A Logarithmic Time Sort for Linear Size Networks

JOHN H. REIF AND LESLIE G. VALIANT

Harvard University, Cambridge, Massachusetts

Abstract. A randomized algorithm that sorts on an N node network with constant valence in $O(\log N)$ time is given. More particularly, the algorithm sorts N items on an N-node cube-connected cycles graph, and, for some constant k, for all large enough α, it terminates within $k\alpha \log N$ time with probability at least $1 - N^{-\alpha}$.

Categories and Subject Descriptors: F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*parallelism, probabilistic computation*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumeric Algorithms and Problems—*routing and layout, sorting and searching*

General Terms: Algorithms, Theory, Validation

Additional Key Words and Phrases: Parallel algorithm, randomized algorithm, routing, sort, sorting network

1. *Introduction*

This paper is concerned with the problem of sorting N items in parallel on a natural fixed-connection graph G having N nodes labeled $\{0, 1, \ldots, N-1\}$ and constant valence. Each node initially contains one key. The set X of all N keys is assumed to have a total ordering $<$. The network sorts by routing each key $x \in X$ to node $j = \text{rank}(x)$, where $\text{rank}(x)$ is defined as $|\{x' \in X \mid x' < x\}|$. This can be viewed as a distributed packet routing problem. Each $x \in X$ is considered to be an atomic packet that has to be routed from its initial node to the node corresponding to its rank. Both the rank computation and the packet routing have to be realized in a completely distributed manner.

We assume that each node contains a single sequential processor with local storage for $O(\log N)$ packets. Thus memory per processor depends on network size but grows quite slowly. The processors are regarded as synchronous for the purpose of step counting, but the algorithm itself does not require global synchronization. In unit time interval a processor may transmit one of its packets along a departing edge and perform some elementary operation such as a comparison. The processors

A preliminary version of this paper appeared in *Proceedings of the 15th Annual ACM Symposium on Theory of Computing* (Boston, Mass., Apr. 25–27). ACM, New York, 1983, pp. 10–16.

The work of J. H. Reif was supported by the National Science Foundation grant MCS 82-00269 and the Office of Naval Research contract N00014-80-C-0674. The work of L. G. Valiant was supported by National Science Foundation grant MCS-83-02385.

Authors' present addresses: J. H. Reif, Department of Computer Science, Duke University, Durham, NC 27706; L. G. Valiant, Aiken Computation Laboratory, Division of Applied Sciences, Harvard University, Cambridge, MA 02138.

Preprint of paper appearing in Journal of the ACM(JACM), Vol. 34, No. 1, January 1987, pp. 60-76.

are capable of generating random bits of information and hence running randomized algorithms in the sense of Rabin [14] and Solovay and Strassen [16].

Clearly, the routing required to sort may require time at least the diameter of the graph. If G has constant valence, then the diameter is at least $\Omega(\log N)$. Hence the $O(\log N)$ time bound for our algorithm is asymptotically optimal. In this paper we restrict ourselves to demonstrating that this bound is achievable in principle and do not pursue the issue of the magnitude of the constant multipliers. We note, however, that it is within a large class of algorithms that is experimentally testable in the sense of [19].

Previous algorithms for sorting N keys on constant-valence, fixed-connection networks of N processors require time $\Omega(\log N)^2$. The bitonic sorter of Batcher [4] achieves this bound on such networks as the cube-connected cycles network [13].

For less realistic models of parallel computation faster algorithms have been known. For example, Wiedermann observed several years ago that the Quicksort of Hoare [8] takes time $O(\log N)$ with high likelihood on a parallel decision tree model. Hirschberg [7], Preparata [12], and Reischuk [15] have $O(\log N)$ time algorithms for various parallel random access models.

Our current algorithm follows the randomized routing ideas introduced in [19]. It can be viewed as a partially successful attempt at reducing the sorting problem to the apparently simpler problem of routing. In the analysis the critical path technique developed by Aleliunas [2] and Upfal [18] for analyzing routing in constant-valence graphs plays an important part.

Recent algorithms of Ajtai, et al. [1] and Leighton [10] achieve deterministic $O(\log N)$ time on $O(N)$ networks. The constant multipliers in their constructions are, however, enormous compared with ours. Also, in contrast with their expander networks, our networks are known to be well suited to many other parallel algorithms.

2. *Network Definitions*

We define various constant-valence networks derived from the n-dimensional binary hypercube. Consider some fixed $n \geq 1$. Let the *node set* be

$$V = \{(w, i) \mid w \in \{0, 1\}^n, \qquad i \in \{0, \ldots, n-1\}\},$$

which has cardinality $N = n2^n$ and hence $n < \log N$. (Logarithms are assumed to have base 2 unless otherwise indicated.) For each $a \in V$ let $address(a) = w$ and $stage(a) = i$ if $a = (w, i)$. Let $w = w[0] \cdots w[n-1]$ where $w[i]$ is the ith bit of w. Let $w' = \mathrm{EXT}(w, i)$ be identical to w, except that $w'[i] \neq w[i]$. Also let $\tilde{w}$ be the integer of which w is the binary representation.

We call an edge from node a to node b *internal* if address(a) = address(b) and *external* if address(b) = EXT(address(a), stage(a)). It is *forward* if stage(b) = stage(a) + 1 mod n, *static* if stage(b) = stage(a), and *reverse* if stage(b) = stage(a) − 1 mod n. The CCC_n network of Preparata and Vuillemin [13] has node set V and exactly all forward internal edges, reverse internal edges, and static external edges (see Figure 1). For ease of description this paper assumes a network more similar to that of Upfal [18], which we call CCC_n^+. It contains node set V and all forward and reverse internal edges and all forward external edges (see Figure 2). (N.B. The graphs we are defining are all undirected. Every edge supports bidirectional communication.)

PROPOSITION 2.1. *Any algorithm for CCC_n^+ can be simulated on CCC_n (and vice versa) with at most a factor-of-two increase in run time.*

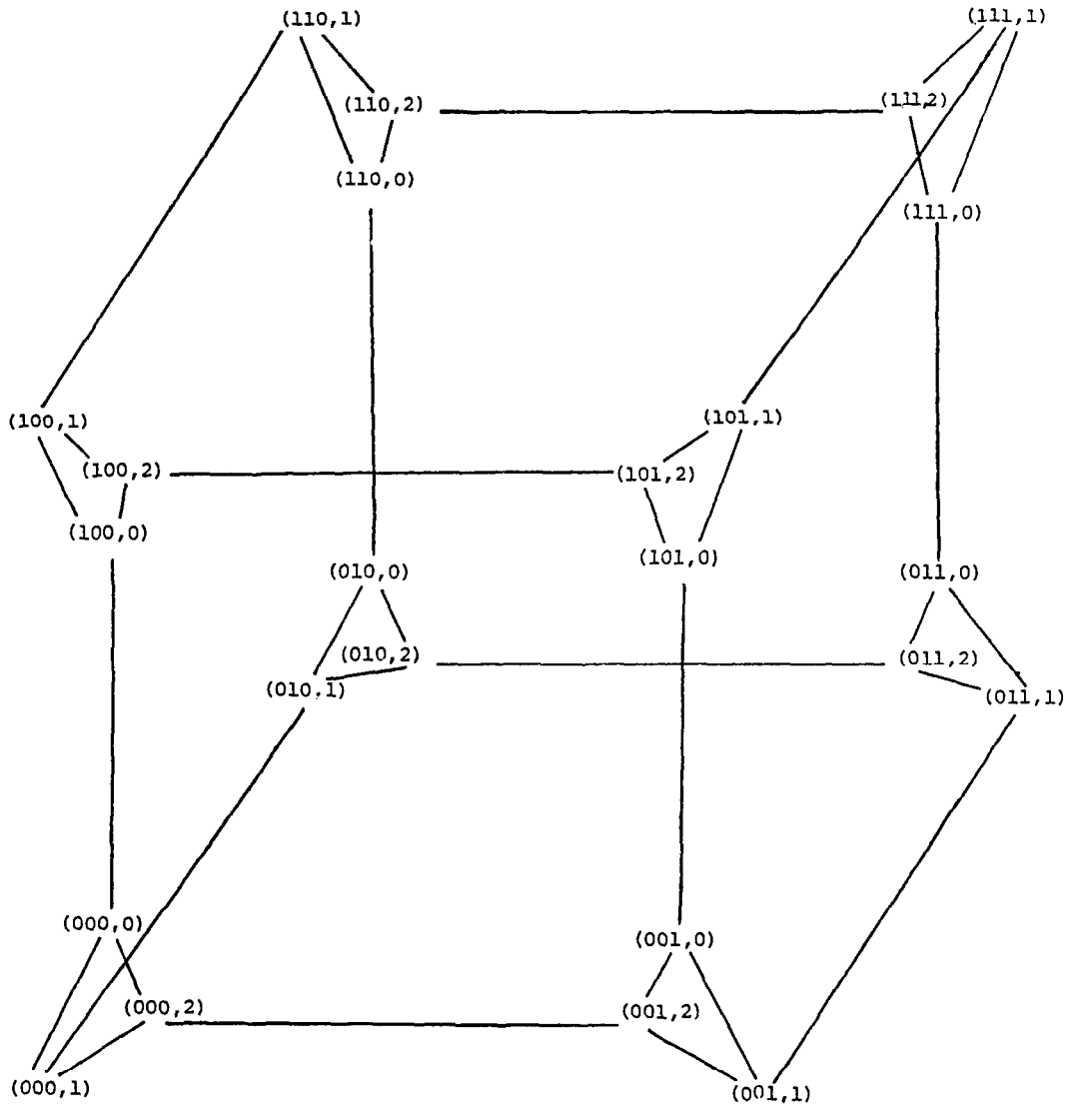

FIG. 1. The CCC_3.

In particular, to simulate on the CCC_n the traversal across a departing forward external edge of the CCC_n^+, we first traverse the departing static external edge and then traverse the departing forward internal edge. The simulation of the CCC_n^+ by the CCC_n is similar.

Finally, we define CCC_n^* (the CCC_n^* is also known as the BUTTERFLY network; see Ullman [17]) to be the network obtained by taking a CCC_n^+ and adding an additional set of nodes $\{(w, n) \mid w \in \{0, 1\}^n\}$ of stage n, and deleting each edge e of CCC_n^+ connecting (in either direction) a node $(w_1, n-1)$ of stage $n-1$ to a node $(w_2, 0)$ of stage 0, and substituting in its place an edge e' connecting (in the same direction) node $(w_1, n-1)$ to node (w_2, n) of stage n (see Figure 3).

The significance of CCC_m^* is that numerous copies of it can be found in CCC_n^* if $n > m$. In particular, for any w_1, w_2 such that $|w_1| + |w_2| = n - m$, the subgraph of CCC_n^* spanned by the nodes $\{(w_1 w w_2, i) \mid w \in \{0, 1\}^m$ and

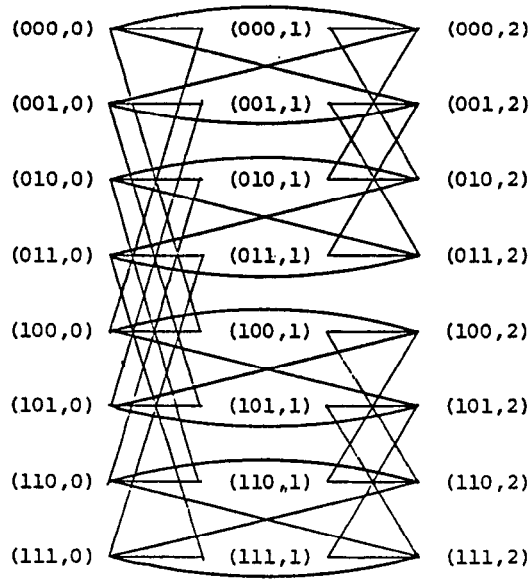

FIG. 2. The CCC_3^+.

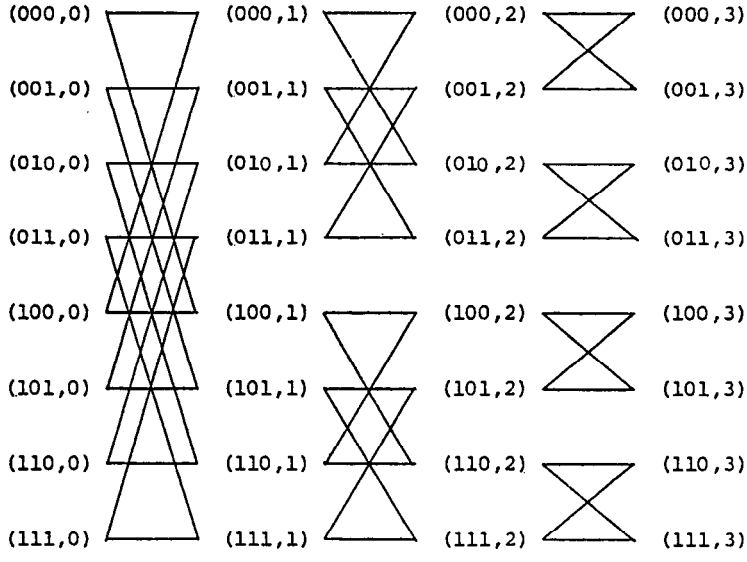

FIG. 3. The CCC_3^*.

$|w_1| \le i \le |w_1| + m\}$ is isomorphic to CCC_m^*. This property will be useful when our sorting algorithm makes recursive calls within subnetworks of the CCC_n^*.

Since each stage 0 node $(w, 0)$ of CCC_n^+ can be used to simulate the nodes $(w, 0)$ and (w, n) of CCC_n^*, we have

PROPOSITION 2.2. *An algorithm for CCC_n^* can be simulated on CCC_n^+ with at most a factor-of-two increase in run time.*

There exist various conditions on algorithms that are sufficient to ensure that they can be transferred from the CCC_n to CCC_n^* with only constant factor slowdown.

All the algorithms considered in this paper can be implemented on a CCC_n^* in this way.

Note that CCC_n and CCC_n^+ are naturally related to the n-dimensional hypercube H_n. Intuitively, for each $w \in \{0, 1\}^n$ the set of nodes $\{a \in V \mid \text{address}(a) = w\}$ can be considered to be a "supernode" of H_n. Each such supernode of H_n is connected by external edges to n other supernodes $\{b \in V \mid \text{address}(b) = \text{EXT}(w, i)$ for $i = 0, 1, \ldots, n-1\}$.

For any m let $\{0, 1\}^{\langle m \rangle}$ be the set of binary strings of length not more than $m - 1$. We define a subdivision of the node set V that indexes the subsets by binary strings from $\{0, 1\}^{\langle n+1 \rangle}$. For each $w \in \{0, 1\}^n$ let $V[w] = \{b \in V \mid \text{address}(b) = w$ and $\text{stage}(b) = 0\}$. For each $w \in \{0, 1\}^{\langle n \rangle}$ let $V[w] = \{b \in V \mid w$ is a prefix of address(b) and $|w| = \text{stage}(b)\}$. Thus $V[\lambda]$ is the set of nodes of stage 0 where λ is the empty string. Let the *root* $v[w]$ of $V[w]$ be the node with address $w0^{n-|w|}$ and stage $|w|$. Note that for $|w| \leq n-1$, $v[w]$ has a departing forward internal edge entering $v[w0]$ and a departing forward external edge entering $v[w1]$. Thus the roots are connected by a balanced binary tree of forward edges. By the *subcube* rooted at $v[w]$, we mean the set of nodes $\{b \in V \mid w$ is a prefix of address(b) and $|w| \leq \text{stage}(b)\}$.

3. *Packet Routing on the CCC_n^+*

This section briefly describes the probabilistic packet routing algorithm of Valiant and Brebner [20] which was applied to the CCC_n^+ by Upfal [18].

We require that each node $a \in V$ contain for each departing edge e a queue Q_e for the packets that are to be transmitted across edge e. Each node also contains its address and stage posted as local variables.

For some constant c let X be the set of cN packets to be routed. Each packet $x \in X$ is initially at a given node $I_x \in V$ and we wish x to be routed to given destination node $D_x \in V$. The algorithm has two phases:

A. (*Random Routing*). Route x from I_x to a node $R_x \in V$ with random address.
B. (*Fixed Destination Routing*). Route x from R_x to D_x.

The random routing of x in phase A is accomplished by repeating for n steps the transmission of x across a randomly chosen departing forward edge (i.e., transmit x across the forward internal edge or forward external edge with equal probability). The packets are then pipelined to the nodes with correct stage by traversing internal edges. Phase B repeats for n stages the following: If x is currently at node $a \neq D_x$ with $j = \text{stage}(a) + 1$ and $\text{address}(a)[j] = \text{address}(D_x)[j]$, then x is transmitted across the forward internal edge departing v and otherwise x is transmitted across the forward external edge departing v. This takes the packets to nodes with the correct addresses.

We have not yet specified the management of the queues of packets at each node. Suppose the *priority* of packet $x \in X$ is assigned to be the number of stages of phases A and B so far accomplished. We allow packet x to be transmitted from each node $a \in V$ if that queue does not contain any packets of lower priority. Let T_A, T_B be the total execution times of phases A and B, respectively. The techniques of Aleliunas [2] and Upfal [17] show the following under the assumptions that there are initially h packets at each node and at most h packets share a common destination node, h being any constant.

THEOREM D. *For some $c \geq 1$ for all sufficiently large α*

$$Pr(T_A > c\alpha n) < N^{-\alpha}, \quad \text{and} \quad Pr(T_B > c\alpha n) < N^{-\alpha}.$$

We note that, since the first phase sends packets to random addresses, the probability that, at its completion, there are more than $c_1\alpha n$ packets at any one node or $c_2\alpha n$ packets at any address can be similarly bounded by $N^{-\alpha}$ (for suitable constants c_1 and c_2).

In our algorithm here we need the following variant of phase A. Suppose now that in a CCC_n^* network the packets are initially all at the nodes of stage 0 with at most hn at each such node. The destinations are random and all at stage n. Suppose that phase A is implemented exactly as above, except for the queuing priorities (which are as described for SDR in Section 5). Then the proof technique of Theorem B easily yields the following:

THEOREM D′. *Under the above assumptions for some* $c \geq 1$ *for all sufficiently large* α

$$Pr(T_A > c\alpha n) < N^{-\alpha}.$$

Alternatively, it can be seen that Theorem D′ follows also from Theorem D, since the routing algorithm described uses the edges between stage $n-1$ and stage 0 at most twice for any packet, and this can be simulated by transmissions along chains of $\log n$ internal edges.

4. *Summary of Our Sorting Algorithm*

In defining sorting, the assumed ordering on the network nodes for the outputs is of little consequence since all reasonable choices can be achieved by a postprocessing routing phase. In the algorithm described below a key sorted to address w_1 will be smaller than a key sorted to address w_2 if and only if $\tilde{w}_1 < \tilde{w}_2$.

A summary of our algorithm for sorting on the CCC_n^+ is as follows:

Input: A set X of $N = n2^n$ keys, with one key at each node of CCC_n^+.

Step A: Call the splitter-finding procedure SF(λ). It finds a set of $2^n/n^6$ elements called *splitters* that divide X, when regarded as an ordered set, into roughly equal intervals. However, the rank in X of the splitters is not yet determined.

Step B: Route each packet to a random node. Call the splitter-directed routing procedure SDR(λ) with the splitters found in step A. This will route the keys belonging to each interval to the $(6\log n)$-dimensional subcube corresponding to it. In this way an approximate sort is achieved, but the keys are not spread completely uniformly around the network.

Step C: Compute the rank of each key using (deterministic) Algorithm C.

Step D: Route each packet to the node corresponding to its rank.

The two novel features of the algorithm are routines SDR, described in Section 5, and SF, which itself uses SDR as a subroutine, described in Section 7. SDR is somewhat like a routing algorithm, except that the destination of a key is determined by the rank of the key relative to a set of $2^l - 1$ previously chosen splitters ($l < n$). The destination will be the $(n-l)$-dimensional subcube corresponding to the splitter interval in which the key lies. The role of the routine SF is to find a set of nearly equidistant splitters. To achieve this, it calls itself recursively, the tree of recursive calls corresponding to the tree defined by the subcube containment structure. Calls of SF that are on distinct branches of the tree are executed asynchronously.

The $O(\log N)$ behavior of each of the four steps A–D is established, respectively, as follows: Theorem A (Section 7), Theorem B (Section 5), Algorithm C (Section 6), and Theorem D (Section 3). We note that Theorem B is invoked in Step B with $n - l = 6\log n$, which is sufficient for the $O(\log N)$ bound. The main result then follows immediately.

MAIN THEOREM. *There is a randomized algorithm that for some k and all n and all sufficiently large α sorts on a CCC_n network and terminates within $k\alpha n$ steps with probability greater than $1 - 2^{-\alpha n}$. The same statement applies also for a CCC_n^* network or a CCC_n^+ network.*

Note that our sorting algorithm below may fail in a given execution, with probability at most $2^{-\alpha n}$. This can be detected by simply timing the execution for $k\alpha n$ steps and then verifying in time $O(n)$ whether a sort has been achieved. In case a sort is not achieved within that time, the sorting algorithm is reinitialized and reexecuted. This is repeated until success is achieved. The probability of success is independent among different executions, even with the same input.

5. *Splitter-Directed Routing*

Let X be a set of cN keys that are totally ordered by the relation $<$. We assume that each key $x \in X$ is initially located at a random node in $V[\lambda]$ chosen independently of any other key in $X - \{x\}$. Suppose that for some l $(1 \leq l < n)$ we are given a set of *splitters* $\Sigma \subseteq X$ of size $|\Sigma| = 2^l - 1$. We index each splitter $\sigma[w] \in \Sigma$ by a distinct binary string $w \in \{0, 1\}^{\langle l \rangle}$ of length less than l. Let $<_\cdot$ denote the ordering defined as follows: For all $w, u, v \in \{0, 1\}^{\langle l \rangle} w\ 0\ u <_\cdot w <_\cdot w\ 1\ v$. (N.B.: If we label the edges of a balanced binary $(2^l - 1)$-node tree with 0 for a left branch and 1 for a right branch, then each node will be labeled by a word $w \in \{0, 1\}^{\langle l \rangle}$ that labels the path to it from the root. The ordering $<_\cdot$ on $\{0, 1\}^{\langle l \rangle}$ corresponds to the ordering of the nodes from left to right in this tree.) We require that, for all $w_1, w_2 \in \{0, 1\}^{\langle l \rangle}$, $\sigma[w_1] < \sigma[w_2]$ iff $w_1 <_\cdot w_2$. We assume that a copy of each splitter $\sigma[w]$ is already available in each node of $V[w]$.

Let $X[\lambda] = X$ where λ is the empty string. Initially we assume that the keys of $X[\lambda]$ are located at $V[\lambda]$, that is, the nodes of V having stage 0. The splitter-directed routing is executed in l temporally overlapping stages $i = 0, 1, \ldots, l - 1$. For each $w \in \{0, 1\}^i$ the set of keys $X[w]$ that are all eventually routed through $V[w]$ is defined recursively. The splitter $\sigma[w]$ partitions $X[w] - \sigma[w]$ into disjoint subsets

$$X[w0] = \{x \in X[w] \mid x < \sigma[w]\}$$

and

$$X[w1] = \{x \in X[w] \mid \sigma[w] < x\},$$

which are subsequently routed through $V[w0]$ and $V[w1]$, respectively.

Procedure SDR implements this as follows: Suppose that a key $x \in X[w]$ is located at a node $a \in V[w]$ with address ww' and stage i. Let B be the first bit of the address suffix w'. Then x is transmitted from node a across the departing forward internal edge if $B \equiv (\sigma[w] < x)$, and x is transmitted across the departing forward external edge otherwise. Thus if $x < \sigma[w]$, then x is transmitted to a node with address prefix $w0$, and if $\sigma[w] < x$, then x is transmitted to a node with prefix $w1$. The detail that remains to be specified is the queuing discipline used by each queue Q_e for forwarding packets. Before the start of SDR we give each packet a priority π, which is randomly chosen from $\{1, 2, \ldots, n\}$. When there is more than one packet in a queue, the one with the lowest numbered priority is transmitted first.

Note that at any one time distinct keys may be at distinct stages. When all the keys have completed stage $l - 1$, the keys $X - \Sigma$ are partitioned into 2^l disjoint subsets of the form $X[w]$ where $w \in \{0, 1\}^l$, and the keys $X[w]$ are then at addresses

prefixed by w. The sets $X[x]$ are thus uniquely defined by the choice of Σ. The following is an immediate consequence of the assumption that $\sigma[w_1] < \sigma[w_2]$ if $w_1 <\cdot\ w_2$:

LEMMA 5.1. *For any* $w_1, w_2 \in \{0, 1\}^l$, *if* $w_1 <\cdot\ w_2$, *then* $x_1 < x_2$ *for all* $x_1 \in X[w_1]$ *and* $X_2 \in X[w_2]$.

Also, since each packet is assumed initially to be at a random node and since the above-described SDR procedure does not modify the last $n - l$ bits in the address of a packet, these last bits remain random at the termination of SDR. Hence we can deduce:

LEMMA 5.2. *For each* $w \in \{0, 1\}^l$ *and each* $x \in X[w]$, *SDR takes* x *to a random node in* $V[w]$ *chosen independently of any other packet.*

Lemma 5.2 can be used to speed up the overall algorithm by avoiding repeated randomization but does not affect the order of the asymptotic run time.

The SDR procedure can be viewed as a generalization of phase B of the routing procedure described in Section 3. It routes packets from random source nodes to a subcube specified by the relative rank of the key among the splitters.

THEOREM B. *For any* n, $c \geq 1$ *consider the network* CCC_n^* *with a set* X *of* $cn2^n$ *packets including a set* Σ *of* $2^l - 1$ *splitters* (*where* $n \geq l$) *such that for all* $w \in \{0, 1\}^l \mid |X[w]| \leq 2cn2^{n-l}$. *Suppose that all the packets that are not splitters are at independently chosen random nodes of* $V[\lambda]$ *and that* T *is the total time for execution of SDR on this input. Then there exist constants* k *and* α_0 *such that for all choices of* n, l, *and* c *and for all* $\alpha \geq \alpha_0$

$$Pr(T > kc\alpha n) < 2^{-c\alpha n} + g(n, l),$$

where $g(n, l) \leq 2n \cdot 18^n \cdot exp(-2^{n-l})$.

PROOF. We adapt the critical path analysis technique introduced by Aleliunas [2] and Upfal [18] for analyzing routing algorithms on constant degree graphs. Note that our notion of priority (i.e., a randomly chosen integer from $\{1, \ldots, n\}$) is different from theirs.

For each node a and priority $\pi \in \{1, \ldots, n\}$, let *task* $\tau = (a, \pi)$ be the job of forwarding all keys of priority π that ever visit node a. Let the *task graph* be a directed graph with the set of possible tasks as nodes. The pair $((a, \pi), (b, \rho))$ is a (directed) edge in the task graph if *either* $a = b$ and $\rho = \pi + 1$ *or* (a, b) is an edge of the CCC* network and $\rho = \pi$. The two cases correspond to the two ways in which the execution of a task τ_2 may depend on the completion of task τ_1. In the first case τ_2 has to wait for packets with lower numbered priorities to be processed at its node. In the second case τ_2 has to wait for the arrival of a packet from an adjacent node.

The task graph is acyclic, since along each edge either the priority number or the node stage number increases by 1. The longest path in such a graph is therefore shorter than $l + n \leq 2n$. Define a *delay sequence* to be the sequence of tasks $(\tau_0, \tau_1, \tau_2, \ldots, \tau_d)$ along some path of the task graph, where the node of τ_0 has stage 0. The outdegree of the task graph is 3, since for each task there is just one edge corresponding to increasing the priority, and two edges corresponding to edge traversals in the CCC* network. Since $d < 2n$, the number of delay sequences starting at any one node is at most 3^{2n}. Since there are 2^n choices of starting node, the total number of delay sequences in the task graph is less than $2^n 3^{2n} \leq 18^n$.

In an execution of SDR the execution of the different tasks may temporally overlap in complex ways. For a particular execution and particular delay sequence $(\tau_0, \ldots, \tau_d)$, for each i $(0 \le i \le d)$, let f_i be the total number of packet transmissions involved in task τ_i, and let t_i be the time when the last such packet has been transmitted. We prove two claims. First, if the total execution time of SDR is T, then there is a delay sequence D such that

$$T(D) \stackrel{def}{=} \sum_{i=0}^{d} f_i$$

is at least T. Second, we show that for some constant k, for all D, for all $c \ge 1$ and all sufficiently large α,

$$\Pr(T(D) > kc\alpha n) < 2^{-c\alpha n - 5n} + 2n \exp(-2^{n-l}). \tag{1}$$

Since there are at most 18^n choices of D and since

$$18^n(2^{-c\alpha n - 5n} + 2n \exp(-2^{n-l})) \le 2^{-c\alpha n} + g(n, l),$$

the theorem has been proved.

To bound the run time T in terms of the delay sequences, we consider a *critical delay sequence* constructed as follows. For a fixed execution and any task τ, let t be the time when task τ is completed. The critical delay sequence is defined as follows: Let τ_d be one of the tasks that has the highest t value. Having defined τ_i, we define τ_{i-1} as one of the tasks τ such that (τ, τ_i) is an edge of the task graph and the completion time t for τ is highest among these. The construction stops when τ_i has stage 0 and priority 1. The $\{\tau_i\}$ are finally reindexed if necessary to start from τ_0.

We observe that for this delay sequence, for each i, $t_i - t_{i-1} \le f_i$: By the definition of τ_{i-1} all three of the tasks that are immediate preconditions for task τ_i are completed at or before time t_{i-1}. If $\tau_i = (a, \pi)$ all the packets with priority $\rho < \pi$ that ever go through a have gone through a by time t_{i-1}. Also all packets of priority π that ever go through the two predecessor nodes of a in the CCC* network have been forwarded by time t_{i-1}. Hence, if τ involves f_i packets, then it must be that $t_i \le t_{i-1} + f_i$. Hence the first claim that the maximal $T(D)$ is a true upper bound on the run time is established.

In order to establish the bound (1) on $T(D)$, consider any particular D and let $\tau_j = (a, \pi)$, where stage$(a) = i$, be any task in D. Let P_j be the set of keys that have nonzero probability of contributing to f_j (i.e., their keys destine them to addresses that agree with a in the first i bits) but would then certainly depart from D at τ_j. (Departure from D is forced *either* because $\tau_{j+1} = (a, \pi + 1)$, since the priority of a packet is invariant, *or* because $\tau_{j+1} = (b, \pi)$ for $b \ne a$ but the packet does not traverse network edge (a, b).) Note that in the latter case the $(i + 1)$st bit of the destination address of a packet that departs from D at τ_j is different from that of a packet that departs later.

Clearly P_j is contained in the union of $X[w]$ for the 2^{l-i} $w \in \{0, 1\}^l$ such that w and a agree in the first i bits. By the assumption about the size of $X[w]$ it follows that $|P_j| \le 2cn2^{n-i}$.

Now pick a random choice for the priorities of all the packets and consider it fixed for the rest of the argument. Let R_j be the subset of P_j that are assigned priority π. Since the priorities of P_j are determined by independent trials and the

probability of being assigned π is n^{-1}, Fact A3 can be used to give

$$\Pr(|R_j| \geq 4c2^{n-i}) \leq \exp(-c2^{n-i}) \leq \exp(-2^{n-l}). \tag{2}$$

Since there are $d < 2n$ choices of j, it follows that with probability greater than $1 - 2n \cdot \exp(-2^{n-l})$ every R_j will be of size at most $4c2^{n-i}$. Note also that the sets $\{R_j\}$ are pairwise disjoint, since the priorities have been fixed: In the case in which $\tau_{j+1} = (a, \pi + 1)$, R_j is the set of packets with priority π whose destinations agree with a in the first i bits. In the alternative case in which $\tau_{j+1} = (b, \pi)$, R_j is the set of packets with priority π whose destinations agree with a in the first i, but not the $(i + 1)$st bit. Hence no packet can belong to more than one R_j.

Finally let K_j be the actual set of keys that do depart from D at τ_j. The question whether, for any particular $x \in R_j$, it is the case that $x \in K_j$ is determined by whether the random initial position of x agrees with a in the last $n - i$ bits. In other words it is a Bernoulli trial with probability 2^{i-n} of success. Hence $|K_j|$ is upper bounded by $|R_j|$ such trials. Under the assumption that $|R_j| \leq 4c2^{n-i}$ the sum of the expectations for each j is at most $4c$, and hence the sum of the expectations of all the trials for all j is at most $8cn$. Hence a bound on the tail of the distribution of

$$\sum_{j=1}^{d} |K_j|$$

can be obtained directly from Hoeffding's theorem (Fact A4) by substituting $8cn$ for the sum of the expectations of the independent trials. An appeal to Chernoff's bound (Fact A2) gives

$$\Pr(\Sigma |K_j| \geq c_3 c\alpha n) < 2^{-c_3 c\alpha n} \tag{3}$$

if $c_3\alpha > 8e^2$.

From previous discussion it is clear that once the priorities are fixed, if a key x does intersect D, then it must depart from it at a unique point that does not depend on the random initial position of x. Repeated involvement of x in D prior to departure is possible only if it corresponds to a chain of edge traversals by x in the CCC* network. We analyze the extent of such previous involvements by first fixing some choice of $K_1, \ldots, K_d$. If a key was involved in D at τ_j, then the probability of a previous involvement at τ_{j-1} is at most one half, independent of subsequent involvements (i.e., If τ_j, τ_{j-1} have different priorities then this probability is 0. Otherwise, it is $\frac{1}{2}$, depending as it does on one bit of the initial random address of the key.). Hence if a key was involved in D at τ_j, then the probability of t previous involvements (i.e., with $\tau_{j-1}, \ldots, \tau_{j-t}$) is at most 2^{-t}. It follows that

$$\Pr(T(D) \geq K + s \mid \Sigma K_j \leq K) \leq 2^{-s}.$$

Hence, assuming as we do here the condition that, for all j, $|R_j| \leq 4c2^{n-i}$, a condition that fails to hold with probability less than $2n \exp(-2^{n-l})$,

$$\begin{aligned}\Pr(T(D) &\geq 2c_3 c\alpha n) \\ &\leq \Pr(T(D) \geq 2c_3 c\alpha n \mid \Sigma |K_j| \leq c_3 c\alpha n) + \Pr(\Sigma |K_j| \geq c_3 c\alpha n) \\ &\leq 2^{-c_3 c\alpha n} + 2^{-c_3 c\alpha n}.\end{aligned}$$

If also $c_3 \geq 2$ and $k \geq 2c_3$, we obtain (1) as claimed. □

We note that the crucial properties of the task graph, which are shared with those used by Aleliunas [2] and Upfal [18], are that they are of depth $O(n)$, they have only $2^{O(n)}$ paths through them, and that the expected execution time of a task is a constant. The reason why we need a different notion of priority from theirs is that in our case the packets are initially distributed among a fraction of $1/n$ of the nodes of the network, namely, those of stage 0, and the stage numbers of the nodes are always monotonically increasing along the path of a packet in our algorithm.

6. *Deterministic Sorting and Routing*

We use some known deterministic algorithms as subroutines in various parts of our algorithm. A crucial step in the splitter finding is sorting a sparse subset of elements. Although $N^{1-\epsilon}$ keys, for any $\epsilon > 0$, can be sorted on various N-node networks (see Nassimi and Sahni [11]) in $O(\log N)$ time, the case in which there are fewer than $N^{1/2}$ keys gives a particularly simple procedure.

LEMMA 6.1. *For any $r < n/2$, 2^r keys can be sorted in deterministic $O(n)$ time on a CCC$_n^*$. Both input and output are assumed to be at nodes of stage zero and address $w0^{n-r}$ for $|w| = r$. Their ordering is defined by the $\tilde{w}$.*

PROOF. In the algorithm we only route packets along disjoint paths or broadcast them along disjoint binary trees. There is no contention or use of queues.

First, for each $w_1 \in \{0, 1\}^r$ the key at address $w_1 0^{n-r}$ is broadcast to addresses (i) $ww_1 0^{n-2r}$ and (ii) $w_1 w 0^{n-2r}$ for all $w \in \{0, 1\}^r$. In case (ii) the binary trees for transmission are immediate. In case (i) we first route from $w_1 0^{n-r}$ to $w_1 w_1 0^{n-2r}$, go back to stage 0, and then use binary trees to broadcast to $ww_1 0^{n-2r}$.

For any pair of keys, say those originating at $w_2 0^{n-r}$ and $w_3 0^{n-r}$, there are addresses ($w_2 w_3 0^{n-2r}$ and $w_3 w_2 0^{n-2r}$) at which they meet. When they meet, they are compared. The rank of w_2 can be computed at address $w_2 0^{n-r}$ by collecting and accumulating the results of comparisons performed at $\{w_2 w 0^{n-2r}\}$, along the binary tree rooted at $w_2 0^{n-r}$. If the rank of w_2 turns out to be $\tilde{w}_4$, then this key has to be routed to address $w_4 0^{n-r}$. This can be done without contention from other keys by sending it along the route $w_2 0^{n-r} \to w_2 w_4 0^{n-2r} \to w_4 w_4 0^{n-2r} \to w_4 0^{n-r}$ (with appropriate stage numbers). □

Step C of the overall algorithm determines the rank of every element, given that it is "almost" sorted. Suppose that for some i we have that all elements are in nodes at stage i and for all $w_1 <\cdot\ w_2$, $|w_1| = |w_2| = i$, the keys in $V[w_1]$ are smaller than the keys in $V[w_2]$. If $i = n$, then we have a complete sort, except that the elements may not be uniformly distributed among the stage n nodes. In this situation the rank of each key can be determined by first doing a bubble-sort on their local cycle. The global rank computation is performed on the binary tree that has nodes as leaves, and consists of all forward internal edges and just those forward external edges along which some address bit changes from 0 to 1. The number of keys in each subcube can be determined recursively by sending these sums from the leaves toward the root and accumulating at each internal tree node. Finally, in a reverse information flow from the root to the leaves, the range of the ranks in each subcube can be determined, and hence the ranks of the individual keys. This all takes $O(n)$ parallel transfers of tokens that contain only binary numbers of $O(n)$ digits. (Note that this rank computation can be made symmetric, in the sense that the processors in the internal nodes all execute the same algorithm.)

In Step C of the actual algorithm we start with only a partial sort (i.e., for all $w_1 <\cdot\ w_2$ with $|w_1| = |w_2| = n - s$ where $s = 6\log_2 n$, for all $x \in V[w_1]$ and

$y \in V[w_2]$, $x < y$). To find ranks in this situation, we determine the rank range for each $X[w_1]$, sort each $X[w_1]$, and finally deduce the rank of each element. The determination of the rank ranges and final rank is as described in the above paragraph. With overwhelming probability each $X[w_1]$ will have at most $2n2^s$ packets. For sorting $X[w_1]$ we assign a separate CCC_t^* to it, where $t = s + \log n - \log s$. At least if t divides n, one can find $n2^n/(t2^t)$ disjoint copies of CCC_t^* in CCC_n^*. The packets are routed to their appropriate copy of CCC_t^* (Theorem D) and then sorted there by some $O(n)$ method such as Batcher's (see Preparata and Vuillemin [13]), which takes $O(\log n)^2$. The above-described algorithm for ranking the elements given a partial sort is called *Algorithm C.*

7. *Splitter Finding*

We describe a procedure SF that, given a CCC_n^* with c packets at each node, finds a subset U of $2^n/n^\delta$ packets called *splitters* that divide the ordered sequence of the $cn2^n$ total packets into intervals that are, with large probability, all of length smaller than $2cn^{\delta+1}$. The procedure is recursive, nested recursive calls corresponding to nested subcubes. The splitters found up to the ith level of recursion divide the ordered sequence into $2^{n(1-2^i/3^i)}$ roughly equal intervals. The subcubes at the ith level are CCC_m^*, where $m = (\frac{2}{3})^i n$ $[i = 0, \ldots, \log_{3/2}(n/2\delta \log n))]$. At the ith level a fraction of about $(\frac{2}{3})^i$ of the packets are considered *active.* The choice of splitters at lower levels is restricted to these active elements. In this way the average density per node of active packets in each CCC_m^* is kept a constant c independent of the cube size. This is necessary for the recursive procedure to succeed. Any integer greater than or equal to 6 suffices as a value of δ.

The set U of all splitters found in a run of SF[λ] is used in Step B of the overall sorting procedure.

The procedure SF applied to the subcube with rote $v[w]$, where $m = n - |w_1|$, is as follows. When the procedure is called initially with $w = \lambda$, all the packets are considered active.

Procedure SF(w)

(1) Let $Y[w]$ be the active packets in $V[w]$. For each $x \in Y[w]$ route x to a random node in $V[w]$.

(2) For each w_1, $|w_1| = m/3 + 2\log n$, choose at random an active element from $V[ww_1]$. (This can be done by a "random tournament," where we allow each node $v \in V[ww_1]$ to attempt to route a copy of a packet currently at v to the root node $v[ww_1]$. If two or more distinct copies are routed through the same node, all but one are randomly deleted, so only one arrives at $v[ww_1]$. This process is deterministic time $O(m)$, since packets travel without contention from the leaves toward the root of a depth $O(m)$ binary tree.)

(3) Let S^* be the set of $n^2 2^{m/3}$ elements found by step (2).

(i) Sort S^* in the CCC* subcube rooted as $v[w]$ using Lemma 6.1.

(ii) For each $j = 1, 2, \ldots, 2^{m/3} - 1$ deterministically route the element ranked jn^2 in S^* to the address whose binary representation is $\tilde{w} + j2^{m/3}$ and stage $n - 2m/3$. Let the set of these elements be S. The members of S are now declared splitters. For each such splitter consider the address ww_1 to which it has been sent. Suppose that w_3 is the longest suffix of w_1 that are all zeros (i.e., $w_1 = w_2 w_3$ where $w_3 \in 0^*$). Denote this splitter by $\sigma[ww_2]$. (Note that this ordering on the splitters satisfies the defined requirement that $\sigma[w_4] < \sigma[w_5]$ iff $w_4 <\cdot\ w_5$.)

(iii) Route each splitter $\sigma[ww_2]$ first to the root of $V[ww_2]$. Then broadcast it to every node in $V[ww_2]$.

(4) For each $x \in Y[w] - S$ decide according to a Bernoulli trial with probability $\frac{2}{3}$ whether it is to remain active. Let the active subset of $Y[w]$ be $Z[w]$. This step takes constant time.

(5) Apply SDR with the newly found splitters to the set of packets $Z[w]$.

(6) For each w' with $|w'| = m/3$ let $Y[ww']$ be the subset of $Z[w]$ routed to subcube $V[ww']$ by (5). For each such w' call in parallel SF(ww') for $Y[ww']$ as active elements, unless $m = 2\delta \log n$.
End of procedure SF.

We have seen that SDR for CCC_m^* takes time $O(m)$ with overwhelming probability. Theorem A will establish that, if SF is run, with parallel recursive calls of SF being allowed to be asynchronous, then the overall algorithm runs in time $O(n)$ with large probability. The main fact that has to be established (Theorem 7.2) is that with overwhelming probability, at every call of SDR the hypotheses of Theorem B are satisfied. All the other operations performed in a call of SF(w) with $|w| = n - m$ can be achieved deterministically in time $O(m)$, with the exception of step (1) which is governed by Theorem D'. (Note that by Lemma 5.2, if step (1) is done initially in a call of SF(λ), then it is not necessary to call it again in subsequent recursions.)

The above description of a call of SF(w) assumes that each step is completed before the next one starts. For the randomized routines in (5), as well as (1), there exist deterministic $O(m)$ algorithms, which can be called at any time during the execution of the randomized routines, that determine whether the execution is completed. Hence, if such algorithms are initiated at every m time units, then the completion of the randomized routines can be detected without increasing the order of the total run time.

First we need a technical lemma that asserts that a set of $2^{m/3}$ roughly equally spaced elements can be selected from a set T by first selecting a set S^* of $n^2 2^{m/3}$ such elements randomly, sorting these, and picking $2^{m/3}$ equally spaced elements from the latter list. We denote by $\omega(1)$ any function $g(r)$ that tends to infinity as $r \to \infty$.

LEMMA 7.1. *Given an ordered set T, suppose that a set S^* of $n^2 2^{m/3}$ elements is chosen from T at random and S^* is then sorted. Let $S \subseteq S^*$ be the subset of elements having positions $n^2, 2n^2, \ldots, (2^{m/3} - 1)n^2$ in the ordered set. Let $N = n2^n$.*

Suppose $t_0, \ldots, t_{f+1}$ is one of the longest ordered subsequences of T such that $t_0, t_{f+1} \in S$ but $t_1, \ldots, t_f \notin S$. Then

(i) $Pr(f > (1 + n^{-1/3})|T|/2^{m/3}) = N^{-\omega(1)}$,
(ii) $Pr(f < (1 - n^{-1/3})|T|/2^{m/3}) = N^{-\omega(1)}$.

Suppose that a subset $Y \subseteq T - S$ is chosen by performing independent Bernoulli trials with probability $\frac{2}{3}$. Let $y_0, \ldots, y_{h+1}$ be the longest ordered subsequence of $Y \cup S$ such that $y_0, y_{h+1} \in S$ but $y_1, \ldots, y_h \notin S$. Then

(iii) $Pr(h > (\frac{2}{3})(1 + 2n^{-1/3})|Y|/(2^{m/3})) = N^{-\omega(1)}$,
(iv) $Pr(h < (\frac{2}{3})(1 - 2n^{-1/3})|Y|/(2^{m/3})) = N^{-\omega(1)}$.

These claims assume that $n^4 2^{m/3} = o(|T|)$ and $n^2 2^{m/3} = o(|T|^{2/3})$.

PROOF. All choices of S^* are equally likely. To prove (i) and (ii), consider any ordered subsequence $t_0, \ldots, t_f$ of T with $f = (1 \pm n^{-1/3})|T|/2^{m/3}$. Then the probability that of the $n^2 2^{m/3}$ members of S^* exactly n^2 lie in the above range and the rest outside is

$$\binom{|T| - f}{n^2 2^{m/3} - n^2}\binom{f}{n^2} \Big/ \binom{|T|}{n^2 2^{m/3}}.$$

Applying Fact A6 with $A = |T| - f$, $a = f$, $X = n^2 2^{m/3} - n^2$, $x = n^2$ gives $G = n^{5/3}$ and an upper bound of

$$\exp\left(\frac{-n^{4/3}}{6}\right),$$

provided $n^4 2^{m/3} = o(|T|)$ and $n^2 2^{m/3} = o(|T|)^{2/3})$. This establishes (i) and (ii) since there are at most 2^n choices of t_0, t_{f+1} and f, respectively.

To show (iii) and (iv), it is sufficient to prove that, in a sequence of $(1 \pm n^{-1/3})|T|/2^{m/3}$ ordered elements of T, the probability that the number of elements chosen to be in Y is outside the range $(\frac{2}{3})(1 \pm 2n^{-1/3})|T|/2^{m/3}$ is negligible. Fact A3 upper bounds this probability by

$$\exp(-n^{-2/3}|T|/(4 \cdot 2^{m/3})),$$

which is bounded above by $\exp(-n^{4/3})$ if $2^{m/3} \cdot n^2 = o(T)$. □

The probabilistic elements in a call of SF may misbehave in three ways: Step (2) may fail to find a complete set of potential splitters. The splitters found in steps (2) and (3) may not split the whole set into equal enough intervals. Step (4) may keep a wrong ratio of packets active. Parts (a)–(c) of the following theorem treat these three points.

THEOREM 7.2. *In a run of SF(λ) the probability of each of the following events for each recursive call of SF(w) is bounded above by $N^{-\omega(1)}$ provided that $m \geq 12 \log n$.*

(*a*) *Step* (2) *fails because $V[ww_1]$ has no active packets.*
(*b*) *In the call SF for subcube w with $|w| = n - m$, two neighboring splitters are created that in the total ordering of the $cn2^n$ elements have more than $2cn2^m$ elements between them.*
(*c*) *In the call SDR for subcube w with $|w| = n - m$, it happens that $|Z[w]| > 2cm2^m$ or $|Z[w]| < cm2^m/2$.*

Since in a run of SF[λ] there are at most 3N such events altogether, the probability that any such event ever *occurs in a run is therefore also bounded by $N^{-\omega(1)}$.*

PROOF. The proof proceeds by induction on the depth of recursion. We assume that the theorem holds down to the current level of recursion and argue that the probability of "going wrong" at the current call is less than $N^{-\omega(1)}$.

(a) Since the active elements of $Y[w]$ have been sent to random nodes in $V[w]$, the probability that all of at least $cm2^m/2$ elements miss $V[ww_1]$ is at most

$$\left(1 - \frac{1}{(2^{m/3}n^2)}\right)^{cm2^m/2}.$$

By Fact A1 (see Appendix) this is bounded above by

$$\exp\left(-\frac{cm2^{m/3}}{(2n^2)}\right) = N^{-\omega(1)}$$

if $m \geq 12 \log n$.

(b) We assume inductively that, in the call of SDR at the ith level of recursion, the set of elements in the corresponding subcube had size at most $(1 + n^{-1/3})^i cn2^m$ (where $m = n(\frac{2}{3})^i$). Applying Lemma 7.1(i) gives that, at the next level of recursion,

the probability of a subcube having more than $(1 + n^{-1/3})2^{-m/3}$ times as many packets is bounded above by $N^{-\omega(1)}$.

(c) We assume inductively that, in the call of SDR at the ith level of recursion, the size of the set of active elements in the subcube corresponding to w is in the range $(\frac{2}{3})^i(1 \pm 2n^{-1/3})^i cn2^m$. Then by Lemma 7.1(iii) the probability that the number of active elements in a subcube at the $(i + 1)$st level call is in the range $(\frac{2}{3})(1 \pm 2n^{-1/3})2^{-m/3}$ times this quantity, which is $(\frac{2}{3})^{i+1}(1 \pm 2n^{-1/3})^{i+1}cn2^{2m/3}$, is bounded by $N^{-\omega(1)}$. □

THEOREM A. *For all $c \geq 1$ there is a c_2 such that for all sufficiently large β if $SF(\lambda)$ is run on CCC_n^* with c packets per node and T is its run time, then*

$$Pr(T > cc_2\beta n) < N^{-c\beta}.$$

PROOF. In a run of SF a critical path is a sequence of nested calls of $SF(\lambda)$, $SF(w_1)$, $SF(w_1w_2)$, ..., $SF(w_1w_2 \cdots w_i)$, ..., where $|w_j| = 2^j3^{-j-1}n$. The deterministic components of each take time proportional to $|w_j|$. When summed for $i = 1, \ldots, \log_{3/2}(n/(12\log n))$, this gives an upper bound of $O(n)$ as required. Hence it remains only to analyze the cumulative probabilistic effects of such a chain of calls of SDR. Note that these calls are probabilistically independent.

Consider a call of SDR on the subcube with address prefix $w_1 \cdots w_i$, and with a set of $2^{l_i} - 1$ splitters, where $l_i = (n - |w_1 \cdots w_i|)/3$. Theorems B and 7.2(c) say that, for all large enough α ($\alpha \geq \alpha_0$), this call of SDR exceeds run time $kc\alpha(\frac{2}{3})^i n$ with probability less than

$$2^{-c\alpha(2/3)^i n} + 2n \cdot 18^n \cdot \exp(-2^{6\log n}) + N^{-\omega(1)} \leq 2^{-c\alpha(2/3)^i n} + 2^{-\Omega(n^5)}.$$

Hence it exceeds run time $kc\alpha_0(\frac{2}{3})^i n + t_i$ with probability less than

$$2^{-t_i/2k}$$

for all large enough n, since, if t_i is written as $kc(\alpha - \alpha_0)(\frac{2}{3})^i n$, then $t_i/k \leq c\alpha(\frac{2}{3})^i n$. It follows that the probability that such a sequence of nested calls takes time more than $(\frac{3}{2})kc\alpha_0 n + t$, where $t = (\frac{3}{2})kc(\alpha - \alpha_0)n$, is less than

$$\begin{aligned}\sum_{\Sigma l_i = t} \prod_i 2^{-t_i/(2k)} &\leq \sum_{\Sigma l_i = t} 2^{-t/(2k)} \\ &\leq 2^{-(3/4)c(\alpha-\alpha_0)n + O((\log n)^2)} \\ &\leq 2^{-c(\alpha-\alpha_0)n/2}\end{aligned}$$

for all large enough n. Note that the summation is over $n^{O(\log n)}(\log n)$-tuples $(t_0, t_1, \ldots, t_{\log n})$. Putting $\beta = \alpha/4$, $c_2 = 6k$ gives that for $\alpha \geq 2\alpha_0 \geq 4$

$$\begin{aligned}\Pr(T > c_2c\beta n) &\leq \Pr(T > (\tfrac{3}{2})ck\alpha n) \\ &\leq 2^{-c\alpha n/4} \\ &\leq N^{-c\beta}\end{aligned}$$

for all sufficiently large n. By then making c_2 large enough, the result holds for all n. □

In conclusion we note that Theorem 7.2(b) ensures that the hypotheses of Theorem B hold when step B of the overall algorithm is invoked. In other words, when SDR is called with the $2^n/n^6$ splitters found in step A, then the number of elements destined for each $(6\log n)$-dimensional subcube is never more than twice the average.

In the unlikely event that step (2) of SF fails in any call of SF, the sorting algorithm is restarted from the beginning.

Appendix. Some Combinatorial Identities

We use the following inequalities. Let exp denote exponentiation of Euler's constant e.

FACT A1. *For all* $x > 0$ $(1 + x^{-1})^x < e$ *and* $(1 - x^{-1})^x < e^{-1}$. *Since for all large enough* x $(1 + x^{-1})^x(1 + (4x)^{-1}) < e < (1 + x^{-1})^x(1 + (2x)^{-1})$, *it follows that* $(1 - x^{-1})^{-x} < e(1 + 4(5(x - 1))^{-1})$ *and* $(1 + x^{-1})^{-x} < e^{-1}(1 + (2x)^{-1})$.

Let $B(m, N, p)$ be the probability that in N independent Bernoulli trials with probability p of success there are at least m successes.

FACT A2 [5]

$$B(m, N, p) \leq \left(\frac{Np}{m}\right)^m \left(\frac{N - Np}{N - m}\right)^{N-m}$$

$$\leq exp(-m - Np) \qquad \text{if } m > Npe^2.$$

FACT A3 [3] *If*

$$m = Np(1 + \beta) \qquad \text{where } 0 \leq \beta \leq 1,$$

then

$$B(m, N, p) \leq exp\left(-\frac{\beta^2 Np}{2}\right).$$

The following result bounds the tails of distributions of the sums of independent trials with unequal probabilities (Poisson trials) in terms of those with equal probabilities (Bernoulli trials).

FACT A4 [9]. *If we have* N *independent trials with respective probabilities* $p_1, \ldots, p_N$, *where* $\Sigma p_i = Np$, *and if* $m \geq Np + 1$ *is an integer, then the probability of at least* m *successes is at most* $B(m, N, p)$.

FACT A5 [6, p. 18]. *If* $n = o(N^{2/3})$, *then*

$$\binom{N}{n} = (1 + o(1))\frac{N^n}{n!} exp\left(-\frac{n^2}{2N}\right).$$

FACT A6. *Suppose* $x < a$, $x < X < A$ *are all functions of* n *such that* $Xx = o(A)$ *and* $X = o(A^{2/3})$. *Let* $x = aP \pm G$, $X = AP \mp G$ *where* $P = (X + x)/(A + a)$, $G = o(aP)$, *and* $G = o(AP)$. *Then for all large enough* n

$$\frac{\binom{a}{x}\binom{A}{X}}{\binom{a + A}{x + X}} \leq (1 + o(1))exp\left(-\frac{G^2}{5aP}\right).$$

PROOF. Applying Fact A5 gives

$$\binom{A}{X} = (1 + o(1))\frac{A^X}{X!} \exp\left(-\frac{X^2}{2A}\right)$$

and

$$\binom{A + a}{X + x} = (1 + o(1))\frac{(A + a)^{X+x}}{(X + x)!} \exp\left(-\frac{X^2 + 2xX + x^2}{2A}\right).$$

Using $Xx = o(A)$ and applying Stirling's formula to $X!$, $(X + x)!$ and $x!$ give

$$\frac{\binom{a}{x}\binom{A}{X}}{\binom{a+A}{x+X}} < \left(\frac{a}{x}\right)^x \left(\frac{A}{X}\right)^X \left(\frac{X+x}{A+a}\right)^{X+x} (1 + o(1)).$$

Substituting $x = aP + G$ and $X = AP - G$ (or $x = aP - G$ and $X = AP + G$) and using Fact A1 give the claimed bound. □

We assume throughout the paper that ratios take integral values whenever this is convenient and otherwise insubstantial.

ACKNOWLEDGMENT. We are grateful to G. H. Gonnet for pointing out an error in an earlier version of the paper and to D. Krizanc for his comments on this one.

REFERENCES

1. AJTAI, M., KOMLÓS, J., AND SZEMERÉDI, E. An $O(n \log n)$ sorting network. In *Proceedings of the 15th ACM Symposium on Theory and Computing* (Boston, Mass., Apr. 25–27). ACM, New York, 1983, pp. 1–9.
2. ALELIUNAS, R. Randomized parallel communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing* (Ottawa, Ont., Canada). ACM, New York, 1982, pp. 60–72.
3. ANGLUIN, D., AND VALIANT, L. G. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. Comput. Syst. Sci. 18*, 2 (1979), 155–193.
4. BATCHER, K. Sorting networks and their applications. In *AFIPS Spring Joint Computer Conference*, vol. 32. AFIPS Press, Reston, Va., 1968, pp. 307–314.
5. CHERNOFF, H. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Stat. 23* (1952), 493–507.
6. ERDÖS, P., AND SPENCER, J. *Probabilistic Methods in Combinatorics.* Academic Press, Orlando, Fla., 1974.
7. HIRSHBERG, P. S. Fast parallel sorting algorithms. *Commun. ACM 21*, 8 (1978), 657–661.
8. HOARE, C. A. R. QUICKSORT. *Comput. J. 5*, 1 (1962), 10–15.
9. HOEFFDING, W. On the distribution of the number of successes in independent trials. *Ann. Math. Stat. 27* (1956), 713–721.
10. LEIGHTON, T. Tight bounds on the complexity of parallel sorting. In *Procedings of the 16th ACM Symposium on Theory of Computing* (Washington, D.C., Apr. 30–May 2). ACM, New York, 1984, pp. 71–80.
11. NASSIMI, D., AND SAHNI, S. Parallel permutation algorithms and a new generalized connection network. *J. ACM 29*, 3 (1982), 642–667.
12. PREPARATA, F. P. New parallel-sorting schemes. *IEEE Trans. Comput. C27*, 7 (1978), 669–673.
13. PREPARATA, F. P., AND VUILLEMIN, J. The cube-connected cycles: A versatile network for parallel computation. *Commun. ACM 24*, 5 (May 1981), 300–309.
14. RABIN, M. O. Probabilistic algorithms. In *Algorithms and Complexity*, J. F. Traub, Ed. Academic Press, Orlando, Fla., 1976.
15. REISCHUK, R. A fast probabilistic parallel sorting algorithm. In *Procedings of the 22nd IEEE Symposium on Foundations of Computer Science.* IEEE, New York, 1981, pp. 212–219.
16. SOLOVAY, R., AND STRASSEN, V. A fast Monte-Carlo test for primality. *SIAM J. Comput. 6* (1977), 84–85.
17. ULLMAN, J. D. *Computational Aspects of VLSI.* Computer Science Press, Rockville, Md., 1984.
18. UPFAL, E. Efficient schemes for parallel communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing* (Ottawa, Ont., Canada). ACM, New York, 1982, pp. 55–59.
19. VALIANT, L. G. A scheme for fast parallel communication. *SIAM J. Comput. 11*, 2 (1982), 350–361.
20. VALIANT, L. G., AND BREBNER, G. J. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing* (Milwaukee, Wis., May 11–13). ACM, New York, 1981, pp. 263–277.