

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 (the Bose–Nelson problem). The values are known for $N \leq 12$, and the best proved bounds for 13 inputs have been $43 \leq S(13) \leq 45$. We prove $S(13) \geq 44$. Deleting the comparators met by the largest and the second largest input leaves a sorting network on $N - 2$ inputs, so $S(N) \geq S(N - 2) + P(2, N)$, where $P(2, N)$ is the least possible worst-case number of comparators met by these two values. In 1972 Van Voorhis stated $P(2, N) \geq \lceil \log_2 F(N) \rceil$ for an explicit function F , but a 2026 audit showed that his proof is invalid. We determine $P(2, N)$ by computer instead. We show that the number of comparators met by the two values in each run is determined by the binary tree formed by the paths of the maximum, the comparators that these paths pass, and the depths at which the second maximum enters a forest of further comparators. This reduces the inequality $P(2, N) \leq L$ to a finite search. For $N = 13$ and $L = 8$ only 25 cases remain, and none of them can be completed, so $P(2, 13) = 9$, and Harder’s $S(11) = 35$ gives $S(13) \geq 44$. Independent checks confirm $P(2, 13) = 9$: a second implementation with its own lemmas, an exhaustive search over networks that does not use our structural reduction, and an integer program for each case. The same searches give $P(2, N) = \lceil \log_2 F(N) \rceil$ for $3 \leq N \leq 27$, which proves the lower bounds on $S(N)$ for $13 \leq N \leq 27$ listed in Dobbelaere’s public table.

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 $S(N) \geq S(N - 1) + \lceil \log_2 N \rceil$ [11] gives $S(13) \geq 43$, and this has been the best proved lower bound.

In a chapter published the same year [12], 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

$$(1) \quad S(N) \geq S(N - 2) + P(2, N),$$

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

Date: 18 October 2026.

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

Key words and phrases. Sorting networks, Bose–Nelson problem, comparator networks, lower bounds, computer-assisted proof.

$S(11) = 35$ 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 since 21 April 2025, together with lower bounds for $14 \leq N \leq 32$, with the note “on suggestion from Jelmer Fiset and based on principles in” Van Voorhis’s chapter. In 2026, however, an audit of the chapter [10] 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, a *pass-through*, and pass-throughs occur already in a size-optimal network on 3 inputs. The audit proves the inequality for two classes of networks and leaves it open in general. It also 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.

We do not repair Van Voorhis’s argument. Instead, we compute $P(2, N)$ for $N \leq 27$, which suffices for the tabulated bounds with $13 \leq N \leq 27$. Throughout, a dagger ($\dagger$) marks a statement whose proof includes a computation. The programs and their logs are in the supplement [14], which archives the repository [13].

Theorem 1.1. $S(13) \geq 44$. ^{$\dagger$}

Theorem 1.2. $P(2, N) = \lceil \log_2 F(N) \rceil$ for $3 \leq N \leq 27$. ^{$\dagger$}

Corollary 1.3. For $13 \leq N \leq 27$, $S(N)$ is at least the value $\Lambda(N)$ in Table 4. ^{$\dagger$} These values equal the lower bounds in Dobbelaere’s table [3].

The tabulated values for $28 \leq N \leq 32$ also need $P(2, 28) \geq 12$, which we do not prove here.

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, which two observations keep 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. Once 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), the costs of all $N(N - 1)$ runs are determined by the tree, the number of pass-throughs on each of its edges, and, for each place where the second maximum leaves the tree, the root and the depth at which it joins the forest that carries it on (Theorem 3.8). Kraft’s equality characterizes the possible depths. Second, the paths of the maximum are short in any network with small cost (Lemma 3.9), which bounds the height of the tree. At $N = 13$ and $L = 8$, for instance, only 4 tree shapes and 25 choices of the pass-through numbers remain, and the depth-first search over the forests rejects each of them.

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, Section 6 the independent checks, and Section 7 proves Theorem 1.1 and Corollary 1.3. No step uses Van Voorhis’s inequality $P(2, N) \geq \lceil \log_2 F(N) \rceil$ or the argument of [12].

2. NETWORKS, TWO-VALUE PRUNING AND AN UPPER BOUND

In this section we fix the model, prove inequality (1) and the monotonicity of $P(2, N)$, and construct networks that give the upper bound in Theorem 1.2.

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, $\text{hi}(g)$ and $\text{lo}(g)$. 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 $\text{hi}(g)$ and the smaller on $\text{lo}(g)$. 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). The audit [10] identifies it as equation (2) of [12] and does not dispute it. 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. $\square$

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 $P(2, N) \leq p_2(T') \leq p_2(T)$. $\square$

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). $\square$

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 $\text{dep}(c)$ 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

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

where c ranges over the internal vertices of B and B over the binary trees with N leaves. Van Voorhis introduced the function F [12]. The audit [10, Section 2] checks that f satisfies the recursion of [12, Theorem 1] and that $\min_B f(B)$ 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 $N \leq 27$. We computed it by a dynamic program over the pairs (height, f), and the values for $13 \leq N \leq 27$ agree with [10, Section 4.2].

N	3	4	5	6	7	8	9	10	11	12	13	14	15
$F(N)$	8	16	36	52	80	96	168	200	256	288	392	424	480
$\lceil \log_2 F(N) \rceil$	3	4	6	6	7	7	8	8	8	9	9	9	9
$\beta(N)$	3	4	5	6	6	6	7	8	8	8	8	8	8
N	16	17	18	19	20	21	22	23	24	25	26	27	
$F(N)$	512	784	848	960	1024	1232	1296	1408	1472	1872	1936	2048	
$\lceil \log_2 F(N) \rceil$	9	10	10	10	10	11	11	11	11	11	11	11	
$\beta(N)$	8	9	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 3.10.

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 $\text{dep}(x)$ comparators above x . The value m passes the comparators above y up to $c = \text{lca}_B(x, y)$, meets M at c and leaves on $\text{lo}(c)$. 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 $\text{lo}(c)$, 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. $\square$

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, which the search of Section 4 enumerates.

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 3.9.

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 $\text{hi}(g)$ is red if and only if g has a red input lead.*
- (b) *Join each red lead $\ell \neq o_M$ to $\text{hi}(g)$, 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 $\text{hi}(g)$, so $\text{hi}(g) \in P_x$. Conversely, if $\text{hi}(g) \in P_x$, 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 $\text{hi}(g)$ 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 $\text{hi}(g)$ 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 $\text{hi}(g)$ 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 $\text{hi}(g)$, which has one child, so it lies on an edge of B or above the root of B . $\square$

From now on T is a top-two correct network on $N \geq 3$ inputs and B is its tree. For a vertex v of B (an input or a branch node), $\text{seg}(v)$ 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 $\text{seg}(v)$. For a pass-through q let $u(q)$ be the vertex with q on $\text{seg}(u(q))$. 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 $\text{pre}(y, c)$ be the number of comparators on the path from y up to c , with c excluded, so that $\delta(x) = |A(c)| + \text{pre}(x, c)$ 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 $S(N) \geq S(N-1) + \lceil \log_2 N \rceil$ of Van Voorhis [11]. The audit [10, Section 6.1] reads Harder's proof of this bound [6, Lemma 17] as identifying the comparators on the paths of the maximum with the branch nodes, and observes that this does not affect the bound. The argument above does not need the identification.

3.2. The runs.

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

- (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 $\text{lo}(c)$.*
- (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) *$|W(x, y)| = \delta(x) + \text{pre}(y, c) + \tau(c)$, 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 $\text{lo}(g)$ 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 $\text{lo}(c)$.

(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 $\text{lo}(c)$ and $A(c)$.

(c) The run meets the $\delta(x)$ comparators on the path from x , the $\text{pre}(y, c)$ 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. $\square$

The same argument applies to every meeting. If m leaves a red comparator v on $\text{lo}(v)$ 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 $\text{lo}(v)$ starts at $\text{lo}(v)$ and, while the current lead enters a blue comparator g , moves to $\text{hi}(g)$. 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 $w(q, v) = \text{lca}_B(u(q), \bar{v})$, and let $\kappa(q, v)$ be the number of comparators from q up to $w(q, v)$, with $w(q, v)$ excluded:

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

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 $\text{lo}(v)$ in some run. In that run m follows the walk from $\text{lo}(v)$ 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 $\text{lo}(q)$, 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 $\text{lo}(w)$, 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 $\text{lo}(q)$, 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 $\text{hi}(q)$. 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 $\text{seg}(u(q))$, 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 $\text{seg}(w')$, 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 $\text{lo}(w)$. 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 $\tau(v)$. $\square$

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 $\text{hi}(q)$ and its port to $\text{lo}(q)$. The port carries neither M , being blue, nor m , and $\text{lo}(q)$ 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 $\text{hi}(q)$, 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 $\text{hi}(g)$ and that input lead to $\text{lo}(g)$. The value M never enters g , and m leaves g on $\text{hi}(g)$ 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. $\square$

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 $\text{hi}(g)$. 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 $\text{lo}(c)$ of all branch nodes c and the leads $\text{lo}(q)$ of those pass-throughs q for which $\text{lo}(q)$ is relevant.*
- (c) *A blue lead that is not relevant enters no comparator.*
- (d) *If $\text{lo}(v)$ is relevant for a red comparator v , then $\rho(v)$ is the root of the tree that contains $\text{lo}(v)$, and $d(v)$ is the depth of $\text{lo}(v)$ 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 $\text{hi}(g)$ carries m whenever that lead does, so $\text{hi}(g)$ 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 $\text{hi}(g)$ are the two input leads of g , both relevant by (N2). A relevant lead that is not $\text{hi}(g)$ 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 $\text{lo}(g)$ for a comparator g , which is not blue because m leaves blue comparators on hi . Each lead $\text{lo}(c)$ of a branch node c is relevant, since m leaves c on $\text{lo}(c)$ in the runs (x, y) with $\text{lca}_B(x, y) = c$.

(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 $\text{lo}(v)$ 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 $\text{seg}(v)$, 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 $\text{cost}_\Gamma(x, y) = \delta(x) + \text{pre}(y, c) + \tau_\Gamma(c)$ with $c = \text{lca}_B(x, y)$, and Γ is *feasible at level L* if $\text{cost}_\Gamma(x, y) \leq L$ 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 $v \rightarrow q$ 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 $\text{cost}_\Gamma(x, y)$ 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 $\text{lo}(q)$ 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 $\text{lo}(v)$ 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), $|W(x, y)| = \text{cost}_\Gamma(x, y)$ 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 $\text{lo}(v)$, 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 $\text{seg}(v)$, 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 $\text{lo}(c)$ avoids dead sources and ends at o_m . Thus m always ends on o_m , the network is top-two correct, and $|W(x, y)| = \text{cost}_\Gamma(x, y)$. $\square$

3.5. Bounds for the search.

Lemma 3.9 (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 $\delta(x)$. $\square$

Corollary 3.10. $P(2, N) \geq \beta(N) := \lceil \log_2 N \rceil + \lceil \log_2(N - 1) \rceil$ 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 3.9 applies. $\square$

Corollary 3.10 amounts to applying the one-value bound twice. For $3 \leq N \leq 27$ 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 3.11 (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*

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

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

4. THE SEARCH

We now describe the search for a configuration feasible at level L , for given $N \geq 3$ and L , and prove that each of its tests is valid. 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 $k(v) \in \{0, \dots, H\}$ 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 canonical string of the decorated tree.
- (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 $\sum 2^{-d} \leq 1$, a depth 0 occurs only as the only member, and a root whose sum has reached 1 receives no further member;
 - (b) an arc $v \rightarrow q$ 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 $d(c) + \pi(c) \leq \sigma(c, x)$ for each input x below c .
- (S5) At a complete assignment, check (K) and evaluate $\text{cost}_\Gamma(x, y)$ for all $x \neq y$. 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 $p_2 \leq L$. Theorem 3.8(a) gives a reduced network T^* with $p_2(T^*) \leq L$ whose configuration Γ is feasible at level L and has depths at most L . Lemma 3.9, applied to T^* , shows that the tree of Γ has height at most H and that $\delta(x) \leq H$ for each x . So (S1) and (S2) produce the skeleton of Γ or an automorphic copy, which

has the same costs after the inputs are renamed. Each $\sigma(c, x)$ is at least $\tau_\Gamma(c) \geq 0$ by Lemma 3.11, so (S3) keeps it. Condition (K) implies (a) for each partial assignment, (A) implies (b), and Lemma 3.11 gives (c). So the search reaches the assignment of Γ , and (S5) reports it feasible. $\square$

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 $v \rightarrow q$.

4.2. Three further tests. The program `abs9` adds 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

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

- (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 $\delta(x) + \text{pre}(y, c) + \tau^-(c) > L$ for some $x \neq y$ with $c = \text{lca}_B(x, y)$.

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 $\sum_c 2^{-\tau_\Gamma(c)} \leq 1$ 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 3.9, 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 $\text{lo}(c)$ lies in R_x , since it carries the maximum of T_x when that maximum starts on an input y with $\text{lca}_B(x, y) = c$. The comparators of T_x on the path in R_x from $\text{lo}(c)$ 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 $\text{lo}(c)$ lies on the path in R_x from another lead $\text{lo}(c')$ to o_m . Suppose it did. Then in some run m leaves c' , later enters c , and leaves c on $\text{lo}(c)$, 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 $\text{lo}(c)$ 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 $\text{lo}(c)$ is 2^{-j} , where j is the number of vertices with two children strictly above $\text{lo}(c)$. Such a vertex is $\text{hi}(g)$ for a comparator g of T_x on the path from $\text{lo}(c)$, so $j \leq \tau_\Gamma(c)$. Hence $\sum_c 2^{-\tau_\Gamma(c)} \leq 1$. Since $\tau_\Gamma(c) \leq \sigma(c, x)$ by Lemma 3.11, a configuration with $\sum_c 2^{-\sigma(c, x)} > 1$ is not feasible. $\square$

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 the audit [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 3.11 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 $\tau^-(v) \leq \tau_\Gamma(v)$ for every source v . $\square$

4.3. Programs. The program **abs5** implements (S1)–(S6). The program **abs9** adds (P1)–(P3) and can split the tree shapes among s processes by their index modulo s . For each tree shape the programs print the number of skeletons that pass (S3), and in **abs9** also (P1), the number of feasible configurations and the number of nodes of the depth-first search. A final line gives the totals, the number of complete assignments reached, and the number of mismatches found in (S6). Kraft sums are computed 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 $3 \leq N \leq 13$ and for $N = 17$ and 21, the values of N between 14 and 27 at which $\lceil \log_2 F(N) \rceil$ increases.

Theorem 5.1. *Let $N \in \{3, 4, \dots, 13\} \cup \{17, 21\}$ 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 $P(2, N) \geq \lceil \log_2 F(N) \rceil$.[†]*

Proof. The searches report no feasible configuration (Table 2). Proposition 4.1 applies, with Lemmas 4.2 and 4.3 for the run of **abs9**. $\square$

Note that for $N \in \{3, 4, 6, 10, 11\}$ no tree has height at most H , as Corollary 3.10 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.

Proof of Theorem 1.2. For $N \in \{3, \dots, 13\} \cup \{17, 21\}$ the lower bound is Theorem 5.1. For each other $N \leq 27$, let N' be the largest element of $\{13, 17, 21\}$ 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 $P(2, N) \geq P(2, N')$. The upper bound is Proposition 2.5. $\square$

6. VERIFICATION

We checked the computation in three ways, listed in Table 3. First, we determined $P(2, N)$ for $N \leq 13$ a second time, with programs that share no code with **abs5** and do not use the model of Section 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

N	L	H	program	shapes	skeletons	search nodes	complete	time (s)
3	2	1	abs5	0	0	0	0	0.00
4	3	1	abs5	0	0	0	0	0.00
5	5	3	abs5	2	11	58	3	0.00
6	5	2	abs5	0	0	0	0	0.00
7	6	3	abs5	1	2	24	0	0.00
8	6	3	abs5	1	1	6	0	0.00
9	7	4	abs5	8	309	1,273	2	0.00
10	7	3	abs5	0	0	0	0	0.00
11	7	3	abs5	0	0	0	0	0.00
12	8	4	abs5	7	67	130,268	57,650	0.01
13	8	4	abs5	4	25	456	0	0.00
17	9	5	abs5	187	262,776	567,225	0	4.69
21	10	5	abs9	177	62,879	453,320	0	3.81

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 **abs9** also (P1). “Search nodes” counts the nodes of the depth-first search, and “complete” the complete assignments it reaches. Times are wall-clock seconds on a shared server.

the runs. Methods B and C search a relaxed tree model that comes with its own structure theorem. The notes on these methods in the supplement [14] prove the Kraft inequality and the structure theorem. Second, we tested **abs5** and its model: an integer program decides each skeleton separately, and millions of networks are reduced, mapped to configurations and run through the code of **abs5**. Third, a search over standard networks that does not use the model settles $N = 12$ and $N = 13$ at $L = 8$ directly. Its pruning uses two lower bounds of Huffman type and a reduction by symmetry, which we tested on prefixes of networks. Unlike methods A, B and C, it comes without written proofs of its pruning rules.

7. CONSEQUENCES FOR SORTING NETWORKS

We now combine Theorem 1.2 with the known values of $S(11)$ and $S(12)$.

Proof of Theorem 1.1. By Lemma 2.2, $S(13) \geq S(11) + P(2, 13)$. Harder proved $S(11) = 35$ [6], with a certificate checked by a verified program [5], and Theorem 1.2 gives $P(2, 13) = 9$. Therefore $S(13) \geq 44$. $\square$

Proof of Corollary 1.3. For $N \geq 3$ we have $S(N) \geq S(N - 1) + \lceil \log_2 N \rceil$ [11] (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

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

By Theorem 1.2 and induction, $S(N) \geq \Lambda(N)$ for $13 \leq N \leq 27$. Table 4 lists $\Lambda(N)$. These are the lower bounds in Dobbelaere’s table [3]. $\square$

The best known sorting network on 13 inputs has 45 comparators [8], so $S(13) \in \{44, 45\}$. Deciding between the two values is beyond the method of this paper, because $P(2, 13) = 9$ is exact: the two-value bound with $S(11) = 35$ cannot give more than 44.

check	scope	outcome
abs9 against abs5	all runs of Table 2 with $N \leq 17$, and $L = \lceil \log_2 F(N) \rceil$ for $N \leq 8$ and $N = 11$	same verdicts and same numbers of feasible configurations
simulation (S6) of feasible configurations	$L = \lceil \log_2 F(N) \rceil$: abs5 for $N \leq 8$ and $N = 11$, abs9 also for $N = 9, 10$	1,258 and 118,825 configurations, no mismatch
networks of Proposition 2.5	all $N(N-1)$ runs, $2 \leq N \leq 32$	$p_2 = \lceil \log_2 F(N) \rceil$
method A: search over networks with exact state compression, isomorphism rejection and a Kraft prune	$3 \leq N \leq 13$ at $L = \lceil \log_2 F(N) \rceil - 1$, and witnesses at $L = \lceil \log_2 F(N) \rceil$ for $N \leq 12$	$P(2, N)$ 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: search over the relaxed tree model, with and without a height bound	$3 \leq N \leq 13, L = \lceil \log_2 F(N) \rceil - 1$ and $\lceil \log_2 F(N) \rceil$	$P(2, N)$ as in Theorem 1.2
test of the prunes of method A	all 15,113,524 states reachable with costs at most L , for $N = 8$ and $L = 6, 7, 8$	the prunes remove no state from which a goal is reachable
tests of the relaxed tree model	3,420 random top-two correct networks with $3 \leq N \leq 8$	no failure
one integer program per skeleton (HiGHS [7])	15 pairs (N, L) with $N \leq 13$, among them $(12, 8)$ and $(13, 8)$	the same skeletons admit a feasible configuration as in abs5 . At $(13, 8)$ none of 508 skeletons on ordered trees does
networks mapped into the model	4,464,896 top-two correct networks with $4 \leq N \leq 11$, among them all 3,972,276 on 4 inputs with at most 9 comparators. Each is reduced, mapped to a configuration and run through the code of abs5	costs equal to simulation, each accepted at its own p_2 , no failure
search over standard networks without the model	$(N, L) = (12, 8), (13, 8), (12, 9), (13, 9)$	infeasible, infeasible, feasible (21 comparators), feasible (23)
tests of the two lower bounds and the symmetry reduction of that search	351,541 prefixes of 20,369 networks, $4 \leq N \leq 13$	no violation

TABLE 3. Independent checks. The supplement lists the programs and logs for each row.

N	13	14	15	16	17	18	19	20
$\Lambda(N)$	44	48	53	57	63	68	73	78
N	21	22	23	24	25	26	27	
$\Lambda(N)$	84	89	95	100	106	111	117	

TABLE 4. The lower bounds $S(N) \geq \Lambda(N)$ of Corollary 1.3.

DATA AND CODE AVAILABILITY

The programs, the logs of every computation in this paper, and the independent checks are archived on Zenodo as [14] (doi:10.5281/zenodo.23453295). They are the contents of

the repository [13] at commit 385bbaf6188d9f08b5b734b487557e5a5c15d696. This paper is doi:10.5281/zenodo.23453293. The code is under the Apache License 2.0, and the text under CC BY 4.0.

USE OF AI TOOLS

The results were found and proved, the programs were written, and the text was drafted, reviewed and revised with Claude Code, Anthropic’s agentic coding tool, running the model Claude Opus 5.5. Literature searches used a search tool built on Google’s Gemini API with Google Search grounding, and the sources it returned were then checked directly. The author takes full responsibility for the content of the paper, including the parts produced with these tools.

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] 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.
- [12] 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.
- [13] T. S. Tomar, *sortnet13-two-value*, GitHub repository, <https://github.com/tejasvi/sortnet13-two-value>, commit 385bbaf6188d9f08b5b734b487557e5a5c15d696.
- [14] T. S. Tomar, *Programs and logs for “Sorting networks on 13 inputs need at least 44 comparators”*, Zenodo, 2026, doi:10.5281/zenodo.23453295.

INDEPENDENT RESEARCHER

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