hexagon:2611.00391v1

SORTING NETWORKS ON 13 INPUTS NEED AT LEAST 44 COMPARATORS

TEJASVI SINGH TOMAR

Abstract. Let $S(N)$ be the least number of comparators in a sorting network on N inputs. We prove $S(13) \geq 44$, one more than the previously best proved lower bound. 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; the lower bound comes from a finite search based on the structure of the paths of these two values. Starting from Harder's $S(11) = 35$ and $S(12) = 39$, these values of $P(2, N)$, the two-value bound $S(N) \geq S(N-2) + P(2, N)$ and the one-value bound $S(N) \geq S(N-1) + \lceil \log_2 N \rceil$ give the twenty lower bounds on $S(N)$ for $13 \leq N \leq 32$ in Dobbelaere's table; these bounds rested on Van Voorhis's inequality $P(2, N) \geq \lceil \log_2 F(N) \rceil$, whose published proof is invalid. The bound $S(13) \geq 44$ is also proved in Lean 4, with every step checked by Lean's kernel.

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 $N \leq 8$ [4], Codish, Cruz-Filipe, Frank and Schneider-Kamp proved $S(9) = 25$ and $S(10) = 29$ [2], and Harder proved $S(11) = 35$ and $S(12) = 39$ [6]. Harder's bound $S(11) \geq 35$ is backed by a certificate, which is checked by a program verified in Isabelle/HOL [5]. For 13 inputs the smallest known sorting network, found by Juillé [8], has 45 comparators. Van Voorhis's one-value bound $S(N) \geq S(N-1) + \lceil \log_2 N \rceil$ [13] gives $S(13) \geq 43$, and this has been the best proved lower bound.

In a chapter published the same year [14], 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

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

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 2.3). With $F(13) = 392$ and $S(11) = 35$ this gives $S(13) \geq 35 + 9 = 44$. Dobbelaere's table of sorting networks [3] has listed 44 as the lower bound for 13 inputs, together with lower bounds for $14 \leq N \leq 32$, crediting them to a suggestion of Jelmer Firet and to principles in Van Voorhis's chapter. Mericanii [10], however, showed that the chapter's 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

Date: 28 October 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.

paths merge. This fails when a path of the maximum passes a comparator whose other input never carries the maximum; we call such a comparator a *pass-through*. Pass-throughs occur already in a size-optimal network on 3 inputs (see the example after Lemma 4.2). Mericanii proves the inequality for two classes of networks and leaves it open in general. The same work derives the tabulated lower bounds for $13 \le N \le 32$ from (1), the one-value bound and $P(2, N) \ge \lceil \log_2 F(N) \rceil$, 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.

We use neither Van Voorhis's inequality $P(2, N) \ge \lceil \log_2 F(N) \rceil$ nor its proof in [14]; instead we compute $P(2, N)$ for $3 \le N \le 32$.

Theorem 1.1. *$S(13) \ge 44$.*

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

Corollary 1.3. *For $13 \le N \le 32$, $S(N)$ is at least the value $\Lambda(N)$ listed in Section 8.*

The proofs of these three results include computations. The sequel [15] proves $S(13) = 45$, and so raises each bound of Corollary 1.3 by one; it extends the structure theory of Section 3 to a weighted form of (1), which it applies to suffixes of a network.

The upper bound in Theorem 1.2 comes from an explicit network for each N (Proposition 2.5). The lower bound rests on a computation, and two observations keep it small. First, after the second maximum meets the maximum, it travels through comparators that the maximum never visits, and it can re-enter the tree of paths of the maximum only through pass-throughs. We delete each pass-through whose other input never carries the second maximum, and each comparator off the tree with an input that never carries it (Lemma 3.5). Then the numbers of comparators met by the two values, for all $N(N-1)$ choices of $x \ne y$, are 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. For the places that share a re-entry point, and for those that reach the output, Kraft's equality describes which combinations of these numbers of comparators are possible. Second, the paths of the maximum are short in any network in which the two values meet few comparators (Lemma 3.10), which bounds the height of the tree. By Lemma 2.3 and Table 1, searches for $3 \le N \le 13$ and $N = 17$, 21 and 28 suffice for the lower bound (Theorem 5.1); those for $N \le 21$ are small. The search for $N = 28$ is far larger, and for some of its slowest tree shapes we also use the symmetries of the tree and its pass-throughs, so that only some of the equivalent choices of re-entry points and numbers of comparators are searched, but at least one from each class is searched (Lemma 4.4).

Section 2 defines $P(2, N)$ and proves (1), the monotonicity of $P(2, N)$ in N and the upper bound. Section 3 describes networks that deliver the two largest values to fixed outputs, and Section 4 gives the search and the validity of each pruning rule. Section 5 reports the searches and proves Theorem 1.2, Section 6 and Appendix A describe the checks, Section 7 gives two formal proofs of Theorem 1.1 in Lean 4, and Section 8 proves Theorem 1.1 and Corollary 1.3.

2. Networks, two-value pruning and an upper bound

2.1. **Comparator networks.** A *comparator network* T on N inputs is a finite acyclic circuit made of N input leads 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 N of them. Given distinct real 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. T is a *sorting network* if its outputs can be listed as $o_1, \dots, o_N$ so that o_i carries the i-th smallest input value for all inputs.

Drawing T on N channels along a topological order of its comparators gives a network in the usual sense in which a comparator may send its maximum to either of its two channels, followed by a permutation of the channels. Renaming the channels removes the permutation, and a sorting network of this kind can be untangled into a standard one with the same number of comparators [9, Exercise 5.3.4–16]. Thus $S(N)$ is the same in both models.

2.2. **Runs and the quantity $P(2, N)$.** Let $N \geq 2$. For distinct inputs x and y, the *run* (x, y) puts the largest value M on x, the second largest value m on y, and smaller 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. Consequently, the leads that carry M and m depend only on (x, y). We write P_x for the set of leads that carry M, which depends only on x, and $W(x, y)$ for the set of comparators with M or m on an input lead.

Definition 2.1. 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 $o_M = o_N$ and $o_m = o_{N-1}$. The next lemma establishes inequality (1), which is equation (2) of [14] (see [10, Section 2]). Its proof extends Harder's argument for one value [6, Lemma 16] to two values.

Lemma 2.2 (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 and o_{N-1}, and $o_1, \dots, o_{N-2}$ 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 2.3 (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 $P(2, N) \leq p_2(T') \leq p_2(T)$. □

Lemma 2.4 (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* $d_1, \dots, d_k \geq 0$ *satisfy* $\sum_i 2^{-d_i} = 1$, *some such tree has leaves at depths* $d_1, \dots, d_k$.
(c) *If* $k \geq 1$ *and integers* $\ell_1, \dots, \ell_k \geq 0$ *satisfy* $\sum_i 2^{-\ell_i} \leq 1$, *some such tree has* k *leaves at depths* $d_i \leq \ell_i$.

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 $D = \max_i d_i \geq 1$. 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). □

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

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

where c ranges over the internal vertices of B and B over the binary trees with N leaves. Van Voorhis introduced the function F [14]. As checked in [10, Section 2], f satisfies the recursion of [14, Theorem 1], and $\min_B f(B)$ reproduces Van Voorhis's table of $F(N)$ for $N \leq 16$. We use the definition above for all N, as [10] does. Table 1 lists $F(N)$ for $N \leq 32$. We computed it by a dynamic program over the pairs (height, f), and the values for $13 \leq N \leq 32$ agree with [10, 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
$\beta(N)$	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
$\beta(N)$	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 $\beta(N) = \lceil \log_2 N \rceil + \lceil \log_2 (N-1) \rceil$ of Corollary 3.11.

Proposition 2.5. *For each* $N \geq 2$ *there is a top-two correct network on* N *inputs with* $2N-3$ *comparators and* $p_2 \leq \lceil \log_2 F(N) \rceil$. *Hence* $P(2, N) \leq \lceil \log_2 F(N) \rceil$.

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 $\mathrm{dep}(x)$ 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 2.4(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 $|W(x, y)| \le K$. The network has $(N - 1) + (N - 2)$ comparators. □

3. The structure of top-two correct networks

3.1. **The tree of maximum paths.** A lead is *red* if it lies in P_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 $n \ge 2$ 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* hi(g) *is red if and only if g has a red input lead.*

(b) Join each red lead $\ell \ne o_M$ to 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 P_x. The maximum leaves each comparator on hi, so no lo output is red. If an input lead of g lies in P_x, the maximum enters g on it and leaves on hi(g), so hi(g) $\in P_x$. Conversely, if hi(g) $\in P_x$, the maximum entered g on an input lead, which lies in P_x.

(b) Let $\ell \in P_x$. If ℓ enters a comparator g, then hi(g) is the next lead of P_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 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 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 hi(g), which has one child, so it lies on an edge of B or above the root of B. □

From now on T is a top-two correct network on $N \ge 3$ inputs and B is its tree. For a vertex v of B (an input or a branch node), seg(v) denotes the edge from v to its parent, or from v to o_M if v is the root, and $k(v)$ denotes the number of pass-throughs on seg(v). For a pass-through q let $u(q)$ be the vertex with q on seg($u(q)$). We call the branch nodes and the pass-throughs the *red comparators*. For a red comparator v, let $A(v)$ be the set of comparators on the path from v to o_M, v included, and let $\bar{v} = v$ if v is a branch node and $\bar{v} = u(v)$ if v is a pass-through. For an input x let $\delta(x)$ be the number of comparators on the path from x to o_M, which are the comparators that M passes when it starts on x. For a branch node c and an input y below c, let pre(y, c) be the number of comparators on the path from y up to c, with c excluded, so that $\delta(x) = |A(c)| + \mathrm{pre}(x, c)$ for each branch node c above 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 2.4(a). Together with the pruning of one path [6, Lemma 16] this gives the one-value bound $S(N) \ge S(N - 1) + \lceil \log_2 N \rceil$ of Van Voorhis [13].

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 P_x. When m enters a comparator g, the lead that carries m is not in P_x, so m leaves on $\mathrm{lo}(g)$ exactly when the other input lead of g lies in P_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 $A(v) \setminus \{v\}$, 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 $\bar{v}$ are incomparable in B, let $w(q, v) = \mathrm{lca}_B(u(q), \bar{v})$, and let $\kappa(q, v)$ be the number of comparators from q up to $w(q, v)$, with $w(q, v)$ excluded:

$$\kappa(q, v) = 1 + \#\{\text{pass-throughs above } q \text{ on } \mathrm{seg}(u(q))\} + \sum_{w'} (1 + k(w')),$$

where w' runs over the branch nodes strictly between $u(q)$ and $w(q, v)$.

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 $\bar{v}$ are incomparable in B, the vertex $w = w(q, v)$ lies in $A(v)$, the value m passes $\kappa(q, v)$ comparators outside $A(v)$ from q up to w, meets M at w and leaves on $\mathrm{lo}(w)$, and $\tau(v) = d(v) + \kappa(q, v) + \tau(w)$.

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 $\bar{v}$, or equal to $\bar{v}$ with q above v, then q would lie in $A(v)$. If $u(q)$ were a proper descendant of $\bar{v}$, or equal to $\bar{v}$ with q below v, then q would come before v in the circuit. Hence $u(q)$ and $\bar{v}$ are incomparable, and w is a proper ancestor of $\bar{v}$, hence $w \in A(v)$. 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 w' strictly between $u(q)$ and w is not an ancestor of $\bar{v}$, so $w' \notin A(v)$. 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 $\mathrm{seg}(w')$, whose pass-throughs are not in $A(v)$ either. At w the value M arrives on the input lead on the side of $\bar{v}$ and m on the other one, so m leaves on $\mathrm{lo}(w)$. The comparators passed from q up to w, with w excluded, are $\kappa(q, v)$ in number and none is in $A(v)$, which gives the formula for $\tau(v)$. □

Each step of this recursion moves to a comparator of $A(v)$ strictly above 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

(N1) the port of each pass-through is relevant, and
(N2) 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 (N1). 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 (N1) or (N2) 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 (N1) 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 (N2). 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 (N2) an irrelevant one enters no blue comparator, and by (N1) 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 $k(v) \geq 0$ of pass-throughs on each edge $\mathrm{seg}(v)$, placed in order along the edge;
(iii) for each *source*, that is, each internal vertex c of B and each pass-through q, either the mark *dead*, allowed for pass-throughs only, or a *root* $\rho \in \{o_m\} \cup \{\text{pass-throughs}\}$ and a *depth* $d \geq 0$;

subject to two conditions:

(K) for each $\rho \in \{o_m\} \cup \{\text{pass-throughs}\}$, the sources assigned to ρ form a nonempty set whose depths satisfy $\sum 2^{-d} = 1$;
(A) the directed graph $G(\Gamma)$ on the red comparators, with an arc from each red comparator to the next comparator above it and an arc from each source v to its root q whenever q is a pass-through, is acyclic.

We call the pair (B, k) of (i) and (ii) the *skeleton* of Γ. The quantities δ, pre, $A(\cdot)$, κ and w are defined from the skeleton as above. For each source v that is not dead, define $\tau_\Gamma(v)$ by the recursion of Lemma 3.4,

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

and put $\tau_\Gamma(v) = \infty$ if the recursion reaches a dead source. 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 $\mathrm{cost}_\Gamma(x, y) \leq L$ for all $x \neq y$.

In the third case of the recursion, $u(q)$ and $\bar{v}$ are incomparable by (A). Otherwise q would lie 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 $v \to q$ 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 $N \geq 3$ and $L \geq 0$.*

(a) If some top-two correct network on N inputs has $p_2 \leq L$, 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 source that is not dead 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* $x \neq y$.

Proof. (a) By Lemma 3.5 we may assume that the network T is reduced. Lemma 3.1 gives B and the numbers $k(v)$. By Lemma 3.6, give each branch node c, and each pass-through q for which $\mathrm{lo}(q)$ is relevant, the root and the depth of its lo output in the forest, and mark the other pass-throughs dead. Condition (K) holds by Lemma 2.4(a). An arc of $G(\Gamma)$ is a directed path in the circuit: from a red comparator up to the next one, or from a source v through $\mathrm{lo}(v)$ and the blue comparators of its tree into the pass-through q. Consequently, a cycle in $G(\Gamma)$ would give a cycle in the circuit, and (A) holds. By Lemmas 3.3, 3.4 and 3.6(d), $|W(x, y)| = \mathrm{cost}_\Gamma(x, y)$ for all $x \neq y$. No recursion reaches a dead source, because the sources it reaches carry m in the run. Hence Γ is feasible at level L. A source v that is not dead carries m in some run (x, y), and that run passes the $d(v)$ blue comparators below $\rho(v)$ after $\mathrm{lo}(v)$, so $d(v) \leq |W(x, y)| \leq L$.

(b) Build the circuit as follows. The red part is the tree B with $k(v)$ comparators on each edge $\mathrm{seg}(v)$, 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 ρ, take a binary tree of blue comparators whose leaves are the lo outputs of the sources assigned to ρ, at the prescribed depths (Lemma 2.4(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 ρ, or the output o_m if $\rho = o_m$. The lo outputs of dead sources 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 (A) 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 source ends at its root by construction. Since Γ is feasible, the recursion from each $\mathrm{lo}(c)$ avoids dead sources 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 at the i-th place on $\mathrm{seg}(h(v))$, counted from below, is the image of the one at the i-th place on $\mathrm{seg}(v)$, and of the assignment that gives the source $h(v)$ the root $h(\rho(v))$ and the depth $d(v)$ if v is not dead, 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 *(the last is defined in Lemma 3.12) 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, $A(\cdot)$, κ and w 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 $w(h(q), h(v)) = h(w(q, v))$. The sources assigned to $h(\rho)$ in $h\Gamma$ are the images of those assigned to ρ 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 (A) holds. Induction along the recursion of Definition 3.7 gives $\tau_{h\Gamma}(h(v)) = \tau_\Gamma(v)$ for every source v that is not dead, and the costs follow. □

The *decorated canonical string* of the subtree at a vertex u is $k(u)\mathtt{L}$ 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 a comparator network on the $N-1$ inputs other than x. In the run (x,y) of T, the value m is never on a lead of P_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 2.4(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 meets 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. *$P(2,N) \geq \beta(N) := \lceil \log_2 N \rceil + \lceil \log_2(N-1) \rceil$ for $N \geq 3$.*

Proof. Some leaf x of B has depth at least $\lceil \log_2 N \rceil$ by Lemma 2.4(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*

$$d(c) + \pi(c) \;\leq\; \tau_\Gamma(c) \;\leq\; \sigma(c,x) := L - \delta(x) - \max_{y \text{ below } c_2} \operatorname{pre}(y,c),$$

where $\pi(c) = \kappa(\rho(c), c)$ if $\rho(c)$ is a pass-through outside $A(c)$, and $\pi(c) = 0$ otherwise.

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

4. The search

Fix $N \geq 3$ and L, and put $H = H_L = L - \lceil \log_2(N-1) \rceil$.

4.1. **The basic search.**

(S1) Enumerate the binary trees B with N leaves and height at most H, 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 t lexicographically.

(S2) For each B, enumerate the numbers $k(v) \in \{0, \ldots, H\}$ with $\delta(x) \leq H$ 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 sources: 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 $v \to q$ to a pass-through q is added only if the graph of the arcs present so far has no path from q to v;
  (c) each internal vertex c satisfies $d(c) + \pi(c) \le \sigma(c,x)$ 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 source 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 and that $\delta(x) \le H$ for each x. 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 $h = \phi h_0$ for some automorphism ϕ of B'. By the same lemma, $h\Gamma$ is feasible at level L, has depths at most L and satisfies $\delta(x) \le H$. 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, (A) 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 source 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 $v \to q$.

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 ρ that c may join under (b), let $\mu(c,\rho)$ be $\min_x \sigma(c,x)$ minus the value that $\pi(c)$ would take if $\rho(c) = \rho$, 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 $\mu(c,\rho) < 0$ for each open root ρ, or if

$$\sum_{c\in U} 2^{-\max_\rho \mu(c,\rho)} \;>\; \sum_{\rho \text{ open}} \Bigl(1 - \sum_{v \text{ assigned to } \rho} 2^{-d(v)}\Bigr).$$

(P3) *Partial costs.* During (S4), define $\tau^-(v)$ by the recursion of Definition 3.7 with $\tau^-(v) = 0$ for sources that are not yet assigned or are dead. Prune if $\delta(x) + \mathrm{pre}(y,c) + \tau^-(c) > L$ for some $x \ne y$ 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 $w(q, v)$, where v is the comparator of the previous meeting. Here $w(q, v) = c$, 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 $(0,1),(0,2),(1,2)$ on 3 inputs, written as usual on channels, with (i, j) sending the maximum to channel j [10, Proposition 3]. Its branch nodes are $c_1 = (0,1)$ and $c_2 = (1,2)$, and $(0,2)$ is a pass-through on the edge of input 2. For $x = 0$ the value m leaves c_1, enters $(0,2)$ 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 [10] 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 $\mu(c, \rho)$. It therefore adds at least $2^{-\max_\rho \mu(c,\rho)}$ 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 sources do not change, and the other terms are nonnegative or infinite. Induction along the recursion gives $\tau^-(v) \le \tau_\Gamma(v)$ for every source 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 sources are visited in a fixed order $s_1, \dots, s_n$. We order the possible values of a source 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 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 $\delta(x) \le H$ 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 $k(v)$ 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 $k(v)$, or a fixed depth of the depth-first search (the number of sources 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 sources, 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 four programs compute Kraft sums in double precision. Their terms are powers of two with exponents between $-L$ and 0, so the sums are exact.

5. Results of the searches

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 5.1. *Let $N \in \{3, 4, \ldots, 13\} \cup \{17, 21, 28\}$ and $L = \lceil \log_2 F(N) \rceil - 1$. The search of Section 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 checks that every one of the 62,641 shapes is covered and that the shares of each divided search are all present. □

N	L	H	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. At $N = 28$ the counts are totals over all shapes, recomputed by the script of the proof of Theorem 5.1: each skeleton is counted once, the nodes above a split of the depth-first search are counted once for each share, and for the 17 shapes searched with the symmetry test the counts are those after that test.

Note that for $N \in \{3, 4, 6, 10, 11\}$ no tree has height at most H, as Corollary 3.11 already shows. At $N = 13$ and $L = 8$, four tree shapes have height at most $H = 4$, 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 extended search examined the shapes 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. No part reports a feasible configuration. A separate run repeated the search for all other shapes (Table 3), so for these 119 shapes, and in particular for the 17 searched with the symmetry test, the result rests on a single computation, made by programs that were checked as Section 6 and Appendix A describe.

Each search with $N \le 21$ took under five seconds; the run at $N = 28$ took of the order of 10^4 core-hours.

Proof of Theorem 1.2. For $N \in \{3, \dots, 13\} \cup \{17, 21, 28\}$ the lower bound is Theorem 5.1. For each other $N \le 32$, 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 2.3 gives $P(2, N) \ge P(2, N')$. The upper bound is Proposition 2.5. □

6. Verification

Three kinds of check do not use the programs of Section 4. Methods A, B and C, which share no code with the basic search and do not use the model of Section 3, compute $P(2, N)$ again for $N \le 13$; a search over standard networks without the model settles $N = 12$ and $N = 13$ at $L = 8$, although its pruning rules have no written proofs; and the formal proofs of Section 7 prove $P(2, 13) \ge 9$ by method A, and one of them also $P(2, N) = \lceil \log_2 F(N) \rceil$ for $3 \le N \le 13$. The basic search and its model are also tested, for $N \le 13$, against an integer program, against simulation and on networks mapped into the model. For $N = 17$, 21 and 28 the checks use the model: the searches with (P1)–(P3) agree with each other and with the basic search where their runs overlap, and a separate run repeated part of the search at $N = 28$ (Section 5). Appendix A describes the methods and lists every check of this section except the formal proofs, with its scope and outcome, in Table 3.

7. Formal proofs in Lean

We proved Theorem 1.1 twice in Lean 4 [11] with Mathlib [12], including Harder's bound $S(11) \ge 35$. Both formalizations are independent of Sections 3 and 4: their proofs of $P(2, 13) \ge 9$ use method A of Appendix A, their bound $S(11) \ge 35$ comes from checking Harder's certificate [5], and every pruning or proof rule that these two computations use is proved in Lean. The main theorem, `Sortnet.S13_ge_44`, has no hypotheses. In the first formalization [18], Lean's kernel checks every step, including the two computations. Its theorems `S13_ge_44`, `main_theorem` and `S11_ge_35` depend only on the standard axioms `propext`, `Classical.choice` and `Quot.sound`, so it trusts Lean's kernel alone. The second formalization [17] evaluates the two computations in native code and therefore also trusts the Lean compiler, the C compiler and the Lean runtime; Section 7.3 describes both. In both, the programs that produce certificates and data are not trusted: they only supply terms, and Lean checks every claim made about them. The supplement contains the sources, build instructions, logs and tests of both.

7.1. **The statement.** The definitions behind the statement 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, 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_44 : ∀ net : Network 13, net.Sorts → 44 ≤ net.length

theorem main_theorem :
    (∀ net : Network 11, net.Sorts → 35 ≤ net.length) →
      ∀ net : Network 13, net.Sorts → 44 ≤ net.length
```

The theorem `S13_ge_44` is in the namespace `Sortnet`. Both formalizations derive it from the proposition shown as `main_theorem` and from the bound $S(11) \geq 35$ for these networks. The formalizations also prove the 0-1 principle, so the theorems hold equally with sorting defined on inputs in $\{0, 1\}$.

7.2. **What is proved.** The formalizations define runs, $W(x, y)$, top-two correctness and $P(2, n)$ as in Section 2, for networks whose comparators may put the maximum on either channel; the run (x, y) has the value $n + 1$ on input x, n on input y, and z on each other input z. The proof combines three results.

(i) Lemma 2.2 for sorting networks in the form above: for $x \neq y$, deleting the comparators of $W(x, y)$ from a sorting network T on $n + 2$ channels leaves a network that sorts with respect to a fixed order of its outputs, and untangling it [9, Exercise 5.3.4–16] gives a sorting network on n channels with $|T| - |W(x, y)|$ comparators.
(ii) A Boolean computation that implies $P(2, 13) \geq 9$.
(iii) A Boolean computation that implies $S(11) \geq 35$.

By (i) and (ii), $S(11) \geq 35$ implies $S(13) \geq 44$, which is `main_theorem`; with (iii) this gives `S13_ge_44`. The two formalizations share the definitions, the proof of (i), and the reductions of (ii) and (iii) to conditions on states and on the steps of a certificate. They differ in the checking functions that verify these conditions and in how the checks are evaluated.

For (ii), the computation works with the states of method A, which compress the runs. A state assigns to each pair of channels (p, q), $p \neq q$, the largest number of comparators met so far by a run that has the largest value on p and the second largest on q, or nothing if no run is there. A network with $p_2 \leq L$ is a sequence of comparators that leads from the start state (cost 0 at every pair) to a *goal*, a state with a single entry that is at most L. We prove three facts about the step from a state f to the state f' after a comparator:

- The potential $\Phi(f) = \sum_p R(\sum_{q \neq p} 2^{f(p,q)})$, where $R(s)$ is the least power of two that is at least s ($R(0) = 0$) and absent entries contribute 0, satisfies $\Phi(f') \geq \Phi(f)$. At a goal $\Phi \leq 2^L$, so a state with $\Phi > 2^L$ cannot reach a goal. This is the Kraft prune of method A.
- Relabeling the channels by a permutation maps reachable goals to reachable goals with the same number of comparators.
- The two orientations of a comparator give states that differ by the transposition of its channels, so comparators with lo $<$ hi suffice.

From these facts we prove in Lean: if a set H of states contains no goal and, for every state in H and every comparator with lo $<$ hi, the next state is the same state, or has $\Phi > 2^L$, or has a relabeling in H, then no state in H reaches a goal. It remains to exhibit such a set H for $N = 13$ and $L = 8$ that contains the start state, and to check the conditions; each formalization shows that a successful check implies $P(2, 13) \geq 9$.

For (iii), the formalizations follow the proof system of Harder's certificates [6]. A network *works* on a set V of vectors in $\{0,1\}^n$ if outputs of equal weight are equal; a sorting network works on every set, and its comparators may here be any pairs of channels. Each step of a certificate claims that every network that works on its set has at least a given number of comparators. In each formalization the rules that the steps use, those of Harder's proof system, are proved in Lean, 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 [5], 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.

7.3. **The two formalizations.** In the first formalization, the data for (ii) are a list H of states, the start state first, with a permutation of the channels and the index of a state of H for each state and each comparator with lo $<$ hi whose next state is neither unchanged nor pruned by the potential Φ, so that the indexed state equals the relabeled next state; for (iii) they are Harder's file, converted into data that Lean modules read. Checking functions written for Lean's kernel verify these conditions and that no state of H is a goal; sets of vectors are stored as numbers read as bitmaps, with operations proved correct in Lean, and the kernel evaluates the checks itself. In the second, an untrusted search proposes for (ii) a set H of states, each relabeled by colour refinement, and Harder's certificate is re-encoded as text that Lean reads at build time; Lean compiles the two checks, runs them, and records each outcome as an axiom. Besides the standard axioms, `#print axioms Sortnet.S13_ge_44` lists exactly these two. The theorem `p2_values` of the second formalization 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 native checks of the kind in (ii), one for each N.

8. Consequences for sorting networks

Proof of Theorem 1.1. By Lemma 2.2, $S(13) \geq S(11) + P(2,13)$. Harder proved $S(11) = 35$ [6], and Theorem 1.2 gives $P(2,13) = 9$. Therefore $S(13) \geq 44$. □

Proof of Corollary 1.3. For $N \geq 3$ we have $S(N) \geq S(N-1) + \lceil \log_2 N \rceil$ [13] (see Remark 3.2) and $S(N) \geq S(N-2) + P(2,N)$ by Lemma 2.2. Starting from $S(11) = 35$ and $S(12) = 39$ [6], put

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

By Theorem 1.2 and induction, $S(N) \geq \Lambda(N)$ for $13 \leq N \leq 32$. The values are

N	13	14	15	16	17	18	19	20	21	22
$\Lambda(N)$	44	48	53	57	63	68	73	78	84	89
N	23	24	25	26	27	28	29	30	31	32
$\Lambda(N)$	95	100	106	111	117	123	129	135	141	147

These are the lower bounds in Dobbelaere's table [3]. □

The method of this paper cannot decide between 44 and the 45 comparators of Juillé's network [8], because $P(2,13) = 9$ is exact: the two-value bound with $S(11) = 35$ gives no more than 44.

Appendix A. The checks of the computation

In the first group of Table 3, method A searches comparator networks directly, with exact compression of states, rejection of isomorphic states and a pruning rule based on a Kraft inequality over the runs, and methods B and C search a relaxed tree model, with and without a height bound. Notes in the supplement [19] prove the Kraft inequality and the structure theorem of the relaxed model. The search over standard networks in the third group prunes with two lower bounds of Huffman type and a reduction by symmetry.

check	outcome
1. Programs that use neither the search nor its model	
method A (networks, Kraft prune), $3 \le N \le 13$ at $L = \lceil \log_2 F(N) \rceil - 1$, and witnesses at $L = \lceil \log_2 F(N) \rceil$ for $N \le 12$	$P(2, N)$ as in Theorem 1.2, witnesses simulated
method A without the Kraft prune, $N = 9$, $L = 7$	infeasible, 4,619,670 states
methods B and C (relaxed tree model), $3 \le N \le 13$	$P(2, N)$ as in Theorem 1.2
tests of the pruning rules of method A on all 15,113,524 states reachable with costs at most L, for $N = 8$ and $L = 6, 7, 8$	no state from which a goal is reachable is pruned
tests of the relaxed tree model on 3,420 random top-two correct networks with $3 \le N \le 8$	no failure
2. Tests of the basic search and its model	
an integer program (HiGHS [7]) for each skeleton, 15 pairs (N, L) with $N \le 13$	the same skeletons feasible as in the basic search; at $(13, 8)$ none of 508 skeletons on ordered trees
4,464,896 top-two correct networks with $4 \le N \le 11$, mapped into the model	costs equal to simulation, each accepted at its own p_2
simulation of the feasible configurations at $L = \lceil \log_2 F(N) \rceil$: the basic search for $N \le 8$ and $N = 11$, the refined search also for $N = 9, 10$	1,258 and 118,825 configurations, no mismatch
3. A search over standard networks, without the model	
$(N, L) = (12, 8), (13, 8), (12, 9), (13, 9)$	infeasible, infeasible, feasible, feasible
tests of its two lower bounds and its symmetry reduction on 351,541 prefixes of 20,369 networks, $4 \le N \le 13$	no violation
4. Consistency of the searches with (P1)–(P3)	
the refined search against the basic search: all runs of Table 2 with $N \le 17$, and $L = \lceil \log_2 F(N) \rceil$ for $N \le 8$ and $N = 11$	same verdicts and numbers of feasible configurations
the extended search against the refined search, on the refined search's 22 runs (Table 2 for $N \le 21$, and $L = \lceil \log_2 F(N) \rceil$ for $N \le 11$)	same numbers of skeletons and of feasible configurations for each shape
the extended search with the symmetry test against the extended search, on the same runs	feasible configurations in the same shapes; with the test switched off the same output, with it the same classes of feasible assignments
split runs against unsplit runs: the extended search with and without the symmetry test, 7 pairs (N, L), every kind of split used at $N = 28$, alone and nested (with the symmetry test, up to seven nested splits)	skeletons, feasible configurations and complete assignments add up
a separate run at $N = 28$ by the basic search, the refined search and two variants of the refined search, which differ only in how (S2) keeps one skeleton from each class, on all shapes except the 119 slowest (62,522 shapes)	no feasible configuration; for each shape not run by the basic search, the same number of skeletons as the extended search
The upper bound	
the networks of Proposition 2.5, $2 \le N \le 32$	$p_2 = \lceil \log_2 F(N) \rceil$

TABLE 3. The checks of the computation, in four groups, and a check of the upper bound. The supplement lists the programs and logs for each row, and every split of the run at $N = 28$.

Data and code availability

The programs, the logs of every computation in this paper, and the independent checks are archived on Zenodo as [19] (doi:10.5281/zenodo.23498529). They are the contents of the repository [16] at commit `be14346d67fd89b14f7e036ea03eae8d561b81ef`, where the four programs of Section 4.4 are `abs5` (basic), `abs9` (refined), `abs9f` (extended) and `abs9g` (extended, with the symmetry test). The Lean proofs of Section 7 read Harder's certificate [5], which the supplement does not contain; their build instructions download the file and check its SHA-256 hash. Part of the logs of the shapes searched with the symmetry test were written by four earlier revisions of `abs9g`; 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. This paper is doi:10.5281/zenodo.23730883. The code is under the Apache License 2.0, and the text under CC BY 4.0.

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] 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).

[4] R. W. Floyd and D. E. Knuth, *The Bose–Nelson sorting problem*, A Survey of Combinatorial Theory (J. N. Srivastava, ed.), North-Holland, Amsterdam, 1973, pp. 163–172, doi:10.1016/B978-0-7204-2262-7.50020-X.

[5] J. Harder, *Certificate of the minimal size of 11-channel sorting networks*, Zenodo, 2019, dataset, doi:10.5281/zenodo.4108365.

[6] J. Harder, *An answer to the Bose–Nelson sorting problem for 11 and 12 channels*, arXiv:2012.04400v3 [cs.DS], 2022.

[7] 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.

[8] 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.

[9] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2nd ed., Addison-Wesley, Reading, MA, 1998.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] T. S. Tomar, *Sorting networks on 13 inputs need 45 comparators*, Zenodo, 2026, doi:10.5281/zenodo.23504358.

[16] T. S. Tomar, *sortnet13-two-value-v5*, GitHub repository, `https://github.com/tejasvi/sortnet13-two-value-v5`, commit `be14346d67fd89b14f7e036ea03eae8d561b81ef`.

[17] T. S. Tomar, *Lean formalization of* $S(13) \geq 44$, part of the supplement [19].

[18] T. S. Tomar, *Kernel-checked Lean formalization of* $S(13) \geq 44$, part of the supplement [19].

[19] T. S. Tomar, *Programs and logs for "Sorting networks on 13 inputs need at least 44 comparators"*, Zenodo, 2026, doi:10.5281/zenodo.23498529.

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