(\varrho)$ at 60, and the number 2^{28} marks the absence of a budget. None of them matters when every budget is at most 56, because $k_{\max}(w)$, H and $\beta(\varrho)$ are at most $\max b(x, y)$. With such budgets every depth in the search is at most 56 as well, so step (6) builds and simulates a network for every feasible answer. The program also answers unknown for malformed input.

9. Lower bounds for cores

As weights of the instances in the search we use lower bounds on the costs of cores. We obtain them in three ways.

The trivial bound. Let X be produced by a prefix P of k standard comparators on 13 channels, and let (x, y) be a scenario of (X, ϵ). By Lemma 5.8 and $S(11) = 35$ [8], its core satisfies

$$s(Y_{xy}) \;\ge\; 35 - (k - w_\epsilon(x, y)),$$

where $w_\epsilon(x, y)$ is the largest value of $|W(i, j)|$ over the runs (i, j) of polarity ϵ of P that end on (x, y). If no such run ends on (x, y), the trivial bound is 0. Since $k \le 25$ in the search, the trivial bound is 0 or at least 10.

Instances on 11 *channels.* A core Y on 11 channels has scenarios of its own, whose cores lie on 9 channels. Proposition 6.2 applied to Y gives: if $u_t \le s(Z_t)$ for the cores Z_t of the scenarios of (Y, ϵ'), and the instance $I(Y, \epsilon'; u; \theta - 1)$ is infeasible, then $s(Y) \ge \theta$. Each weight u_t is justified in one of two ways.

(E) u_t is at most the exact value $s(Z_t)$, computed by Harder's program.

(B) For some polarity ϵ'', the instance $I(Z_t, \epsilon''; z; u_t - 1)$ on 9 channels is infeasible, where z consists of the exact values of the 7-channel cores of (Z_t, ϵ''), computed by Harder's program. Then $s(Z_t) \ge u_t$ by Proposition 6.2.

Sharing. By Lemma 5.1(c), a bound on $s(Y)$ holds for every set similar to Y. A bound proved for the core of one scenario of one prefix thus serves every scenario, of any prefix, whose core is similar to it. In particular, a trivial bound found for one prefix can be larger than the trivial bound of another prefix with a similar core.

Harder's program computes $s(Z)$ for a well-behaved set Z by the recursion $s(Z) = 1 + \min s(Z')$ over the successors Z' of Z, taken up to similarity, with lower bounds from the Huffman bound [8, Lemma 12, Theorem 26, Corollary 56]. By Lemma 5.4 all sets on 9 and 7 channels that we pass to it are well-behaved. We changed the program so that it handles sets on up to 13 channels, and added a command-line front end that prints $s(Z)$ for a set given as a bitmap. When the changed program stores a state a second time, it keeps the larger of the two lower bounds and the smaller of the two upper bounds. For the tables of exact values (Section 11) no bounds are supplied in advance, and then every update of a state starts from the stored state and never weakens its bounds, so for these tables the rule has no effect.

The search computes these bounds as needed. For a child at depth k it first uses the trivial bounds and the bounds already proved for similar cores. If the instance on 13 channels is still feasible, the instance solver of the search returns a configuration, and for the scenarios whose costs in it reach their budgets, the search tries to raise the bounds on their cores through instances on 11 channels. For the weights on 9 channels it uses the exact values already known, including a table loaded at the start, and bounds of type (B) otherwise, and it computes exact values for the scenarios on 9 channels whose costs reach their budgets.

10. The search over prefixes

The search follows the generate-and-prune approach of Codish, Cruz-Filipe, Frank and Schneider-Kamp [2], which builds the sets of outputs of all prefixes layer by layer and keeps one representative of each class of equivalent sets. We do not use the subsumption test of [2]; instead, as in Harder's algorithm [8, Section 5], a set is discarded when a lower bound on its cost exceeds the remaining budget. Sets are identified up to the symmetries of [8, Corollary 6], namely permutations of the channels and complementation.

A *state* of depth k is a set $X \subseteq \{0,1\}^{13}$ produced by a prefix of k standard comparators, stored together with that prefix. The search produces *layers* $L_0, L_1, \ldots$, which are finite lists of states, the states of L_k having depth k, with $L_0 = \{\{0,1\}^{13}\}$. For a state $R \in L_{k-1}$ and a standard comparator (i, j) that is useful for R, the *child* $R(i, j)$ has depth k, the prefix of R followed by (i, j), and the budget $44 - k$. Given the layers, a *record* of a child is *valid* if it is one of the following:

- *cut*: for some polarity ϵ and weights v that are lower bounds on the costs of the cores (Section 9), the instance $I(R(i, j), \epsilon; v; 44 - k)$ is infeasible, so $s(R(i, j)) \ge 45 - k$ by Proposition 6.2;
- *alive*: a set similar to the child lies in L_k;

- *duplicate*: the record names another child. Each duplicate record on the way names a child that is similar to its own child and has at most its depth, and after finitely many steps one reaches a valid cut or alive record.

The search builds L_k from the alive children of depth k, one for each canonical form up to similarity (Section 10.1), and stops at the first empty layer.

Theorem 10.1. *Suppose that every child of every state in $L_0, \dots, L_{24}$ has a valid record, that no state in these layers has 14 or fewer vectors, and that L_{25} is empty. Then $S(13) \geq 45$.*

Proof. Suppose $S(13) = s \leq 44$, and let $g_1, \dots, g_s$ be a standard sorting network on 13 channels with s comparators (Lemma 5.1(a)). Let X_j be the set produced by $g_1, \dots, g_j$. Then $s(X_j) = s - j$: the remaining comparators sort X_j, and a shorter network sorting X_j, preceded by $g_1, \dots, g_j$, would sort $\{0,1\}^{13}$ with fewer than s comparators. Each g_{j+1} is useful for X_j: otherwise X_{j+1} equals X_j or arises from it by exchanging two channels, and $s(X_{j+1}) = s(X_j)$ by Lemma 5.1(c).

We show by induction on $j \leq s$ that some layer L_d with $d \leq \min(j, 24)$ contains a state similar to X_j. For $j = 0$ this is L_0. Let $R \in L_d$, $d \leq j$, be the image of X_j under a permutation of the channels possibly combined with complementation, and let $g_{j+1} = (a, b)$. The same map takes X_{j+1} to the image of R under a comparator (a', b'), where $\{a', b'\}$ is the image of $\{a, b\}$. If $a' < b'$, this image arises from the image under the standard comparator (b', a') by exchanging channels a' and b'. Hence R has a child $C = R(p, q)$, with $\{p, q\} = \{a', b'\}$ and $p > q$, that is similar to X_{j+1}. The comparator (p, q) is useful for R, because g_{j+1} is useful for X_j and the map preserves this property. The child C has depth $d+1 \leq j+1$, and $s(C) = s(X_{j+1}) = s-j-1 \leq 44-(d+1)$. Hence its record is not a cut. If it is alive, L_{d+1} contains a set similar to C. If it is a duplicate, the names lead to a child C' of depth $d' \leq d+1$ that is similar to C and has a valid cut or alive record. Since $s(C') = s(C) \leq 44 - d'$, that record is not a cut but alive, and $L_{d'}$ contains a set similar to C. In both cases this layer has depth at most $d+1 \leq 25$ and is not empty, so its depth is at most 24.

For $j = s$ the set X_s consists of the 14 sorted vectors, so some layer L_d with $d \leq 24$ contains a state with 14 vectors. This contradicts the hypothesis. □

Proof of Theorem 1.1. Juillé's network [11] gives $S(13) \leq 45$, and we check this in Lean (Appendix A.5). The certificate (Section 11) records the layers $L_0, \dots, L_{25}$ and a record for every child of every state in $L_0, \dots, L_{24}$, and its verification confirms the hypotheses of Theorem 10.1, using the components listed in Section 13. Hence $S(13) \geq 45$. □

10.1. **The program.** Our search program is a modification of Harder's program sortnetopt [9], from which it takes the representation of sets as bitmaps, the canonical form of a set up to similarity and the search for exact values. The layered search, the instance solver and the code that writes the certificate are new. For each child the program builds the two instances on 13 channels, one for each polarity, first with the trivial bounds and the bounds already known for similar cores, and then with bounds raised by instances on 11 channels, as described at the end of Section 9. It decides them with its instance solver, which the verification does not use. When the solver reaches a limit on its work, the child is kept. Each process keeps its own table of the best bound known for each core in canonical form and reuses a bound found for a core in every later scenario whose core is similar to it.

Duplicates are detected within one process: the canonical form of each child is looked up among those of the children already generated by the same process, at any depth, and a repeated child refers to the first one. We computed layers 1 to 12 with one process, which used only the trivial bounds and their sharing and needed no exact values. From depth 13 on, the states of each layer were divided among several processes, each of which kept its tables from layer to layer,

and a coordinator merged the alive children of all processes, kept one state for each canonical form, and divided the result again as the next layer.

11. The certificate and its verification

11.1. **The records.** Each process of the search writes, for its share of a layer or for a part of such a share, one output, a record for each child that it generates; each process also writes a file of bounds on cores, the *claims*. A share whose process failed was computed again in 2, 4 or 8 parts, and the last line of the coordinator's log that concerns the share names the outputs used; every other share uses the output of its own process. A record of kind `A` says that the child is alive and gives a map that takes it to a set listed in the merged set of its layer. A record of kind `D` says that the child is a duplicate: it names another child of the same process and gives two maps that take the two children to the same set. A record of kind `T` or `C` says that the child is cut and gives the polarity ϵ and the weights v_t of an infeasible instance, with, for each scenario whose weight exceeds its trivial bound, a claim of bound at least v_t and a map from the core of the scenario to the core of the claim; `C` marks the cuts found after bounds were raised by instances on 11 channels. A claim of kind `K T` states a bound $s(Y) \geq \theta$ for a core Y, justified by the trivial bound of a named scenario of a child of the same process, with the polarity and a map that takes the core of the scenario to Y; one of kind `K L` states such a bound justified by an instance on 11 channels, given by its polarity and weights, each weight tagged `E` or `B` for the justifications (E) and (B) of Section 9. Each claim has an identifier, by which the records name it; the claims also list the cores on 11 channels. The coordinator writes the list of states of each layer, the share or part of each process, the log of the run, and the merged set of each layer, which lists each state with its canonical form, the image of the state under the map of the alive record that produced it. Layers 1 to 12 consist of the alive children of the one process that computed them. The supplement [17] specifies the format in full, so that a verifier needs nothing from the search code.

11.2. **The verifier.** The verifier consists of a *record checker*, which checks the records and claims of the processes, and a *layer checker*, which checks the layers as a whole. We wrote both from the format specification and the mathematics in Sections 5 to 9, without the search code. They decide every instance with `lb2cr`, which we wrote from a separate derivation of the mathematics in Sections 7 and 8, without the code of the search's instance solver, and they ask Harder's program for exact values. For each layer of each process, the record checker checks the following.

(i) In layers 2 to 12 the input of the layer is the list of alive children of the previous layer of the same process. In the other layers the layer checker matches it with the share, or part of a share, named in the coordinator's files. Each state of the input is the set produced by its prefix, which consists of standard comparators and has the length of the depth. The only state of depth 0 is the full set.

(ii) Each standard comparator that is useful for a state gives exactly one record, and no record names a comparator that is not useful.

(iii) For a cut, it computes the trivial bounds of all scenarios of the given polarity from the prefix, checks that each weight above its trivial bound is covered by the named claim and map, and checks that `lb2cr` answers infeasible for the instance. The program computes the symmetries of each instance itself and ignores any hint in the record.

(iv) For an alive child, it checks that the child has more than 14 vectors and that its image under the map is listed in the merged set of its layer.

(v) For a duplicate, it checks that the two maps give the same set, that the named child has a depth at most that of the duplicate, and that following the names ends at a verified alive or cut record.

For each claim that a record uses, the record checker checks the following.

(vi) For `K T`, it recomputes the trivial bound of the named scenario and applies the map.
(vii) For `K L`, it checks that `lb2cr` answers infeasible for the instance on 11 channels with budget $\theta - 1$. For a weight tagged `E`, it checks that the weight is at most the exact value of the core on 9 channels, or at most a bound of type (B) for it. For a weight tagged `B`, it checks (B) for one of the two polarities on a set similar to the core, with exact values of the cores on 7 channels.

In all of these, maps must be permutations of the channels, with or without complementation, and the identifiers of claims must be unique. Since Proposition 8.9 covers budgets up to 56, every instance passed to `lb2cr` must have budgets between 0 and 56. The verifier records the largest budget of each kind of instance. The layer checker checks that the shares split each layer, and the parts of a share computed again split that share, as the coordinator's files say; that the outputs used are those of the share's own process or, for a share computed again, those named in the log; that every state of a layer is covered; that layers 1 to 12 end in the states of depth 12 from which layers 13 on start; that every line of a merged set comes from an alive record of a used output, and that the images of these alive records are exactly the listed canonical forms; that each layer is the merged set of the previous one; and that layer 25 is empty.

Two of the three processes that failed at depth 14 were restarted and also finished their shares. The verifier also checks the records of these two unused outputs, which later duplicates of the same processes may name; some of these records would not pass as records of a used output, and no record of a used output depends on them.

11.3. **Exact values.** We computed the exact values in advance for the cores on 9 and 7 channels whose values the verifier needs, and stored them in two tables. Each entry gives the value for one image of the core under a map of the kind above, computed by the verifier before the look-up. This image is not always a canonical form, but by Lemma 5.1(c) any image will do. The claims use 15,430,581 cores on 9 channels, counted up to the verifier's representatives, and 3,624,738 of them needed exact values. For the others, bounds of type (B) cover every weight of the claims that use them. For each core with an exact value e, the verifier also passes the two instances on 9 channels with budget e to `lb2cr`, with the exact values on 7 channels as weights, and requires that neither is infeasible. An infeasible answer would show that `lb2cr` or Harder's program is wrong.

12. Results

We list the layers in Table 3. The search went extinct at depth 25, which gives the last hypothesis of Theorem 10.1: the three states of depth 24 have 134 children, all of them cut. In total the outputs used contain 41,142,659 records of children, of which 31,899,147 are cuts, 2,120,883 alive and 7,122,629 duplicates. The layers $L_1, \ldots, L_{24}$ contain 662,820 states.

Deeper in the search, more of the cuts were found only after bounds were raised by instances on 11 channels: the cuts of kind `C` make up 9% of the cuts at depth 13 but between 22% and 30% from depth 16 on. The largest budgets of the instances that the verifier passed to `lb2cr` were 41, 28 and 25 on 13, 11 and 9 channels, so all of them lie within the range of Proposition 8.9.

13. Trusted components and cross-checks

The proof of the lower bound in Theorem 1.1 rests on Sections 2 and 3, Sections 5 to 10 and the verification of Section 11. The verification checks every record and every claim that a record uses. It does not check the following three components.

k	$\lvert L_{k-1}\rvert$	children	alive	duplicate	cut `T`	cut `C`	$\lvert L_k\rvert$
1	1	78	1	77	0	0	1
2	1	77	2	75	0	0	2
3	2	152	5	146	1	0	5
4	5	375	13	358	4	0	13
5	13	961	36	902	23	0	36
6	36	2,624	106	2,389	129	0	106
7	106	7,617	300	6,616	701	0	300
8	300	21,234	823	17,186	3,225	0	823
9	823	57,392	2,398	42,095	12,899	0	2,398
10	2,398	164,723	7,311	107,586	49,826	0	7,311
11	7,311	494,205	21,956	282,325	189,924	0	21,956
12	21,956	1,459,781	65,659	717,716	676,406	0	65,659
13	65,659	4,292,552	318,344	1,163,812	2,544,340	266,056	100,762
14	100,762	6,475,632	490,716	1,138,486	4,241,800	604,630	142,881
15	142,881	9,006,443	489,270	1,243,569	5,938,250	1,335,354	133,426
16	133,426	8,213,431	358,766	1,072,034	5,263,682	1,518,949	94,630
17	94,630	5,667,481	212,574	719,135	3,513,826	1,221,946	54,471
18	54,471	3,167,491	99,158	374,536	1,942,775	751,022	24,920
19	24,920	1,405,142	37,391	156,818	859,050	351,883	9,250
20	9,250	504,626	11,567	53,685	312,013	127,361	2,812
21	2,812	147,750	3,403	16,976	92,216	35,155	807
22	807	40,768	892	4,743	25,922	9,211	208
23	208	10,114	174	1,132	6,635	2,173	40
24	40	1,876	18	232	1,232	394	3
25	3	134	0	0	103	31	0

TABLE 3. The layers of the search. Row k counts the children of the states of L_{k-1} by the kind of their record. One process computed layers 1 to 12. From layer 13 on, alive children of different processes can be similar, and $\lvert L_k\rvert$ counts their similarity classes. Duplicates are found within one process.

(1) *Harder's program*, with the changes described in Section 9, used for the exact values of the sets on 9 and on 7 channels in the two tables of Section 11. The proof that this algorithm is correct requires well-behaved sets, and these sets are well-behaved (Lemma 5.4). The same search for exact values computes $S(11) = 35$ in [8], where its results are backed by certificates; the values in our tables have no certificates.
(2) *The bound* $S(11) = 35$, used by every trivial bound. Section 1 and Appendix A.2 describe the checks of Harder's certificate for $S(11) \geq 35$.
(3) *The program* `lb2cr`, for the answer infeasible on each instance that the verifier passes to it. Its correctness for budgets at most 56 is the content of Sections 7 and 8 (Proposition 8.9).

We trust the verifier itself, the compilers and the hardware in the usual way, but not the search, its instance solver or its bookkeeping. We tested each component. The program `lb2cr` agrees with an exhaustive search over networks on random instances with 4 to 8 channels, it never answered infeasible in the check at exact values of Section 11, and its feasible answers were turned into networks whose runs we simulated. Prefixes of Juillé's network were never infeasible at budget $45 - k$. A separate exact solver agrees with a sample of the 7-channel table, and part of the 9-channel table was computed twice without conflict. The supplement [17] contains the programs and logs of these tests, except for the repeated computation of part of the 9-channel

table, which it describes. The two proofs in Lean (Appendix A) check the records of the outputs used, with the claims that these records use, and rely on none of these three components. The native proof relies, besides Lean's kernel, on the Lean compiler and its interpreter, the C compiler and the runtime for its computations. The kernel-checked proof relies on Lean's kernel, on the definitions of networks and sorting, which use only Lean's core library, and on the three standard axioms; it does not rely on Lean's compiler, on the data or on the programs that produced them. Every module, Mathlib's included, was checked by Lean's kernel when it was built, and we ran no independent replay of the declarations of this proof.

14. Consequences for larger networks

We now combine Theorems 1.1 and 1.2.

Proof of Corollary 1.3. For $N \geq 3$, $S(N) \geq S(N-1) + \lceil \log_2 N \rceil$ by the one-value bound [22] (Remark 3.2), and $S(N) \geq S(N-2) + P(2,N)$ by Lemma 2.3. Let

$$\Lambda(N) = \max\bigl(\Lambda(N-1) + \lceil \log_2 N \rceil,\ \Lambda(N-2) + \lceil \log_2 F(N) \rceil\bigr), \qquad \Lambda(11) = 35,\ \Lambda(12) = 39,$$

and let Λ' satisfy the same recursion with $\Lambda'(12) = 39$ and $\Lambda'(13) = 45$. Since $S(12) = 39$ [8] and $S(13) = 45$ (Theorem 1.1), Theorem 1.2 and induction give $S(N) \geq \Lambda'(N)$ for $13 \leq N \leq 32$. Theorem 1.1 enters only through the initial value $\Lambda'(13) = 45$. Since $\Lambda(13) = 44$, we have $\Lambda'(14) = \max(49, 48) = 49 = \Lambda(14) + 1$, and if $\Lambda' = \Lambda + 1$ at $N-1$ and $N-2$, then also at N. Hence $\Lambda'(N) = \Lambda(N) + 1$ for $13 \leq N \leq 32$. The same induction from $S(11) = 35$ and $S(12) = 39$ [8] gives $S(N) \geq \Lambda(N)$, the tabulated bounds of Section 1, with Theorem 1.2 in place of Van Voorhis's inequality and without Theorem 1.1. □

N	13	14	15	16	17	18	19	20	21	22
$\Lambda(N)$	44	48	53	57	63	68	73	78	84	89
$\Lambda(N)+1$	45	49	54	58	64	69	74	79	85	90
N	23	24	25	26	27	28	29	30	31	32
$\Lambda(N)$	95	100	106	111	117	123	129	135	141	147
$\Lambda(N)+1$	96	101	107	112	118	124	130	136	142	148

Table 4. The values $\Lambda(N)$ and the lower bounds $S(N) \geq \Lambda(N) + 1$ of Corollary 1.3.

Table 4 lists the bounds. Each step of the recursion takes the larger of the two bounds, so $\Lambda(N) + 1$ is the strongest lower bound on $S(N)$ that the one-value bound and Lemma 2.3 give from $S(11)$, $S(12)$, $S(13)$ and Theorem 1.2. The bounds $S(14) \geq 50$ and $S(16) \geq 59$ of [21], which build on Theorem 1.1, are stronger for $N = 14$ and $N = 16$.

Appendix A. Proofs in Lean

We formalized both bounds of Theorem 1.1 in Lean 4 [13] with Mathlib [14]. Two proofs of the lower bound check the certificate of Section 11. In the first, the large computations run as compiled code (Appendix A.3); in the second, Lean's kernel checks every computation itself (Appendix A.4). A short proof checks the upper bound in the kernel (Appendix A.5). Two further proofs, in the supplement [18], check the bound $P(2,13) \geq 9$ of Theorem 1.2 (Appendix A.6). All of them use the same definitions of networks and of sorting, and the bound $S(11) \geq 35$ enters each proof of a lower bound through one of the two checkers of Harder's certificates (Appendix A.2).

A.1. **The statements.** The definitions behind the statements use only Lean's core library. A comparator on n channels is a pair $i < j$ that puts the minimum on channel i and the maximum on channel j, which is the standard comparator (j, i) of Section 2.1. A network is a list of comparators, and a network sorts if its output is nondecreasing on every input vector of natural numbers:

```
structure Comparator (n : Nat) where
  i : Fin n
  j : Fin n
  lt : i < j

def Comparator.apply (c : Comparator n) (v : Fin n → Nat) : Fin n → Nat :=
  fun k => if k = c.i then min (v c.i) (v c.j)
    else if k = c.j then max (v c.i) (v c.j) else v k

abbrev Network (n : Nat) := List (Comparator n)

def Network.eval (net : Network n) (v : Fin n → Nat) : Fin n → Nat :=
  net.foldl (fun w c => c.apply w) v

def Network.Sorts (net : Network n) : Prop :=
  ∀ v : Fin n → Nat, ∀ a b : Fin n, a ≤ b → net.eval v a ≤ net.eval v b

theorem S13_ge_45 : ∀ net : Network 13, net.Sorts → 45 ≤ net.length

theorem S13_le_45 : ∃ net : Network 13, net.Sorts ∧ net.length = 45
```

Both theorems are in the namespace `Sortnet`. All the Lean projects of the three records [17, 19, 18] contain identical copies of the file of these definitions, `S13/Defs.lean`, and use the same versions of Lean and Mathlib, so `Network` and `Network.Sorts` mean the same in every statement. Together, therefore, the two theorems state $S(13) = 45$ for these definitions. No single Lean theorem states $S(13) = 45$; it would need the proof of Appendix A.5 and one of the large builds of Appendices A.3 and A.4 in one project. The native proof, the proof of the upper bound and the projects of [18] also prove the 0-1 principle (`Network.sorts_iff_sorts01`), over the same definitions, so the theorems hold equally with sorting defined on inputs in $\{0, 1\}$.

The main theorems of this appendix (`S13_ge_45` in both proofs, `S13_le_45`, `juille_sorts`, `juille_length` and `S13_ge_44` in both proofs) depend on no axiom besides the standard axioms `propext`, `Classical.choice` and `Quot.sound`, except that the native proofs add one axiom for each computation that they run as compiled code. In the kernel-checked proof of $P(2, 13) \geq 9$, every theorem depends only on the standard axioms.

A.2. **Harder's certificates in Lean.** The proofs follow the proof system of Harder's certificates [8]. A network *works* on a set V of vectors in $\{0, 1\}^n$ if any two members of V whose outputs have the same weight have the same output; a sorting network works on every set, and its comparators may here be any pairs of channels. A number b is a lower bound for a set if every network that works on the set has at least b comparators. Each step of the certificate claims a lower bound in this sense for its set V. These definitions and the rules below are those of Harder's verified checker [9], where the notion of a lower bound for a set is the predicate `pls_bound` of the file `checker/verified/Sorting_Network.thy` and the rules are Isabelle/HOL lemmas. The rules that the steps use are proved directly in Lean, not translated from the Isabelle/HOL proofs. These rules are: a claim for a set gives the same claim for every larger set, for every relabeling of the channels and for the set with all values inverted; if the empty network does not work on

V, and every comparator maps V to V, to the image of V under the transposition of its two channels, or to a set that needs at least $b-1$ comparators, then V needs b (the *successor rule*); and, for the channels i at which V contains the vector with a single 1 (or a single 0), bounds b_i for the vectors of V with that value at i show that V needs $\lceil \log_2 \sum_i 2^{b_i} \rceil$ comparators. The proof of the last rule is a potential argument: one comparator reduces $\sum_i 2^{b_i}$ by at most a factor of two. A checking function verifies each step of the certificate from earlier steps, and a proof in Lean by induction over the steps shows that a successful check makes every claim true. The certificate is Harder's published file [7], with 12,659,079 steps. Its last step claims the bound 35 for all vectors of width 11, which gives $S(11) \geq 35$ for sorting networks in the form above.

There are two checkers, and the soundness of each is proved in Lean as just described. The *kernel checker* represents a set of vectors of width w by a number whose bit x is set if and only if the vector with code x is in the set. The image under a comparator, a transposition of two channels, the inversion of all vectors and the deletion of a channel are then a few operations on such numbers. We prove each of these operations correct in Lean bit by bit and lift the results to sets of vectors. An untrusted program re-encodes Harder's certificate file, checks every step again, and writes packed data (claims and steps) and the Lean modules that read them. In a module of steps, each theorem states that a range of steps passes the check and is proved by `Eq.refl true`, so that the kernel evaluates the check itself. A tree of theorems then combines the ranges, and finally the claim of the last step is checked by `decide +kernel`.

The *native checker* is a checking function of about 200 lines. An untrusted program re-encodes the certificate as text, read by Lean at build time as string constants, and Lean compiles the check and runs it (the mechanism `native_decide`, described in Appendix A.3).

A.3. **The native proof of the lower bound.** In this proof we check the certificate with our own search over configurations and our own checks of the records, and we prove in Lean that the instances this search rejects are infeasible. The Lean project is part of the supplement [17]; its main theorem is `Sortnet.S13_ge_45` (Appendix A.1). The proof combines seven results, proved in Lean, some of them with the help of computations.

(i) Theorem 7.10 for formal instances on channels `Fin n` (theorem `feasible_valid`): every feasible instance whose list of scenarios is nonempty and has $x \neq y$ in each scenario has a valid configuration. The formal configuration also satisfies the reachability condition of Section 8.1, from which `Comp.solve` derives (N4) and the part of (N3) that marks nodes `dead`, in place of the deletion argument of Lemma 8.2.

(ii) A search over configurations, `Comp.solve`, which performs the search of Section 8.5 without the tests of Lemmas 8.3 and 8.5 to 8.8 and without the conditions (N5) and (N6), and in step (6) checks only (V2) and (V3); leaving out a test can only make the search accept more instances. It accepts every instance on at most 13 channels that has $x \neq y$ in each scenario and a valid configuration (theorem `solve_complete`), so with (i) an instance on at most 13 channels that it rejects is infeasible. Kraft sums are exact integers in units of 2^{-56}, with weight 0 for a depth above 56, and all other arithmetic is exact.

(iii) The weighted two-value bound for the sets of the search: if the instance of a state for one polarity, with weights that are lower bounds for the cores of its scenarios, is infeasible with budget L, then every network that works on the state has at least $L+1$ comparators.

(iv) The trivial bound (Section 9), from $S(11) \geq 35$, given by a computation of the native checker of Harder's certificate (Appendix A.2).

(v) The induction over the layers: if the empty network works on no listed state, and for each listed state X of depth d and each standard comparator c the set $c(X)$ is X, or the image of X under the exchange of the two channels of c, or similar to a listed state of

depth at most $d+1$, or needs at least $44-d$ comparators, then every sorting network on 13 channels has at least 45 comparators.

(vi) The bounds on cores: a claim `K L` holds if `Comp.solve` rejects its instance on 11 channels and each weight is a lower bound for its core on 9 channels, which comes from route (B) of Section 9, by a rejected instance on 9 channels whose weights are proved bounds on 7 channels, or from the table of values on 9 channels.

(vii) Every value in the tables of values on 9 and on 7 channels, as a lower bound, in computations by checkers whose soundness is proved in Lean.

An untrusted program converts the released certificate files into data that the proof reads at build time. The checking functions recompute from the prefixes the states, the useful comparators, the children, the scenarios, the cores and the trivial bounds, and the soundness theorems hold for any data. The records that prove the entries of (vii) form a separate file of the supplement, found with programs that are not part of the release, and the proof checks every one of them. The tactic `native_decide` compiles each check, runs it and records its result as an axiom; apart from overrides in Lean's core and standard libraries, the compiled code comes from the definitions about which the soundness theorems are proved.

A.4. **The kernel-checked proof of the lower bound.** The proof of this section leaves every computation to Lean's kernel. It follows the same certificate, but its checking functions and data are written so that the kernel can evaluate them. Its theorem has the same name and statement as in the native proof, and the final module checks that its type is the same Lean expression. Each check is a Boolean function applied to data, stated as `f a = true` (or `f a = false` for a refutation) and proved by `Eq.refl` or `decide +kernel`, so the kernel accepts it only after it has evaluated `f a` itself; sets of 0-1 vectors are natural numbers read as bitmaps, and the soundness theorems of the checking functions hold for any data. The proof has five parts: (a) $S(11) \geq 35$, from a check of each step of Harder's certificate by the kernel checker (Appendix A.2); (b) and (c) two tables of values on 9 channels, written as numerals, whose entries are proved to be lower bounds from steps of Harder's certificate, from other entries or from certificates of their own; (d) the claims `K L`, whose instances on 11 channels are refuted; and (e) the induction over the layers, whose cuts use instances on 13 channels, each refuted in its own declaration. The instances are refuted by a search over configurations written for the kernel, with a proof that every instance it rejects is infeasible; for the instances on 11 channels it also uses Lemmas 8.5 and 8.7, which are proved in Lean. We also prove Lemmas 8.3, 8.6 and 8.8 in Lean, although the search does not use them. The Lean project, its data and the record of its build are archived on Zenodo [19].

A.5. **The upper bound.** We prove the theorem `Sortnet.S13_le_45` with the network `Sortnet.juille`, the list of the 45 comparators of Juillé's network [11] in 10 layers (Table 5). The theorem `juille_length` states that it has 45 comparators, and `juille_sorts` that it sorts. The proof is part of [19], as the archive `lean_s13_upper.tar.gz`.

By the 0-1 principle, proved in the formalization, a network sorts if it sorts the $2^{13} = 8{,}192$ inputs with entries in $\{0, 1\}$. These inputs run through the network together. Channel k carries one number with 8,192 bits, and bit x of that number is the value of channel k on input number x; at the start it is bit k of x. On the values 0 and 1 the minimum is the bitwise AND and the maximum is the bitwise OR, so a comparator replaces the numbers a and b on its channels by a AND b and a OR b. The kernel runs the 45 comparators on these 13 numbers and checks that the results are nested: for channels $a \leq b$, every bit set in the number on channel a is also set in the number on channel b. So every 0-1 input leaves the network sorted. The check is proved by `decide +kernel`, with no compiled code. Lemmas prove that the bitwise computation agrees

layer	comparators
1	(12, 0) (10, 1) (9, 2) (7, 3) (11, 5) (8, 6)
2	(6, 1) (3, 2) (11, 4) (9, 7) (10, 8)
3	(4, 0) (2, 1) (6, 3) (8, 7) (10, 9) (12, 11)
4	(6, 4) (9, 5) (11, 8) (12, 10)
5	(5, 0) (8, 3) (7, 4) (11, 6) (10, 9)
6	(1, 0) (5, 2) (9, 6) (8, 7) (11, 10)
7	(3, 1) (4, 2) (6, 5) (10, 9)
8	(2, 1) (4, 3) (7, 5) (8, 6)
9	(3, 2) (5, 4) (7, 6) (9, 8)
10	(4, 3) (6, 5)

TABLE 5. Juillé's network on 13 channels, as standard comparators (Section 2.1), applied layer by layer and from left to right within a layer.

with `Network.eval` at every bit position, that the starting numbers have the stated bits, and that every 0-1 input vector is one of the 8,192 bit patterns.

The modules of this proof also pass a replay by Lean's kernel, and an export of the three theorems with their dependencies passes two independent implementations of the kernel.

A.6. **The bound $P(2, 13) \geq 9$.** The supplement [18] contains two Lean proofs of $P(2, 13) \geq 9$, one checked by the kernel and one that uses native code; Section 7 of [20] describes them. Both define runs, $W(x, y)$, top-two correctness and $P(2, n)$ as in Section 2, and prove Lemma 2.3 for sorting networks as defined in Lean. They work with the states of method A (Section 4.6): a state records, for each pair of channels, the largest number of comparators met so far by a run with the largest value on the first channel and the second largest on the second, or nothing if no run is there, and a network with $p_2 \leq L$ leads from the start state to a *goal*, a state with a single entry, which is at most L. A proof in Lean shows that no state of a set H of states reaches a goal if H contains no goal and, for each state of H and each comparator with lo $<$ hi (which suffices by symmetry), the next state is the same state, is discarded by the Kraft prune of method A, or is a relabeling of a state of H; the proofs check this for $L = 8$ and a set H that contains the start state. With $S(11) \geq 35$ (Appendix A.2) they derive the theorem `Sortnet.S13_ge_44`; Theorem 1.1 improves the bound to 45. In the kernel-checked proof Lean's kernel evaluates the check of H itself; the native proof runs it as compiled code and also proves `p2_values`, which states $P(2, N) = \lceil \log_2 F(N) \rceil$ for $3 \leq N \leq 13$; its upper bounds come from explicit networks checked by Lean's kernel, and its lower bounds from compiled checks.

DATA AND CODE AVAILABILITY

The certificate, the verifier with `lb2cr`, the two tables of exact values, the logs of the verification, the specification of the certificate format, the search program and the native Lean project with its data are archived on Zenodo as [17] (doi:10.5281/zenodo.24299104). The search program and the program that computed the tables of exact values are modifications of Harder's program sortnetopt [9] at commit `0b5d09c`. The sortnetopt repository has no licence, so the supplement [17] gives our changes as two patches against commit `0b5d09c`, and the supplement's build instructions fetch the original code. In the patches only the added lines are ours; the context lines and the removed lines are Harder's code and are not covered by our licences. We have asked Jannis Harder for permission to distribute these patches; the request was pending when this version was published. The kernel-checked proof of the lower bound, with its data and the record of its build, and that of the upper bound are archived on Zenodo as [19] (doi:10.5281/zenodo.23603571).

The programs and logs of the computation of $P(2, N)$, the independent checks and the Lean proofs of $P(2, 13) \geq 9$ are archived on Zenodo as [18] (doi:10.5281/zenodo.23498529); they are the contents of the repository [16] at commit `be14346d67fd89b14f7e036ea03eae8d561b81ef`, whose README files number results, sections and tables as in [20], not as in the present paper, and call the basic, refined and extended searches `abs5`, `abs9` and `abs9f`. Part of the logs of the 17 shapes searched with the symmetry test were written by four earlier revisions of the program used for them (`abs9g` in [18]); they give the same output as the released program on 53 small runs, and on 42 shares at $N = 28$ the released program reproduces the results in the logs. None of these archives contains Harder's certificate file [7]. Its steps, re-encoded as Lean data, are part of the kernel-checked proof of the lower bound $S(13) \geq 45$ and are distributed in [19] under CC BY 4.0 with the attribution: Jannis Harder, CC BY 4.0, doi:10.5281/zenodo.4108365. The build instructions of the native proof and of the proofs in [18] fetch the certificate and give its SHA-256 digest, which the kernel-checked proof also records. Our own code is under the Apache License 2.0, and our text (doi:10.5281/zenodo.24299103) and data are under CC BY 4.0.

Acknowledgements

We thank Jannis Harder for publishing sortnetopt, its formally verified checker and the certificate for $S(11) \geq 35$; our search program and the program for the exact-value tables are built on sortnetopt [9], and the search uses that bound [7, 8].

Changes in version 5

This version corrects version 4 (doi:10.5281/zenodo.23731570) as follows; the results and their proofs are unchanged. (V5-1) The abstract credits Harder's method; Section 1 credits the method, partial sorting networks and the Huffman bound where it introduces the two-value argument. (V5-2) Twelve pronouns referring to Jannis Harder are replaced by the name or removed by rewording. (V5-3) Section 10.1 and the data statement describe the reuse of Harder's program sortnetopt: they state which programs derive from it, give the status of the request for permission to distribute the patches, and note that the context lines and the removed lines in the patches are not under our licences; the data statement now applies our licences only to our own code, text and data. (V5-4) The data statement gives the licence notice for the data re-encoded from Harder's certificate. (V5-5) Section 6 relates the weighted bound to Harder's Huffman bound. (V5-6) Sections 1 and 10 credit the generate-and-prune approach of [2]; Section 10 also credits Harder's algorithm [8, Section 5] for discarding sets by lower bounds and [8, Corollary 6] for the symmetries. (V5-7) The term pass-through, the red and blue leads and the closed form of F are credited to [15]. (V5-8) Acknowledgements are added. (V5-9) Section 1 cites the earlier certified proofs [3, 4]. (V5-10) The reference [9] gives the archived release and the file in which the checker lemma is restated. (V5-11) The data statement gives the DOI of this version, and the DOI of the certificate record [17] points to its version 2. (V5-12) Appendix A.2 credits the notion of a network that works on a set, the lower bounds for sets and the rules of the proof system to Harder's verified checker. (V5-13) Section 9 states why the rule for re-stored states has no effect on the tables of exact values.

References

[1] R. C. Bose and R. J. Nelson, *A sorting problem*, J. ACM **9** (1962), no. 2, 282–296, doi:10.1145/321119.321126.
[2] M. Codish, L. Cruz-Filipe, M. Frank, and P. Schneider-Kamp, *Sorting nine inputs requires twenty-five comparisons*, J. Comput. System Sci. **82** (2016), no. 3, 551–563, doi:10.1016/j.jcss.2015.11.014.
[3] L. Cruz-Filipe and P. Schneider-Kamp, *Optimizing a certified proof checker for a large-scale computer-generated proof*, Intelligent Computer Mathematics (CICM 2015), Lecture Notes in Comput. Sci., vol. 9150, Springer, Cham, 2015, pp. 55–70, doi:10.1007/978-3-319-20615-8_4.

[4] L. Cruz-Filipe, K. S. Larsen, and P. Schneider-Kamp, *Formally proving size optimality of sorting networks*, J. Automat. Reason. **59** (2017), 425–454, doi:10.1007/s10817-017-9405-9.
[5] B. Dobbelaere, *Smallest and fastest sorting networks for a given number of inputs*, web page, `https://bertdobbelaere.github.io/sorting_networks.html`, consulted 2026-10-18 (history entry of 2025-04-21).
[6] R. W. Floyd and D. E. Knuth, *The Bose–Nelson sorting problem*, A Survey of Combinatorial Theory (J. N. Srivastava et al., eds.), North-Holland, Amsterdam, 1973, pp. 163–172, doi:10.1016/B978-0-7204-2262-7.50020-X.
[7] J. Harder, *Certificate of the minimal size of 11-channel sorting networks*, Zenodo, 2019, dataset, doi:10.5281/zenodo.4108365.
[8] J. Harder, *An answer to the Bose–Nelson sorting problem for 11 and 12 channels*, arXiv:2012.04400v3 [cs.DS], 2022.
[9] J. Harder, *Lower size bounds for sorting networks* (sortnetopt), source code, 2020, `https://github.com/jix/sortnetopt`, commit `0b5d09c`. Its release v1, archived as doi:10.5281/zenodo.4139152, predates that commit. Isabelle/HOL files: the checker lemma `check_proof_get_bound_spec` in `checker/verified/Checker.thy`, restated in `Checker_Codegen.thy`; the size bound `lower_size_bound` in `Sorting_Network_Bound.thy`.
[10] Q. Huangfu and J. A. J. Hall, *Parallelizing the dual revised simplex method*, Math. Program. Comput. **10** (2018), no. 1, 119–142, doi:10.1007/s12532-017-0130-5.
[11] H. Juillé, *Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces*, Proceedings of the Sixth International Conference on Genetic Algorithms (L. J. Eshelman, ed.), Morgan Kaufmann, San Francisco, 1995, pp. 351–358.
[12] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2nd ed., Addison-Wesley, Reading, MA, 1998.
[13] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, Automated Deduction – CADE 28, Lecture Notes in Comput. Sci., vol. 12699, Springer, 2021, pp. 625–635, doi:10.1007/978-3-030-79876-5_37.
[14] The mathlib Community, *The Lean mathematical library*, Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020), ACM, 2020, pp. 367–381, doi:10.1145/3372885.3373824.
[15] Mericanii, *Twenty tabulated lower bounds for sorting networks rest on an invalid proof: an audit of Van Voorhis (1972), and a partial repair*, draft, August 2026, `https://mericanii.com/work/s13/paper-correction.pdf`, consulted 2026-10-18.
[16] T. S. Tomar, *sortnet13-two-value-v5*, GitHub repository, `https://github.com/tejasvi/sortnet13-two-value-v5`, commit `be14346d67fd89b14f7e036ea03eae8d561b81ef`.
[17] T. S. Tomar, *Certificate and verifier for "Sorting networks on 13 inputs need 45 comparators"*, Zenodo, 2026, doi:10.5281/zenodo.24299104.
[18] T. S. Tomar, *Programs and logs for "Sorting networks on 13 inputs need at least 44 comparators"*, Zenodo, 2026, doi:10.5281/zenodo.23498529.
[19] T. S. Tomar, *A kernel-checked Lean proof that sorting networks on 13 inputs need 45 comparators*, Zenodo, 2026, doi:10.5281/zenodo.23603571.
[20] T. S. Tomar, *Sorting networks on 13 inputs need at least 44 comparators*, Zenodo, 2026, doi:10.5281/zenodo.23498528.
[21] T. S. Tomar, *Sorting networks on 14 inputs need 50 comparators*, Zenodo, 2026, doi:10.5281/zenodo.23622888.
[22] D. C. Van Voorhis, *An improved lower bound for sorting networks*, IEEE Trans. Comput. **C-21** (1972), no. 6, 612–613, doi:10.1109/TC.1972.5009021.
[23] D. C. Van Voorhis, *Toward a lower bound for sorting networks*, Complexity of Computer Computations (R. E. Miller and J. W. Thatcher, eds.), Plenum Press, New York, 1972, pp. 119–129, doi:10.1007/978-1-4684-2001-2_12.

ORCID: `https://orcid.org/0000-0003-4668-3639`