Comparator Networks for Binary Heap Construction*

Gerth Stølting Brodal†
Max-Planck-Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken, Germany

M. Cristina Pinotti‡
Istituto di Elaborazione della Informazione
CNR
56126 Pisa, Italy

January 21, 1998

Abstract

Comparator networks for constructing binary heaps of size n are presented which have size $\mathcal{O}(n \log\log n)$ and depth $\mathcal{O}(\log n)$. A lower bound of $n \log\log n - \mathcal{O}(n)$ for the size of any heap construction network is also proven, implying that the networks presented are within a constant factor of optimal. We give a tight relation between the leading constants in the size of selection networks and in the size of heap construction networks.

Introduction

The heap data structure, introduced in 1964 by Williams [17], has been extensively investigated in the literature due to its many applications and intriguing partial order. Algorithms for heap management — insertion, minimum deletion, and construction — have been discussed in several models of computation. For the heap construction algorithm, Floyd has given a sequential algorithm building the tree in a bottom-up fashion in linear time, which is clearly optimal. On the weak shared memory machine model, EREW-PRAM, Olariu and Wen can build a heap of size n in time $\mathcal{O}(\log n)$ and optimal work [14]. On the powerful

*This research was done while the first author was visiting the Istituto di Elaborazione della Informazione, CNR, Pisa.

†Supported by the Carlsberg foundation (Grant No. 96-0302/20). Partially supported by the ESPRIT Long Term Research Program of the EU under contract No. 20244 (ALCOM-IT). Email: brodal@mpi-sb.mpg.de.

‡Email: pinotti@iei.pi.cnr.it.

CREW-PRAM model, the best-known heap construction algorithm was given by Raman and Dietz and takes $\mathcal{O}(\log\log n)$ time [6]. The same time performance holds for the parallel comparison tree model [5]. Finally Dietz showed that $\mathcal{O}(\alpha(n))$, where $\alpha(n)$ is the inverse of Ackerman's function, is the expected time required to build a heap in the randomized parallel comparison tree model [5]. All the above parallel algorithms achieve optimal work $\mathcal{O}(n)$, and the time optimality of the deterministic algorithms can be argued by reduction from the selection of the minimum element in a set.

In this paper we address the heap construction problem for the simplest parallel model of computation, namely *comparator networks*. Comparator networks perform only comparison operations, which may occur simultaneously. The most studied comparator networks are sorting and merging networks. In the early 1960's, Batcher proposed the odd-even merge algorithm to merge two sequences of n and m elements, $n \geq m$, which can be implemented by a merging network of size $\mathcal{O}((m+n)\log m)$. In the early 1970's Floyd [12] and Yao [18] proved the asymptotic optimality of Batcher's networks. The lower bound has recently been improved by Miltersen, Paterson and Tarui [13], closing the long-standing factor-of-two gap between upper and lower bounds. It is noteworthy to recall, that merge can be solved in the comparison tree model with a tree of depth $m+n-1$.

Batcher also showed how his merge algorithm could be used to implement sorting networks with size $\mathcal{O}(n\log^2 n)$ and depth $\mathcal{O}(\log^2 n)$ to sort n inputs [12]. For a long time, the question remained open as to whether sorting networks with size $\mathcal{O}(n\log n)$ and depth $\mathcal{O}(\log n)$ existed. In 1983, Ajtai, Komlós and Szemerédi [1] presented sorting networks with size $\mathcal{O}(n\log n)$ and depth $\mathcal{O}(\log n)$ to sort n items. This result, although partially unsatisfying due to big constants hidden by the $\mathcal{O}$-notation, reveals that the sorting problem requires the same amount of work in both comparison tree and comparator network models.

Selection, sorting and merging are strictly related problems. Several sequential algorithms with linear work have been discussed for selection. The first is due to Blum *et al.* [4] and requires $5.43n$ comparisons. This result was later improved by Schönhage *et al.* to $3n$ [16] and by Dor and Zwick to $2.95n$ [7, 8]. Bent and John proved a lower bound of $2n$ for this problem [3]. Dor and Zwick [9] improved it to $(2+\epsilon)n$ [9]. For a survey of previous work on lower bounds in the comparison tree model, see the paper by Dor and Zwick [9].

For comparator networks Alekseyev [2] proved that an (n,t)-selection network, which selects the t smallest item in a set of n elements, has at least size $(n-t)\lceil\log(t+1)\rceil$.[1] For $t=\Omega(n^\epsilon)$ and $0<\epsilon\leq 1$, the existence of a work optimal selection network immediately follows by the sorting networks of Ajtai *et al.*. However, since selection networks do not need to do as much as sorting networks, and due to the big constant hidden by the sorting networks in [1], selection networks with improved constant factors in both depth and size have been developed. In particular, Pippenger proposes a $(n,\lfloor n/2\rfloor)$-selection network with size $2n\log n$ and depth $\mathcal{O}(\log^2 n)$ [15]. More recently, Jimbo and Marouka have constructed a $(n,\lfloor n/2\rfloor)$-selection network of depth $\mathcal{O}(\log n)$ and of size at most $Cn\log n+\mathcal{O}(n)$, for any arbitrary $C>3/\log 3\approx 1.89$, which improves Pippenger's construction by a constant factor in size and at the same time by an order in depth [11].

The preceding summary shows that work optimal comparator networks have been studied for merging, sorting, and selection. Although the heap data structure has historically been

[1]All logarithms throughout this paper have basis 2

strictly related to these problems, we are not aware of any comparator network for the heap construction problem. In this scenario, we show that heap construction can be done by comparator networks of size $\mathcal{O}(n \log\log n)$ and depth $\mathcal{O}(\log n)$, and that our networks reach optimal size by reducing the problem of selecting the smallest $\log n$ elements to heap construction. Finally, since finding the minimum requires at least a network of size $n-1$ and depth $\lceil \log n \rceil$, our heap construction networks also have optimal depth.

1 Preliminaries

Let us review some definitions, and agree on some notations used throughout the paper.

A *binary tree* of size n is a tree with n nodes, each of degree at most two. A node x of a binary tree belongs to level k if the longest simple path from the root to x has k edges. The height of the tree is the number of edges in the longest simple path starting at the root of the tree. The subtree T_x rooted at node x at *level* k is the tree induced by the descendants of x.

A *complete binary tree* is a binary tree in which all the leaves are at the same level and all the internal nodes have degree two. Clearly, it has *height* $\lfloor \log n \rfloor$.

A *heap shaped binary* tree of height h is a binary tree whose $h-1$ uppermost levels are completed filled and the h-th level is filled from the left to the right.

In a *heap ordered* binary tree, each node contains one element which is greater or equal to the element at its parent.

Finally, a *binary heap* is defined as a heap-shaped and heap-ordered binary tree [17], which can be stored in an array H as an implicit tree of size n, as depicted in Figure 1. The element of the root of the tree is at index 1 of the array, (*i.e.*, root is stored in $H[1]$), and given an index i of a node x, the indices of its left and right children are $2i$ and $2i+1$, respectively.

A *comparator network* with n inputs and size s is a collection of n horizontal lines, one for each input, and s comparators. A *comparator* between line i and j, briefly $i:j$, compares the current values on lines i and j and is drawn as a vertical line connecting lines i and j. After the comparison $i:j$, the minimum value is put on line i, while the maximum ends up on line j. Finally, a comparator network has *depth* d, if d is the largest number of comparators that any input element can pass through. Assuming that each comparator produces its output in constant time, the *depth* of a comparator network is the running time of such a network. From now on, let us refer to comparator networks simply as networks. For a comprehensive account of comparator networks, see [12, pp. 220-246].

2 Sequential heap construction

It is well known that an implicit representation of a binary heap H of size n can be built in linear sequential time by the heap construction algorithm of Floyd [10]. Because we base our heap construction networks on Floyd's algorithm, we rephrase it as follows:

Assuming that the two binary trees rooted at the children of a node i are heaps, the heap-order property in the subheap rooted at i can be reestablished simply by bubbling

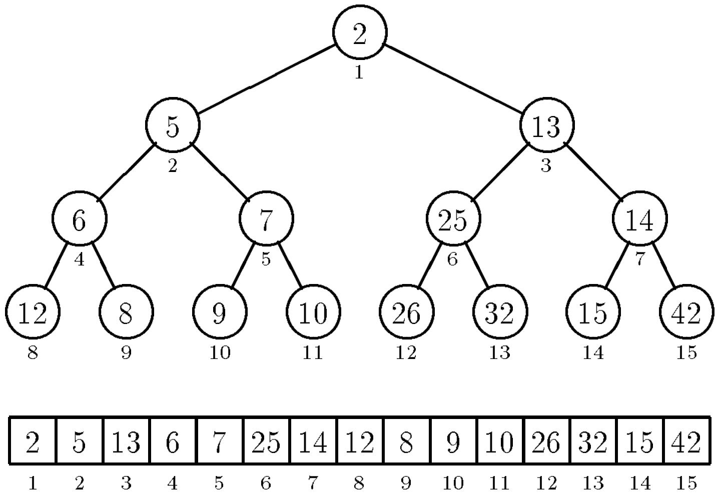

Figure 1: A binary heap of size 15 and its implicit representation.

down the element $H[i]$. We let the bubbling down procedure be denoted Siftdown. At each step, Siftdown determines the smallest of the elements $H[i], H[2i]$, and $H[2i+1]$. If $H[i]$ is the smallest, then the subtree rooted at node i is a heap and the Siftdown procedure terminates. Otherwise, the child with the smallest element and $H[i]$ are exchanged. The node exchanged with $H[i]$, however, may violate the heap order at this point. Therefore, the Siftdown procedure is recursively invoked on that subtree.

We can now apply Siftdown in a bottom-up manner to convert an array H storing n elements into a heap. Since the elements in the subarray $H[(\lfloor n/2 \rfloor + 1)..n]$ are all leaves, each is a 1-element heap to begin with. Then, the remaining nodes of the tree are visited to run the Siftdown procedure on each one. Since the nodes are processed level by level in a bottom up fashion, it is guaranteed that the subtrees rooted at the children of the node i are heaps before Siftdown runs at that node.

In conclusion, observe that the Siftdown routine invoked on a subheap of height i performs $2i$ comparisons in the worst case, and that the worst case running time of the heap construction algorithm of Floyd described above is $\sum_{i=0}^{\lfloor \log n \rfloor} \frac{n}{2^i} \cdot 2i = \mathcal{O}(n)$, which is optimal.

3 Heap construction networks of size $n \log n$

In this section we present heap construction networks which have size at most $n\lfloor \log n \rfloor$ and depth $4\lfloor \log n \rfloor - 2$. Notice that any sorting network could also be used as a heap construction network. The networks presented in this section are used in Section 4 to construct improved heap construction networks of size $\mathcal{O}(n \log \log n)$, and in Section 5 to give a reduction from selection to heap construction.

Lemma 1 gives a network implementation of the sifting down algorithm used in the heap construction algorithm by Floyd [10].

Lemma 1 *Let T be a binary tree of size n and height h. If the subtrees rooted at the children of the root satisfy heap order, then the elements of T can be rearranged to satisfy heap order with a network of size $n-1$ and depth $2h$. At depth $2i+1$ and $2i+2$ of the network the comparators are only between nodes at level i and $i+1$ in T. All comparators correspond to edges of T, and for each edge there is exactly one comparator.*

Proof. If the tree has height zero, no comparator is required. Otherwise let r be the root and u and v the children of r. If u or v is not present, the steps below which would involve v or u are skipped.

First we apply the comparators $r : u$ and $r : v$. Because T_u and T_v were assumed to be heap ordered subtrees, r now has the minimum. After the two comparators the heap order can be violated at the roots of *both* T_u and T_v. We therefore recursively apply the above to the subtrees T_u and T_v. Notice that the two recursively constructed networks involve disjoint nodes and therefore can be performed in parallel. If r only has one child we still charge the network depth two to compare r with its children to guarantee that all comparisons done in parallel by the network correspond to edges between nodes at the same levels in T.

The depth of the network is two plus the depth of the deepest recursively constructed network. By induction it follows that the depth of the network is $2h$, and that the network at depth $2i+1$ and $2i+2$ only performs comparisons between nodes at level i and $i+1$ in T. Furthermore, the network contains exactly one comparator for each edge of T. $\square$

Notice that the network has $n-1$ comparators while the corresponding algorithm of Floyd only needs h comparisons. By replacing the sifting down algorithm in Floyd's heap construction algorithm by the sifting down networks of Lemma 1, we get the following lemma.

Lemma 2 *Let T be a binary tree of size n and height h which does not satisfy heap order, and let n_i be the number of nodes at level i in T. Then a network exists of size $\sum_{i=0}^{h} i \cdot n_i$ and depth $4h-2$ which rearranges the elements of T to satisfy heap order. All comparators correspond to edges of T.*

Proof. Initially all nodes at level h of T by definition are heap ordered binary trees of height zero. Iteratively for each level $i = h-1, \ldots, 0$ we apply the sifting down networks of Lemma 1 in parallel to the 2^i subtrees rooted at level i of T, to make these subtrees satisfy heap order. The resulting tree then satisfies heap order. By Lemma 1 all comparators correspond to edges of T.

The edge between a node v at level i and its parent corresponds to a set of comparators in the resulting network. These comparators are performed exactly when we apply the sifting down networks of Lemma 1 to an ancestor of v, *i.e.*, there are exactly i comparators corresponding to this edge. The total number of comparators is $\sum_{i=0}^{h} i \cdot n_i$.

By Lemma 1 the depth of the network is $\sum_{i=0}^{h} 2i = h^2 + h$. But because the networks constructed by Lemma 1 proceeds top-down on T, having exactly depth two for each level of T, the applications of Lemma 1 can be *pipelined*. After the first two comparators of the applications of Lemma 1 to subtrees rooted at level i, the applications of Lemma 1 to subtrees rooted at level $i-1$ can be initiated. The application of Lemma 1 to the root of the tree can therefore be initiated at depth $2(h-1)+1$ of the network, *i.e.*, the network has depth $2(h-1) + 2h = 4h-2$. $\square$

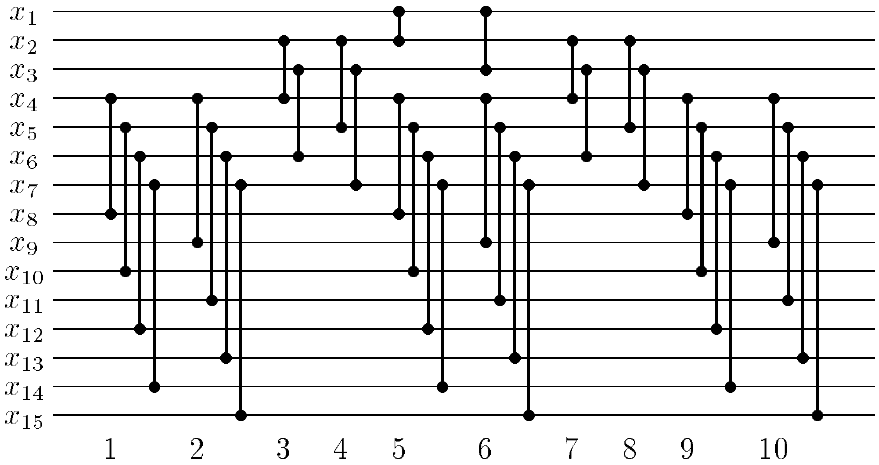

Figure 2: A heap construction network for $n = 15$. All comparators are of the form $i : j$, where $i < j$.

Theorem 3 *There exists a heap construction network of size at most $n\lfloor \log n \rfloor$ and depth $4\lfloor \log n \rfloor - 2$. All comparators correspond to edges of T.*

Proof. Let the n input lines represent a heap shaped binary tree of height $\lfloor \log n \rfloor$. The theorem then follows from Lemma 2. $\square$

In Figure 2 we show the network of Theorem 3 for $n = 15$. The network has size 34 and depth 10. Notice that the first two comparators of the application of Lemma 1 to the root of the tree ($1 : 2$ and $1 : 3$) are done in parallel with the third and fourth comparator of the applications of Lemma 1 to the subtrees rooted at nodes 2 and 3.

4 Heap construction networks of size $\mathcal{O}(n \log \log n)$

In the following we give improved heap construction networks of size $\mathcal{O}(n \log \log n)$ and depth $\mathcal{O}(\log n)$. The improved networks are obtained by combining the networks of Theorem 3 with efficient selection networks.

An arbitrary sorting network is obviously also an (n, t)-selection network, *e.g.*, the sorting network of size $\mathcal{O}(n \log n)$ by Ajtai *et al.* [1]. Due to the large constants involved in the sorting network of Ajtai *et al.*, Pippenger [15] and Jimbo and Maruoka [11] have developed specialized $(n, \lfloor n/2 \rfloor)$-selection networks of size $\mathcal{O}(n \log n)$ where the involved constants are of reasonable size. The following lemma was developed by Jimbo and Maruoka [11].

Lemma 4 (Jimbo and Maruoka) *For an arbitrary constant $C > 3/\log 3 \approx 1.89$, there exist $(n, \lfloor n/2 \rfloor)$-selection networks of size at most $Cn \log n + \mathcal{O}(n)$ and depth $\mathcal{O}(\log n)$.*

Unfortunately, neither Pippenger [15] or Jimbo and Maruoka [11] state bounds for general (n, t)-selection networks. The following lemma is a consequence of Lemma 4, and is sufficient for our purposes.

Lemma 5 *For an arbitrary constant $C > 6/\log 3 \approx 3.79$, there exist (n,t)-selection networks of size $Cn\log t + \mathcal{O}(n)$ and depth $\mathcal{O}(\log n \cdot \log t)$.*

Proof. The n input lines are partitioned into $\lceil n/t \rceil$ blocks $B_1, \ldots, B_{\lceil n/t \rceil}$ of size t each. By applying the selection networks of Lemma 4 to $B_1 \cup B_2$ we find the t least elements of $B_1 \cup B_2$. By combining the $\lceil n/t \rceil$ blocks in a treewise fashion with $\lceil n/t \rceil - 1$ applications of Lemma 4 to $2t$ elements, we find the t least elements of the n inputs. The resulting network has size $(\lceil n/t \rceil - 1)(C \cdot 2t \log 2t + \mathcal{O}(2t)) = 2Cn\log t + \mathcal{O}(n)$ and depth $\mathcal{O}(\log n \cdot \log t)$, for $C > 3/\log 3$. $\square$

We need the following definition. Let $\mathcal{P}$ be an arbitrary connected subset of nodes of a binary tree T which contains the root of T. Let $x_1 \leq x_2 \leq \cdots \leq x_{|\mathcal{P}|}$ be the set of elements in $\mathcal{P}$, and let $x'_1 \leq x'_2 \leq \cdots \leq x'_{|\mathcal{P}|}$ be the set of elements in $\mathcal{P}$ after applying a network $\mathcal{N}$ to T. We define a network $\mathcal{N}$ to be *heap-convergent*, if $\mathcal{N}$ for all possible inputs, all connected subsets $\mathcal{P}$ of nodes of T containing the root of T, and $i = 1, \ldots, |\mathcal{P}|$ satisfies $x'_i \leq x_i$. Notice that sorting networks are *not* heap-convergent. If $\mathcal{P}$ is the path to the rightmost node in the lowest level of a tree, then $\mathcal{P}$ always contains the maximum element after applying a sorting network, but the maximum element could initially be anywhere in the tree.

Lemma 6 *A comparator corresponding to an edge in a binary tree T is a heap-convergent network.*

Proof. Let the comparator be $u : v$, where v is a child of u in T. If $\mathcal{P}$ does not contain u it does not contain v either, implying that the elements in $\mathcal{P}$ are unchanged. If $\mathcal{P}$ contains both u and v, the set of elements is also unchanged. If $\mathcal{P}$ contains u but not v, the comparator $u : v$ can only replace the element at u with a smaller element from v in which case $x'_i \leq x_i$ for all $i = 1, \ldots, |\mathcal{P}|$. $\square$

Because the networks constructed by Theorem 3 only contain comparators corresponding to tree edges and heap convergence is a transitive property we immediately have the following corollary:

Corollary 7 *The networks constructed by Theorem 3 are heap-convergent.*

Theorem 8 *If for some constants C and d, there exist (n,t)-selection networks of size $Cn\log t + \mathcal{O}(n)$ and depth $\mathcal{O}(\log^d n)$, then there exist heap construction networks of size $Cn \log\log n + \mathcal{O}(n \log\log\log n)$ and depth $4\log n + \mathcal{O}(\log^d \log n)$.*

Proof. Assume without loss of generality that $n \geq 4$. Let the n input lines represent a heap shaped binary tree T of height $h = \lfloor \log n \rfloor$, and let $k = \lceil \log h \rceil \geq 1$. The heap construction network proceeds in three phases.

1. To each subtree T_v rooted at level $h - 2k + 1$, apply in parallel $(|T_v|, 2^k - 1)$-selection networks, such that all elements at the upper k levels of T_v become less than or equal to all elements at the remaining levels of T_v.
2. Apply the heap construction networks of Theorem 3 to the uppermost $h - k$ levels of T.

3. In parallel apply Theorem 3 to each subtree T_v rooted at level $h - 2k + 1$.

It follows immediately from Step 2 that the uppermost $h - 2k$ levels of the tree satisfy heap order and from Step 3 that each subtree rooted at level $h - 2k + 1$ satisfies heap order. What remains to be proven for the correctness of the algorithm is that for all nodes v at level $h - 2k + 1$, the subtree T_v only contains elements which are greater or equal to the elements on the path from the root to v.

After Step 1, the $2^k - 1$ least elements $e_0 \leq \cdots \leq e_{2^k-2}$ of T_v are at the uppermost k levels of T_v, which are exactly the levels of T_v which overlap with Step 2. Let $p_0 \leq \cdots \leq p_{h-2k}$ denote the elements on the path from the root to v (excluding v) after Step 2. Because the network applied in Step 2 is heap-convergent and $2^k - 2 \geq h - 2k$, we have $p_i \leq e_i$ for $i = 0, \ldots, h - 2k$ by letting $\mathcal{P}$ consist of the path from the root to v together with the upper k levels of T_v. We conclude that after Step 2 all elements on the path from the root to v are smaller than or equal to all the elements in T_v, and that after Step 3, T satisfies heap order.

From Theorem 3 we get the following upper bound on the size and depth of the resulting network. The size is bounded by

$$\left(Cn \log 2^k + \mathcal{O}(n)\right) + \mathcal{O}\left(\frac{n}{2^k} \log \frac{n}{2^k}\right) + \left(n \log 2^{2k} + \mathcal{O}(n)\right) ,$$

which is $(C + 2)n \log \log n + \mathcal{O}(n)$, and the depth is bounded by

$$\mathcal{O}\left(\log^d 2^{2k}\right) + (4(h - k) - 2) + (4(2k - 1) - 2) ,$$

which is $4 \log n + \mathcal{O}(\log^d \log n)$.

The "+2" in the size bound comes from the application of the heap construction networks of Theorem 3 in Step 3. If we instead apply the above construction recursively in Step 3, we get heap construction networks of size $Cn \log \log n + (C + 2)n \log \log \log n + \mathcal{O}(n)$ and depth $4 \log n + \mathcal{O}(\log^d \log n)$. $\square$

Notice that in Steps 1 and 3 we could have used arbitrary sorting networks, but in Step 2 it is essential that the heap construction network used is heap-convergent. By applying the construction recursively $\mathcal{O}(\log^* n)$ times the asymptotic size could be slightly improved, but the constant in front of $n \log \log n$ would still be C. From Lemma 5 we get the following corollary:

Corollary 9 *For an arbitrary constant $C > 6/\log 3 \approx 3.79$, there exist heap construction networks of size $Cn \log \log n + \mathcal{O}(n \log \log \log n)$ and depth $4 \log n + \mathcal{O}(\log^2 \log n)$.*

5 A lower bound for the size of heap construction networks

We now prove that the construction of the previous section is optimal. Let $S(n, t)$ denote the minimal size of (n, t)-selection networks, and let $H(n)$ denote the minimal size of heap construction networks on n inputs. The following lower bound on $S(n, t)$ is due to Alekseyev [2].

Lemma 10 (Alekseyev) $S(n,t) \geq (n-t)\lceil \log(t+1) \rceil$.

Theorem 11 $H(n) \geq S(n, \lfloor \log n \rfloor) - \mathcal{O}(n)$.

Proof. The theorem is proven by giving a reduction from (n,t)-selection to heap construction. We prove that (n,t)-selection can be done by networks with size $H(n) + 2^{t+1} - 2t - 2$.

First we construct a heap over the n inputs with a network of size $H(n)$, and make the observation that the t least elements can only be at levels $0, \ldots, t-1$ of the heap.

The minimum is at the root, *i.e.*, at output line one. To find the second least element we consider the implicit heap given by the lines $n, 2, 3, \ldots, 2^t - 1$. Notice that the root is now line n. By applying the sifting down network of Lemma 1 to the levels $0, \ldots, t-1$ of this tree the remaining $t-1$ least inputs are at levels $0, \ldots, t-2$ of this tree. The second least element is now at output line n. By iteratively letting the root be lines $n-1, n-2, \ldots, n-t-2$, and by applying Lemma 1 to trees of decreasing height, the t least elements will appear in sorted order at output lines $1, n, n-1, n-2, \ldots, n-t+2$. If the t smallest inputs are required to appear at the first t output lines, the network lines are permuted accordingly.

The total number of comparators for the $t-1$ applications of Lemma 1 is

$$\sum_{i=0}^{t-1} (2^{i+1} - 2) = 2^{t+1} - 2t - 2 \ .$$

We conclude that the resulting (n,t)-selection network has size $H(n) + 2^{t+1} - 2t - 2$, implying $H(n) \geq S(n,t) - 2^{t+1} + 2t + 2$. By letting $t = \lfloor \log n \rfloor$ the theorem follows. □

By combining Lemma 10 and Theorem 11, we get the following corollary.

Corollary 12 $H(n) \geq n \log\log n - \mathcal{O}(n)$.

6 Conclusion

The parallel construction of heaps has been addressed for several parallel models of computation: EREW-PRAM [14], CRCW-PRAM [6], the parallel comparison tree model and the randomized parallel comparison tree model [5]. These algorithms all achieve optimal $\mathcal{O}(n)$ work. In this paper we have addressed the problem for the most simple parallel model of computation, namely comparator networks.

Opposed to merging and selection, which both can be solved in sequential linear time but require networks of size $\Theta(n \log n)$, we have shown that heap construction can be done by networks of size $\mathcal{O}(n \log\log n)$ and depth $\mathcal{O}(\log n)$, and that this is optimal. By combining the results of Theorem 8 and Theorem 11, we get the following characterization of the leading constant in the size of heap construction networks compared to the leading constant in the size of (n,t)-selection networks.

Theorem 13 *If for constants C_1 and C_2,*

$$C_1 n \log t - \mathcal{O}(n) \leq S(n,t) \leq C_2 n \log t + \mathcal{O}(n) \ ,$$

then

$$C_1 n \log\log n - \mathcal{O}(n) \leq H(n) \leq C_2 n \log\log n + \mathcal{O}(n \log\log\log n) \ .$$

Acknowledgment

Thanks to Peter Sanders for his comments on an earlier draft of this paper.

References

[1] Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19, 1983.

[2] Vladimir Evgen'evich Alekseyev. Sorting algorithms with minimum memory. *Kibernetika*, 5(5):99–103, 1969.

[3] Samuel W. Bent and John W. John. Finding the median requires $2n$ comparisons. In *Proc. 17th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 213–216, 1985.

[4] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.

[5] Paul F. Dietz. Heap construction in the parallel comparison tree model. In *Proc. 3rd Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 621 of *Lecture Notes in Computer Science*, pages 140–150. Springer Verlag, Berlin, 1992.

[6] Paul F. Dietz and Rajeev Raman. Very fast optimal parallel algorithms for heap construction. In *Proc. 6th Symposium on Parallel and Distributed Processing*, pages 514–521, 1994.

[7] Dorit Dor and Uri Zwick. Selecting the median. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 28–37, 1995.

[8] Dorit Dor and Uri Zwick. Finding the alpha n-th largest element. *Combinatorica*, 16:41–58, 1996.

[9] Dorit Dor and Uri Zwick. Median selection requires $(2+\epsilon)n$ comparisons. In *Proc. 37th Ann. Symp. on Foundations of Computer Science (FOCS)*, pages 125–134, 1996.

[10] Robert W. Floyd. Algorithm 245: Treesort3. *Communications of the ACM*, 7(12):701, 1964.

[11] Shuji Jimbo and Akira Maruoka. A method of constructing selection networks with $O(\log n)$ depth. *SIAM Journal of Computing*, 25(4):709–739, 1996.

[12] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.

[13] Peter Bro Miltersen, Mike Paterson, and Jun Tarui. The asymptotic complexity of merging networks. *Journal of the ACM*, 43(1):147–165, 1996.

[14] Stephan Olariu and Zhaofang Wen. Optimal parallel initialization algorithms for a class of priority queues. *IEEE Transactions on Parallel and Distributed Systems*, 2:423–429, 1991.

[15] Nicholas Pippenger. Selection networks. *SIAM Journal of Computing*, 20(5):878–887, 1991.

[16] Arnold Schönhage, Michael S. Paterson, and Nicholas Pippenger. Finding the median. *Journal of Computer and System Sciences*, 13:184–199, 1976.

[17] John William Joseph Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, 1964.

[18] Andrew C. Yao and Frances F. Yao. Lower bounds on merging networks. *Journal of the ACM*, 23:566–571, 1976.