Note On The AKS Sorting Network

(Expository note)

by

Richard Cole
Colm Ó'Dúnlaing

Ultracomputer Research Laboratory
Courant Institute of Mathematical Sciences
715 Broadway, 10th Floor
New York, NY 10003

Ultracomputer Note #109
Computer Science Department Technical Report #243

This work was supported in part by an IBM Faculty Development Award and by NSF grants DCR-84-01633 and DCR-84-01898.

ABSTRACT

The Ajtai-Komlos-Szemeredi sorting network, as improved and simplified by Michael S. Paterson, is described.

1. Introduction.

Efficient sorting circuits have been known for a considerable time. Batcher [Ba-1968] described a circuit of depth $1/2 \log N(\log N + 1)$ for sorting N numbers. For a long time the question of whether there was an $O(\log N)$ depth sorting circuit was open. This question was resolved by [AKS-1983] who gave a construction (we will refer to it as the AKS network). The constant (multiplying log N) in this construction was enormous. Their method was subsequently considerably simplified by Paterson [Pa-1983]. This note presents Paterson's method, with some further simplifications. Even though the constant here is considerably reduced by comparison with the original AKS network, it is still impractically large. The question of whether there exists a practical $O(\log N)$ depth sorting network is still a major open question. Incerpi and Sedgewick have conjectured that an approach based on Shellsort might lead to such a network [IS-1984].

2. The tree-sort metaphor.

Let N be the number of keys to be sorted; suppose for convenience that N is a power of 2. Imagine a tree T with N leaves and depth $\log N$. Sorting the keys means storing each key in a leaf of T so that left-to-right traversal of these leaves yields the keys in ascending order. We could imagine the following sorting algorithm, which begins with all keys attached to the root of T, and successively redistributes the keys down the tree until each key is attached to the correct leaf of T. The algorithm proceeds in $\log N$ parallel stages: at each stage the keys at a node are split evenly into two halves, the lower half going to the left child and the upper half going to the right child. If all the splitting is done exactly then we have a sorting algorithm. Unfortunately, an exact split cannot be accomplished in bounded depth, so the given scheme could at best be realized as a $\log^2 N$ time algorithm. The AKS algorithm makes a plausible compromise: at each stage, an *approximate split* is performed in bounded depth, with a small but nonzero chance of error. Keys are redistributed after each stage, but they are not all passed down the tree. Some keys are sent back up, the aim being to enable those keys which were incorrectly placed to move to the correct part of the tree. Intuitively, after this process is continued sufficiently long all errors have the opportunity to correct themselves, and the keys have all settled in the correct leaf nodes.

3. Expander graphs and approximate splitting.

Following the treatment in the Ajtai-Komlos-Szemeredi paper, we define a (k,ε)-*expander graph* to be a directed bipartite graph with two sides A and B of equal cardinality, and edge-set E, such that for every subset X of A, the set Γ_X of vertices adjacent to vertices in X, i.e.,

$$\Gamma_X = \{ y \in B: \exists x \in X ((x,y) \in E) \}$$

has a given lower bound on its cardinality, namely,

$$|\Gamma_X| \geq \frac{1-\varepsilon}{\varepsilon}\min(\varepsilon|B|,|X|).$$

A similar formula is required to hold for all subsets Y of B:

$$|\Gamma_Y| \geq \frac{1-\varepsilon}{\varepsilon}\min(\varepsilon|A|,|Y|).$$

Furthermore, it is required that the edges E be partitioned into k disjoint matchings $M_1, \cdots, M_k$.

An expander graph is crucial to the approximate splitting technique. It proceeds as follows. Suppose that A and B are two arrays of the same size n, each containing sort keys. We wish to move the keys in $A \cup B$ so all the smaller keys are in A and all the larger keys in B. As noted before, we do not wish to accomplish this exactly, but with only a certain tolerance of error. Given an expander graph on $A \cup B$, supposing its edge-set partitioned into matchings $M_1 \cdots M_k$ as described above, we define an operation $\text{SPLIT}_\varepsilon(A,B)$ as follows:

```
FOR i:= 1 TO k DO
  FORALL edges (u,v) IN M_i DO in parallel
    IF A[u] > B[v] THEN
      swap A[u] with B[v];
```

We shall see that SPLIT_ε approximately splits in a sense now to be made precise. Define an *initial segment* S of the set of keys stored in $A \cup B$ as any subset of keys satisfying

$$y \in S \text{ and } x \text{ is a key in } A \cup B \text{ and } x \leq y \text{ implies } x \in S.$$

Similarly one can define a terminal segment in the obvious analogous sense.

Lemma. Suppose that S is an initial segment of keys in $A \cup B$ and $|S| \leq n$ (here $n = |A| = |B|$). Then after SPLIT_ε is performed, the number of keys in S which are not stored in A is at most $\varepsilon|S|$. A similar and symmetric result holds for terminal segments $|S|$.

Proof. Let Y be the set of locations in B whose entries contain keys in S after the operation is performed, and as before let Γ_Y denote the set of nodes in A adjacent to Y in the graph. We know that every key in Γ_Y is no larger than at least one key in S, so S contains the keys in both Y and Γ_Y. Note that $|S| \geq |Y| + |\Gamma_Y|$ since Y and Γ_Y are disjoint. If $|Y| \leq \varepsilon|A|$ we can deduce that $|Y| + |\Gamma_Y| \geq |Y|/\varepsilon$; thus $|Y| \leq \varepsilon|S|$ as asserted. The other case, where $|Y| > \varepsilon|A|$ leads us to the conclusion that $|Y| + |\Gamma_Y| > |A|$ which contradicts the bound on $|S|$. •

We are interested in performing a more special-purpose split than that just described, for the purposes of the redistribution indicated above. Suppose that we have n keys stored in an array B. We wish to define a parallel operation $\text{CENTRIFUGE}_{\varepsilon,\lambda}(B)$ which has the property that for any initial (respectively, terminal) segment S of length at most $\lambda n/2$, at most $\varepsilon|S|$ keys remain outside the lower

(respectively, upper) $\lambda n/2$ locations after the operation, and also for any initial (respectively, terminal) segment S of size at most $n/2$, at most $\varepsilon|S|$ keys in S remain in the lower (respectively, upper) $n/2$ locations in the array after the operation. It is straightforward to implement $\text{CENTRIFUGE}_{\varepsilon,\lambda}$ as a sequence of t $\text{SPLIT}_{\varepsilon/t}$ operations on subintervals of the array, where $t = \lceil \log_2(2/\lambda) \rceil$. (Below, values $\lambda = 1/8$, $t = 4$ will be chosen.)

We remark that if we use a probabilistic construction for an expander graph then the $\text{SPLIT}_{1/d}$ circuit has depth $O(d \log d)$ while if we use a deterministic construction it has depth $O(d^2)$. The probabilistic result can be shown by a straightforward counting argument (consider bipartite graphs whose edge sets comprise c d log d random matchings, where c is a constant independent of d; for suitable c, at least half of the resulting graphs are expander graphs). The deterministic result has been shown by Pippenger [Pi-1986] to be a consequence of the recent [LPS-1986] deterministic expander graph construction.

4. Parametric definition of network. Strangeness.

As said already, the sorting network is described using a metaphor of a tree coupled with the parallel operation $\text{CENTRIFUGE}_{\varepsilon,\lambda}$ used to redistribute the keys. Here we introduce several parameters of Paterson's construction which will later be replaced by well-chosen constants.

[1] The algorithm is conceived as attaching keys to nodes of a complete binary tree T with N leaves: we assume that N is a power of two.

[2] Each stage of the algorithm is governed by a distribution of keys in 'bags' among the nodes of T. Each 'bag' has a 'capacity' which diminishes from stage to stage. All bags at the same level ℓ in the tree have the same capacity, $rA^{\ell-1}$, where r is a quantity which diminishes between stages and A is a constant greater than 1. (The root is a special case; it has capacity $(1 - \lambda)r$. λ is described in [5], below.)

[3] At any stage of the algorithm, all the bags at alternate levels (odd or even depending on the stage) are empty. There is a maximal depth d at which nonempty bags exist, and that depth increases from stage to stage. We speak of the nodes at that maximal depth as forming the 'frontier,' so the frontier descends from stage to stage.

[4] Initially, the bag at the root has capacity N, and has all the keys attached to it. Ultimately, all bags except the leaves will have zero capacity (actually, capacity less than one), and each leaf bag will have capacity 1. At any time a bag may possibly not be filled to capacity, but it will never be filled beyond its capacity.

[5] There is a constant λ which governs the redistribution of keys between stages. Suppose that a bag (at a node v) is nonempty at a given stage. Suppose that it has capacity b, and for the moment suppose that v is not the root and the bag is full. $\text{CENTRIFUGE}_{\varepsilon,\lambda}$ is performed on the keys in the bag, where ε is another parameter to be instantiated, in an array B of size b, say; then the leftmost $\lambda b/2$ and the rightmost $\lambda b/2$ keys are returned to the parent node, and of the remaining $(1-\lambda)b$ keys, the left half are sent down to the left child and the right half are sent down to the right child.

If v is the root, nothing is sent 'up': half is sent down to the left child and half to the right child; this is achieved with a SPLIT_ε operation.

Actually, a bag may be only partially filled. In this case, the $\text{CENTRIFUGE}_{\varepsilon,\lambda}$ operation is modified (as explained in Section 7) to divide the keys as follows. If there are more than λb keys they are divide into 4 sets: the leftmost $\lambda b/2$ keys, the rightmost $\lambda b/2$ keys, the left half of the remaining keys, and the right half of the remaining keys; the keys are distributed as before. If there are at most λb keys they are simply returned to the parent.

[6] Given a key x, its *target* is the leaf node in which it should be stored upon completion of the sort. (This definition assumes that all keys are distinct without prejudicing the correctness of the sorting method.) During the algorithm it may be stored at nodes which are not ancestors of the target node. The *strangeness* of a key x stored at node u at some point of the algorithm is the difference in levels between u and v where v is the least common ancestor of u and the target of x. Intuitively, the strangeness of a key is the shortest distance it must ascend the tree before descending to its target node. Call a key *strange* or *a stranger* if it has nonzero strangeness. Of course, a key may become strange and cease to be strange several times during the course of the algorithm.

Errors in the $\text{CENTRIFUGE}_{\varepsilon,\lambda}$ operation will cause keys to acquire nonzero strangeness, and it is our job to make the number of strangers dwindle to zero.

For a node u, define its level i descendants to be the descendants of u exactly i levels deeper in the tree. For any node u, for $s \geq 1$, define $S_s(u)$ to be the number of keys of strangeness s *or greater* currently in the bags associated with the level $(s-1)$ descendants of u divided by the current capacity of these bags, if these bags are not empty; otherwise define $S_s(u) = 0$.

Summarizing the critical parameters.

They are: λ, which determines how many keys should be sent back up and how many sent down; A, the expansion ratio of the bag capacities (with increasing depth); and ε, the error tolerance in the CENTRIFUGE operation.

Together with these parameters we shall choose two more parameters, μ and δ, which will allow the following **critical invariant** to be maintained: for all nodes u, at all stages of the algorithm, and for all $s \geq 1$,

$$S_s(u) \leq \mu\delta^s. \qquad [A]$$

Finally, there is one derived parameter $\nu < 1$ which represents the rate of decrease of bag capacity from stage to stage.

5. Constraints upon the parameters.

In order to obtain a correctly terminating algorithm, certain constraints must be maintained among the several parameters $N, T, \lambda, \varepsilon, A, \mu, \delta$, and ν. We should always bear in mind that the critical constraint is constraint [A] listed above.

First, let us consider the deflation rate ν, which we require to be strictly less than 1. This parameter depends explicitly on the others: consider a node u, not the root, which is currently empty with capacity b. At the next stage its capacity will be νb. Its children currently have capacity Ab and will each send a proportion λ of their keys up to u; its parent has capacity b/A and will send a proportion of $(1-\lambda)/2$ of its keys down to u. Cancelling out the factor b we obtain the constraint

$$\nu = 2\lambda A + \frac{1-\lambda}{2A} < 1. \qquad [B]$$

Next, we maintain the root under capacity to avoid figuring out what to do with spare keys; i.e., the root always sends all its keys down, and therefore should be maintained at $(1-\lambda)$ of its capacity. But we wish also that it maintain at least as high a deflation rate as the other nodes in T. When it receives keys it only receives them from its children, so in order that it continue to be at $(1-\lambda)$ capacity (or less), we require

$$2\lambda A \le (1-\lambda)\nu. \qquad [C]$$

Suppose that x is a key with strangeness $s > 0$; if it is 'low,' i.e., its target node is to the left of its current node u, then x is at the lower end of the range of keys stored at u, since all keys $y \le x$ must have strangeness s or greater; similarly if it is 'high.' This is why the extremes of the bag are sent back up the tree (with a small error tolerance). We must maintain λ sufficiently large to accommodate all strangers (in the absence of splitting error). It is conceivable that all strangers in a node are at the same end (all low or all high), and therefore we require that $\lambda/2 \ge S_1$, i.e., in view of [A],

$$2\mu\delta \le \lambda. \qquad [D]$$

Next, for $s \ge 2$, we want to adjust the parameters so that invariant [A] is maintained from stage to stage (the case $s = 1$ is much trickier and will be treated separately). Suppose that b is the current capacity of the node, so its capacity at the next stage will be νb. Consider its level $(s-1)$ descendants. They can import keys of strangeness s in two ways: (i) keys from their children which had strangeness $s+1$, and (ii) 'misaddressed' keys from their parents which had strangeness $s-1$. The contribution, when we scale the respective bag capacities, comes out as $((\varepsilon b)S_{s-1} + 4A^2 bS_{s+1})(2A)^{s-2}$, so, substituting for S_i according to [A] and cancelling out common terms we obtain the constraint

$$4A\delta^2 + \frac{\varepsilon}{A} \le 2\nu\delta. \qquad [E]$$

Finally, we consider the constraints necessary to maintain S_1 sufficiently low. Consider a node v with parent u and sibling w. Let us write T_v for the subtree of T rooted at v, similarly, T_u and T_w. We shall speak of 'strangers in T_u' and so forth to mean keys in T_u whose targets are outside the subtree T_u.

Ultimately, the number of keys whose targets are in T_u precisely matches the number of leaves in T_u. If there were no strangers anywhere in the tree T, these keys would be distributed as follows: all the keys in T_u; and (ignoring rounding), for each even-level ancestor z of u, 2^{-d} of the keys stored at z,

where d is the height of z above u. Now we want to estimate the maximum number of strangers that could be passed to node v at this stage. For clarity, we assume that v is a left child.

Let b be the capacity of v. There are two obvious sources of strangers passed to v.

(i) keys of strangeness 2 or greater passed up from the children of v; this contributes at most

$$2bAS_2$$

strangers to v.

(ii) 'misaddressed' keys sent to v from u which should have been sent elsewhere. Consider the t $\text{SPLIT}_{\varepsilon/t}$ operations performed at node u in the current stage. Either a misaddressed key is too large, in which case it was (incorrectly) placed in the left half of u's bag by the first $\text{SPLIT}_{\varepsilon/t}$ operation, or the misaddressed key is too small, in which case it was not placed in the leftmost $\lambda/2$-th portion by the final $t-1$ $\text{SPLIT}_{\varepsilon/t}$ operations (recall that $t = \lceil \log 2/\lambda \rceil$). The first term contributes a splitting error of $\varepsilon/t \cdot b/2A$, while the second term contributes a splitting error of $\frac{\varepsilon(t-1)}{t} \cdot \frac{b\lambda}{2A}$ strangers. Thus the splitting error can then contribute

$$\frac{\varepsilon b}{2At} + \frac{\varepsilon(t-1)b\lambda}{2At}$$

more strangers to v.

The last source of strangers is from u, not through error, but through skewness. Of all the $(1-\lambda)b/A$ keys which are split between v and w, not counting strangers at the node, there could be an excess ξ b of keys whose targets are in T_w. Since splitting error has already been accounted for, we can assume that precisely half of this excess will be sent down to v. There are three sources of excess here.

(iii) If all the strangers in u were at the rightmost end of u, then S_1 of the keys in u which belonged in v were sent up. This contributes

$$\frac{b}{A}S_1$$

'skew' keys to u.

(iv) It is possible that all the keys stored at proper ancestors of u and at other nodes outside T_u, which belong in T_u, actually belong in T_v. Let us count the keys stored at proper ancestors of u which belong in T_u. There is a 'legitimate' contribution of 1/4 of the keys stored at the grandparent of u, and so forth: this amounts to

$$\frac{b}{4A^3} + \frac{b}{16A^5} \cdots$$

i.e.,

$$\frac{b}{A(4A^2-1)}$$

Any excess over this legitimate contribution must be balanced by strangers in T_u, so we add to it the total count of strangers in T_u:

$$\frac{bS_1}{A} + 4bAS_3 + 16bA^3S_5 + \cdots$$

i.e., in view of [A],

$$\frac{b\mu\delta}{A(1-4A^2\delta^2)}$$

(v) Last, it is possible that there be no strangers in T_v but many in T_w; this increases the 'demand' for keys in T_w, and what is more to the point, all of these keys may be in u. This gives a contribution of

$$2bAS_2 + 8bA^3S_4 + \cdots$$

i.e., in view of [A],

$$\frac{2b\mu A\delta^2}{1-4A^2\delta^2}$$

keys to the skewness. So ξ is bounded by

$$\sigma = \frac{\mu\delta}{A} + \frac{1}{A(4A^2-1)} + \frac{\mu\delta(1+2A^2\delta)}{A(1-4A^2\delta^2)}.$$

Half of the terms contributed in (iii), (iv), and (v) must be added into the possible number of strangers passed to v. In conclusion, if the constraint [A] is to be maintained at the next stage for S_1, we require that

$$2A\mu\delta^2 + \frac{\varepsilon}{2At} + \frac{\varepsilon(t-1)\lambda}{2At} + \frac{\mu\delta}{2A} + \frac{1}{2A(4A^2-1)} + \frac{\mu\delta(1+2A^2\delta)}{2A(1-4A^2\delta^2)} \leq \nu\mu\delta. \qquad [F]$$

Remark: We observe that we have overcounted the number of strangers contributed by u to v. For in bounding the number of skew keys we assumed, in term (iii), that all the strangers in u were at the right end of u's bucket, in which case they do not contribute to the second error term in (ii) above. So let c be the number of keys with strangeness 1 or greater at the left end of u's bucket. Then the number of skew keys passed to v is $\max[0, \frac{b\sigma - c}{2}]$. And the error, in term (ii) above, is actually bounded by

$$(\frac{b}{2A} - \max[0, \frac{b\sigma - c}{2}])\frac{\varepsilon}{t} + \frac{c(t-1)\varepsilon}{t}.$$

Thus the total error is bounded by:

$$2A\mu\delta^2 b + \max[0, \frac{b\sigma - c}{2}]\cdot(1 - \varepsilon/t) + \frac{b\varepsilon}{2At} + \frac{c(t-1)\varepsilon}{t}$$

Clearly this is maximized when $c = 0$ (provided $\varepsilon \leq \frac{t}{2t-1}$; $\varepsilon \leq 1/2$ suffices).

Also, in (v) we assumed that there were no strangers in T_v, while in (i) we count strangers contained in T_v. Thus suppose that there are $2bAS_2 - d$ keys of strangeness at least 2 in v's children; the term in (v) is reduced by the same amount. The total error is maximized when $d = 0$. Thus the constraint [F] can be replaced by the constraints

$$2A\mu\delta^2 + \frac{\sigma}{2}(1 - \varepsilon/t) + \frac{\varepsilon}{2At} \le \nu\mu\delta \qquad \textbf{[G]}$$

with

$$\sigma = \frac{\mu\delta}{A} + \frac{1}{A(4A^2-1)} + \frac{\mu\delta(1 + 8A^4\delta^3)}{A(1 - 4A^2\delta^2)} \qquad [H]$$

and

$$\varepsilon \le 1/2. \qquad [I]$$

6. Fixing the parameters.

We start by evaluating the running time. The initial bucket capacity of the leaves is $\frac{N}{1-\lambda}A^{\log N}$, since the root initially has capacity N, which corresponds to a regular node of capacity $\frac{N}{1-\lambda}$. After T stages, the buckets at the leaves have capacity $\frac{N}{1-\lambda}A^{\log N}\nu^T$. The sort is complete when the buckets at the leaves have capacity 1, that is when $T = \frac{\log 2A}{\log 1/\nu}\log N + 0(1)$. Recall that each stage requires a $\text{CENTRIFUGE}_{\varepsilon,\lambda}$ operation (i.e. t $\text{SPLIT}_{\varepsilon/t}$ operations). Using the best current constructions, based on expander graphs, as stated in section 3, the depth of a $\text{SPLIT}_{\varepsilon/t}$ circuit is $O(\varepsilon/t \log \varepsilon/t)$ if we use a random construction, and $O((\varepsilon/t)^2)$ if we use a deterministic construction. A reasonable choice of parameters is $A = 3$, $\lambda = 1/8$, $\delta = 1/20$, $\mu\delta = 1/16$, $\varepsilon = 0.178$, $t = 4$. This yields a circuit of depth $65.16f \log N + O(1)$, where f is the depth of a $\text{SPLIT}_{1/22.48}$ circuit.

We have yet to show that when the algorithm is complete there are no errors. That is, in each leaf, the number of strangers is less than 1 (and hence there are no strangers in any leaf). But when the bucket capacity b is reduced to less than 16 the number of strangers is $\mu\delta b < 1$. Thus at termination a sort has been effected.

7. Effect of rounding error.

To accommodate rounding error, at each node we provide space for up to e *excess* items in addition to the bucket capacity. These excess items will always be at the correct node (that is, the number of items with strangeness at least s is bounded by $b\mu\delta^s$, where b is the bucket capacity). Recall that we used the following rule for distributing items at a nonroot node: Return the leftmost and rightmost $\lambda b/2$ items to the parent, and divide the other items evenly among the node's children. Clearly, we may be unable to achieve exactly this. Thus we use the following rules for distributing the items at a nonroot node v of capacity b, containing b′ items (by assumption $b' \le b + e$). We assume that v is a left child of its parent u (the rules for a right child are analogous).

1) If $b' \le \lambda b$ return all the items to the parent. Otherwise apply case 2 or 3, as appropriate.

2) If b' is even, return $\lfloor \lambda b/2 \rfloor$ items from each extreme to the parent.

3.1) If b' is odd and $2\lfloor \lambda b/2 \rfloor < \lfloor \lambda b \rfloor$ return $\lceil \lambda b/2 \rceil$ items from the right extreme and $\lceil \lambda b/2 \rceil - 1$ items from the left extreme to the parent.

3.2) If b is odd and $2\lfloor \lambda b/2 \rfloor = \lfloor \lambda b \rfloor$ return $\lfloor \lambda b/2 \rfloor$ items from the right extreme and $\lfloor \lambda b/2 \rfloor - 1$ items from the left extreme to the parent.

In both cases 2 and 3 the left (resp. right) half of the remaining items is passed to the left (resp. right) child.

The difference between λb (the number of items that ought to be passed to the parent, if we could have fractional items) and the number of items actually passed to the parent from node v is less than 2. Thus the number of excess items, e, passed to each child from v, is less than $e/2 + 1$; that is, $e < 2$.

We note that if we return $\lfloor \lambda b/2 \rfloor$ items from each extreme, assuming the SPLITs to have been without error, we would be returning all the strangers present, for fractional strangers cannot exist. Even in case 3, as we show, we would be returning all the strangers. For items with strangeness 1 belong in u, the parent of v. Thus items at the left extreme of v must have strangeness 2 or greater. There are at most $\lfloor \delta \lambda b/2 \rfloor = \lfloor 1/20 \cdot \lambda b/2 \rfloor$ such items; this is bounded by $\lfloor \lambda b/2 \rfloor - 1$ so long as $b \geq 16$. As we show next, for $b < 16$, b' must be a multiple of 4, and thus case 3 does not apply. A definition is helpful.

Definition: A node is a *root node* at the end of step T if it is a nonempty node and all its proper ancestors are empty.

Clearly, the number of items contained in a root node is a multiple of 4. Also, the root nodes are all at the same level in the tree.

Lemma. If a nonempty node has a bucket with capacity $b < 16$ then it is a root node.

Proof. We prove the result by induction on T, the number of sorting steps performed so far by the circuit. Clearly, the result is true after zero steps. So suppose that it is true after T steps; we show that it is also true after $T+1$ steps. Consider a root node r. Case 1: At the end of step T, r has capacity at least 16. Then at the end of step T its children have capacity at least $16A$, and at the end of step $T+1$ they have capacity $16A\nu \geq 16$. Thus if there is any nonempty node with capacity less than 16, at the end of step $T+1$, it must be at a level one higher than r in the tree; such a node is a root node. Case 2: At the end of step T, r has capacity less than 16. By assumption the grandchildren of r have capacity ≥ 16. At the end of step $T+1$ the children of r will be root nodes. Nonempty nodes at a lower level in the tree will have capacity at least $16A\nu \geq 16$; thus the result holds in this case too. •

Corollary. The number of items contained in a node with capacity $b < 16$ is a multiple of 4.

Next, we describe how to perform the $\text{CENTRIFUGE}_{\varepsilon,\lambda}$ operation so as to divide the items unevenly, as required. To do this, it will be necessary to generalize the SPLIT_ε operation. We show how to divide a set of size $r+s$, $r \leq s$, into two sets of sizes r and s, where the r items are, approximately, the smaller r items, and the s items are approximately the larger s items. We apply the SPLIT_ε operation to the following set of $2s$ items: the given items plus $s - r$ imaginary items of value $-\infty$ (a comparison

involving a $-\infty$ item is ignored). For an initial segment S of the input, the number of items not in the left half of the output is bounded by $\varepsilon(|S| + (s-r))$, while for a terminal segment S the number of items not in the right half of the output is bounded by $\varepsilon |S|$. An analogous construction is used for the case $r > s$. In both cases, to achieve an error of size $\varepsilon |S|$ for both initial and terminal segments S, we apply a $\text{SPLIT}_{\varepsilon'}$ operation, with $\varepsilon' = \dfrac{1}{1/\varepsilon + (s-r)}$.

We perform the partition of the items at a node using 4 levels of the generalized SPLIT operation. The error terms in Section 5 are affected only when a terminal segment is the shorter of the two segments resulting from the SPLIT. As we demonstrate below (by example), while the terminal segment may be shorter, it is at most 2 items shorter. So instead of performing a $\text{SPLIT}_{\varepsilon/t}$ operation we perform a $\text{SPLIT}_{\varepsilon'}$ operation with $\varepsilon' = \dfrac{1}{(t/\varepsilon) + 2} = 1/24.48$; then the error in the analysis of section 5 is unaffected. In particular, the fraction of nodes with strangeness $\geq s$ at each node obeys the bounds given in Section 5.

We demonstrate the partition by example. Suppose that there are $16r$ items present, but that the capacity of the node is less than $16r$. Then we will pass $r-1$ items from each extreme of the bucket to the parent. To achieve this we SPLIT the items into two sets of size $8r$. We now describe the SPLITs that are performed on the leftmost segment (analogous SPLITs are performed on the rightmost segment). We SPLIT the segment into a left piece of size $4r-1$ and a right segment of size $4r+1$. The left segment is SPLIT again, with the new left segment having size $2r-1$; it is SPLIT once more, with the new left segment having size $r-1$.

It remains to show that the skew term is not increased by the new distribution of items. But this is clear. For each node always passes either the right number of items to its parent or too few items. Thus the ancestors of a given node, between them, contain at most the number of items assumed by the analysis of Section 5. And, also, as shown above, the number of strangers at each node obeys the bounds given in Section 5.

We conclude

Theorem: There exists a sorting circuit of depth $65.16\ f \log N + O(1)$, where f is the depth of a $\text{SPLIT}_{1/24.48}$ circuit.

8. Alternative Approaches.

We first describe a minor improvement to the above method. We then describe a more complex version of the above construction. In neither case do we give all the details for only small reductions in the constant (in the running time) are achieved.

The minor improvement follows. Consider the $\text{CENTRIFUGE}_{\varepsilon,\lambda}$ operation applied to a set of b items. Its purpose is twofold.

(1) Obtain a small error term for each extreme segment of length $\leq \lambda b/2$.

(2) Obtain a small error term for the two extreme segments of length $b/2$.

We reduce the error in (2) without altering the depth of the circuit, as follows. Perform $t-1$ (= 3) $\text{SPLIT}_{\varepsilon/t}$ operations. This yields 8 segments. Combine the middle two segments and perform a $\text{SPLIT}_{\varepsilon/t}$ operations on the resulting $b/4$ items. Also perform $\text{SPLIT}_{\varepsilon/t}$ operations on each of the extreme segments. This will reduce the error term in (2) (roughly) from $\varepsilon b/2t$ to $\varepsilon b/8t + \varepsilon^2 b/t^2$ (actually, as in the remark in section 5, (2) is not exactly the term we are interested in). We leave to the interested reader the choice of a new δ, ε pair satisfying modified constraints [E], [G], [H], and [I].

The more complex version follows. We use the same basic structure as in section 5, but instead of passing the skew items to a child we keep them at the node. This appears intuitively reasonable, and in fact we do obtain a slightly better constant with this approach. However, as the constraints are considerably more complicated we leave the analysis of this approach to the interested reader.

9. References.

[AKS, 1983] M. Ajtai, J. Komlos, E. Szemeredi, ''An O(n log n) sorting network'', *Combinatorica*, 3(1983), 1-19.

[B, 1968] K. E. Batcher, ''Sorting Networks and their applications'', *Proc. AFIPS Spring Joint Summer Computer Conf.*, Vol.32, 307-314.

[IS, 1983] J. Incerpi and R. Sedgewick, ''Improved upper bounds on shellsort'', *Twenty Fourth Annual Symposium on Foundations of Computer Science*, 48-55.

[LPS, 1986] A. Lubotzky, R. Phillips, and P. Sarnak, ''Ramanujan conjecture and explicit constructions of expanders and super-concentrators'', *Eighteenth Annual Symposium on Theory of Computing*, p. 240-246.

[Pa, 1983] M. Paterson, public communication.

[Pi, 1986] N. Pippenger, public communication.