A Tight Lower Bound for Top-Down Skew Heaps*

Berry Schoenmakers†

January, 1997

Abstract

Previously, it was shown in a paper by Kaldewaij and Schoenmakers that for top-down skew heaps the amortized number of comparisons required for meld and delmin is upper bounded by $\log_\phi n$, where n is the total size of the inputs to these operations and $\phi = (\sqrt{5}+1)/2$ denotes the golden ratio. In this paper we present worst-case sequences of operations on top-down skew heaps in which each application of meld and delmin requires approximately $\log_\phi n$ comparisons. As the remaining heap operations require no comparisons, it then follows that the set of bounds is tight. The result relies on a particular class of self-recreating binary trees, which is related to a sequence known as Hofstadter's G-sequence.

1 Introduction

Top-down skew heaps are probably the simplest implementation of *mergeable* priority queues to date while still achieving good performance. As with other so-called self-adjusting data structures the catch is that the performance is merely good in the amortized sense, but in many applications this is perfectly acceptable. Figure 1 displays a purely functional version of top-down skew heaps, which is based on the original version of Sleator and Tarjan, the inventors of skew heaps [9]. Compared to the set of programs described in [9], however, we use operation single instead of an insert operation (note that insertion of a into heap x can be achieved as meld.(single.a).x).

The efficiency of skew heaps is entirely due to the particular way operation meld is defined. Informally the effect of meld.x.y can be described by two steps. First, the rightmost paths of trees x and y are merged, where the left subtrees of the nodes on the merge path stick to their nodes. Second, the left and right subtrees are swapped for every node on the merge path. Intuitively, the second step turns the potentially long merge path, which is a rightmost path, into a leftmost path of the resulting heap. In actual implementations it is worthwhile to program meld performing a single pass over the rightmost paths, while at the same time building up the leftmost path (see [9]). As shown in [3], it is then possible to get a simple implementation of mergeable priority queues that permits an interesting degree of concurrency.

In [4, 8] the following upper bounds have been proven for the amortized costs of the operations in terms of comparisons (each unfolding of meld.x.y requires one comparison if

*Appears in *Information Processing Letters* 61(5) 279–284, March 14, 1997

†DigiCash, Kruislaan 419, 1098 VA Amsterdam, Netherlands. berry@digicash.com

$$
\begin{array}{lcll}
\mathsf{empty} & = & \langle\,\rangle & \\
\mathsf{isempty}.x & = & x = \langle\,\rangle & \\
\mathsf{single}.a & = & \langle\langle\,\rangle, a, \langle\,\rangle\rangle & \\
\mathsf{min}.\langle t, a, u\rangle & = & a & \\
\mathsf{delmin}.\langle t, a, u\rangle & = & \mathsf{meld}.t.u & \\
\mathsf{meld}.\langle\,\rangle.y & = & y & \\
\mathsf{meld}.x.\langle\,\rangle & = & x & \\
\mathsf{meld}.\langle t, a, u\rangle.y & = & \langle \mathsf{meld}.u.y, a, t\rangle, & a \leq \mathsf{min}.y \\
\mathsf{meld}.x.\langle t, a, u\rangle & = & \langle \mathsf{meld}.x.u, a, t\rangle, & a \leq \mathsf{min}.x
\end{array}
$$

Figure 1: Purely functional top-down skew heaps, where $\langle\,\rangle$ denotes the empty tree and $\langle t, a, u\rangle$ denotes a tree with left subtree t, root a, and right subtree u.

x and y are both nonempty). These bounds improve upon the original bounds of Sleator and Tarjan [9] by more than a factor of two. As explained in [8, Chapter 5], these bounds do only hold for functional programs that restrict the use of skew heaps to "linear usage" as if operations delmin and meld were destructive (e.g., using both $\mathsf{delmin}.x$ and $\mathsf{meld}.x.y$ in the same expression is not allowed). It is interesting to note that Okasaki has been able to remove this restriction of linear usage for many purely-functional data structures by making judicious use of lazy evaluation (see [6, 7]).

Theorem 1 *(cf. [8, Lemma 9.2]) There exists a potential function such that the amortized costs for top-down skew heaps satisfy (in terms of comparisons):* empty, isempty.x, *and* min.x *cost 0,* single.a *costs at most 1,* delmin.x *costs at most* $\log_\phi |x|$, *and* meld.$x.y$ *costs at most* $\log_\phi(|x| + |y|)$, *where* $\phi = (\sqrt{5}+1)/2$ *denotes the golden ratio.*

Here, we have used $|x|$ to denote the size of tree x, which is defined equal to one plus the number of nodes of x. A recursive definition is given by: $|\langle\,\rangle| = 1$ and $|\langle t, a, u\rangle| = |t| + |u|$.

In this paper we will show that the above set of bounds is in fact tight. We do so by presenting worst-case sequences of operations on top-down skew heaps in which the actual cost of each operation matches the alloted amortized cost of Theorem 1. Clearly, meld forms the central operation on skew heaps. In the next section we first consider a special version of meld that operates on *unlabelled* binary trees. This special version takes maximal time for a particular class of trees. In the subsequent section we then show that these cases actually arise in applications of skew heaps, which implies that the bounds of Theorem 1 are tight. Our methods resemble the methods used to obtain lower bounds on the amortized complexity of union-find data structures (see, e.g., [1, 11, 12]), in which finding a suitable class of self-recreating (or, self-reproducing) trees also constitutes an important part of the solution. Throughout the analysis we find it instrumental to use a functional notation.

2 Unlabelled case

We consider the following operation $\bowtie$ on unlabelled binary trees, which is strongly related to operation meld:

$$\begin{aligned} \langle\,\rangle \bowtie \langle\,\rangle &= \langle\,\rangle \\ \langle t, u\rangle \bowtie y &= \langle y \bowtie u, t\rangle. \end{aligned}$$

As part of operation $x \bowtie y$ the rightmost paths of x and y are traversed in alternating order starting with x. Note that for each application of $x \bowtie y$ we need that $x \neq \langle\,\rangle$ if $y \neq \langle\,\rangle$, which is ensured if $|x| \geq |y|$. This will be the case.

The goal of the analysis is to define a sequence of trees for which $\bowtie$ is expensive, while the resulting tree is again an element of the sequence. In light of Theorem 1 the cost of $x \bowtie y$ (which is defined as the sum of the lengths of the rightmost paths of x and y) should be close to $\log_\phi(|x| + |y|)$. It will be no surprise that the Fibonacci sequence plays an important role in our construction.

Define the Fibonacci sequence F_k, $k \geq 0$, as usual by $F_0 = 0$, $F_1 = 1$, and $F_k = F_{k-1} + F_{k-2}$, $k \geq 2$. Next define two related functions $L(n)$ and $R(n)$, $n \geq 0$, as follows:[1]

$$\begin{aligned} &L(0) = R(0) = 0 \\ &L(n) = F_{k-1} + L(n - F_k) \\ &R(n) = F_{k-2} + R(n - F_k), \quad n \geq 1, \end{aligned}$$

where k is uniquely determined by $F_k \leq n < F_{k+1}$. Hence $k \geq 2$. As an alternative characterisation of L and R we will often use the next two lemmas.

Lemma 1 *$L(F_k + a) = F_{k-1} + L(a)$, for $0 \leq a \leq F_{k-1}$ and $k \geq 1$.*

Lemma 2 *$R(F_k + a) = F_{k-2} + R(a)$, for $0 \leq a \leq F_{k-1}$ and $k \geq 2$.*

A few simple properties of L and R are given by the next lemma.

Lemma 3 *$L(n) + R(n) = n$, $L(n) \geq R(n)$, $L(n+1) \geq L(n)$, and $R(n+1) \geq R(n)$, for $n \geq 0$.*

Next we prove the three main lemmas we need.

Lemma 4 *$L(L(n-1)) = R(n)$, for $n \geq 1$.*

Proof By induction on n. If $n = 1$ both sides are zero. For $n \geq 2$, let k denote the unique integer satisfying $F_k < n \leq F_{k+1}$ (hence $k \geq 2$). Then we have:

$$\begin{aligned} & L(L(n-1)) \\ = & \quad \{ \text{ definition } L \ \} \\ & L(F_{k-1} + L(n - 1 - F_k)) \\ = & \quad \{ \text{ Lemma 1, using } 0 \leq L(n - 1 - F_k) \leq F_{k-2} \ \} \\ & F_{k-2} + L(L(n - 1 - F_k)) \\ = & \quad \{ \text{ induction hypothesis } \} \\ & F_{k-2} + R(n - F_k) \\ = & \quad \{ \text{ Lemma 2, using } 0 \leq n - F_k \leq F_{k-1} \ \} \\ & R(n). \end{aligned}$$

[1] Through the on-line version of Sloane's "Encyclopedia of Integer Sequences" [10], we found out that we had rediscovered Hofstadter's G-sequence [2, p.137], since function $L(n)$ satisfies $L(0) = 0$ and $L(n) = n - L(L(n-1))$, $n \geq 1$ (use Lemmas 3 and 4).

□

Lemma 5 *$L(L(n)-1) = R(n-1)$, for $n \geq 1$.*
Proof By induction on n. If $n = 1$ both sides are zero. For $n \geq 2$, let k denote the unique integer satisfying $F_k < n \leq F_{k+1}$ (hence $k \geq 2$). Then we have:

$$
\begin{aligned}
& L(L(n)-1) \\
= & \quad \{ \text{ Lemma 1, using } 1 \leq n - F_k \leq F_{k-1} \} \\
& L(F_{k-1} + L(n-F_k) - 1) \\
= & \quad \{ \text{ Lemma 1, using } 0 \leq L(n-F_k) - 1 \leq F_{k-2} \} \\
& F_{k-2} + L(L(n-F_k) - 1) \\
= & \quad \{ \text{ induction hypothesis } \} \\
& F_{k-2} + R(n - F_k - 1) \\
= & \quad \{ \text{ definition } R \} \\
& R(n-1).
\end{aligned}
$$

□

Lemma 6 *$R(L(n)-1) = R(L(n-1))$, for $n \geq 1$.*
Proof For any $n \geq 1$, we have:

$$
\begin{aligned}
& R(L(n)-1) - R(L(n-1)) \\
= & \quad \{ \text{ Lemma 3 (twice) } \} \\
& L(n) - 1 - L(L(n)-1) - L(n-1) + L(L(n-1)) \\
= & \quad \{ \text{ Lemmas 5, 4, resp. } \} \\
& L(n) - 1 - R(n-1) - L(n-1) + R(n) \\
= & \quad \{ \text{ Lemma 3 (twice) } \} \\
& 0 \,.
\end{aligned}
$$

□

Given functions L and R we now define a sequence of unlabelled binary trees G_n, $n \geq 0$, as follows:

$$
\begin{aligned}
& G_0 = \langle\,\rangle \\
& G_n = \langle G_{L(n-1)}, G_{R(n-1)} \rangle, \quad n \geq 1.
\end{aligned}
$$

We dub these trees "golden trees" because the ratio of the sizes of the left subtrees to the right subtrees approaches the golden ratio ϕ (since $L(n)/R(n)$ approaches ϕ). The golden trees can be seen as a supersequence of the Fibonacci trees [5, p.414] in the sense that a golden tree G_n corresponds to the Fibonacci tree of order k whenever $n = F_{k+1} - 1$, $k \geq 0$. See Figure 2, trees G_0, G_1, G_2, G_4, G_7, and G_{12} correspond to Fibonacci trees.

The main lemma for golden trees is stated below. It says that if two golden trees of appropriate sizes are melded, then the result is a golden tree as well. In particular, if the left and right subtrees of the root of a golden tree G_n are melded, then the result is a G_{n-1} tree.

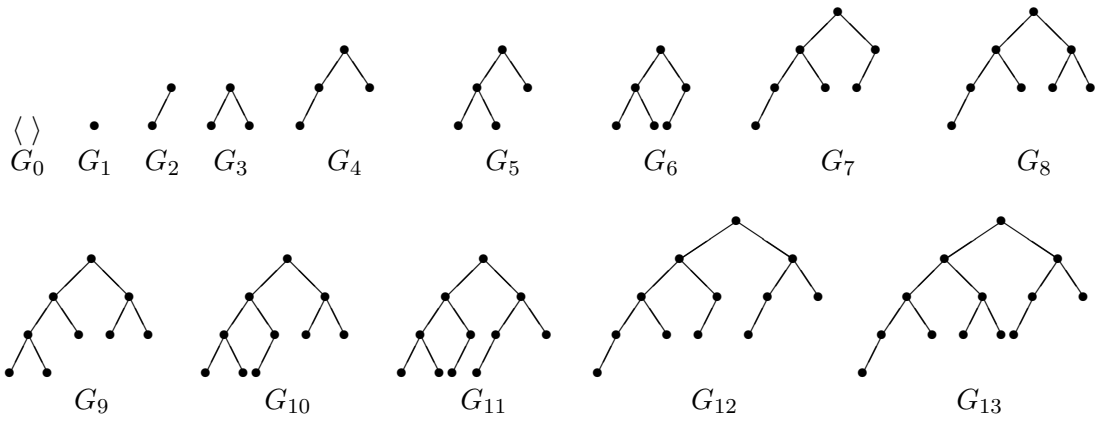

Figure 2: G_n trees for $n = 0, \ldots, 13$.

Lemma 7 $G_{L(n)} \bowtie G_{R(n)} = G_n$, $n \geq 0$.
Proof By induction on n. Clearly true for $n = 0$. For $n \geq 1$, we note:

$$
\begin{aligned}
& G_{L(n)} \bowtie G_{R(n)} \\
= & \quad \{ \text{ definition } G \ \} \\
& \langle G_{L(L(n)-1)}, G_{R(L(n)-1)} \rangle \bowtie G_{R(n)} \\
= & \quad \{ \text{ definition } \bowtie \ \} \\
& \langle G_{R(n)} \bowtie G_{R(L(n)-1)}, G_{L(L(n)-1)} \rangle \\
= & \quad \{ \text{ Lemmas 4, 6, and 5, resp. } \} \\
& \langle G_{L(L(n-1))} \bowtie G_{R(L(n-1))}, G_{R(n-1)} \rangle \\
= & \quad \{ \text{ induction hypothesis, using } L(n-1) < n \ \} \\
& \langle G_{L(n-1)}, G_{R(n-1)} \rangle \\
= & \quad \{ \text{ definition } G \ \} \\
& G_n.
\end{aligned}
$$

□

The cost of computing G_n from $G_{L(n)}$ and $G_{R(n)}$ can now be related to the size of G_n. Clearly, we have $|G_n| = n + 1$. As defined before, the cost of $x \bowtie y$ is equal to the sum of the lengths of the rightmost paths of x and y. Alternatively, the cost of $x \bowtie y$ is equal to the length of the leftmost path of $x \bowtie y$ (see, for example, Figure 3). Using $||x||$ to denote the length of the leftmost path of x, formally defined by $||\langle\ \rangle|| = 0$ and $||\langle t, u \rangle|| = ||t|| + 1$, we then have the following lemmas.

Lemma 8 $L(F_k - 2) = F_{k-1} - 1$ *and* $L(F_{k+1} - 3) = F_k - 2$ *, for* $k \geq 3$.
Proof (Only first part, second part is similar.) By induction on k. If $k = 3$ both sides are zero, and if $k = 4$ both sides are one. For $k \geq 5$, we have:

$$
\begin{aligned}
& L(F_k - 2) \\
= & \quad \{ \text{ definition } F \ \} \\
& L(F_{k-1} + F_{k-2} - 2)
\end{aligned}
$$

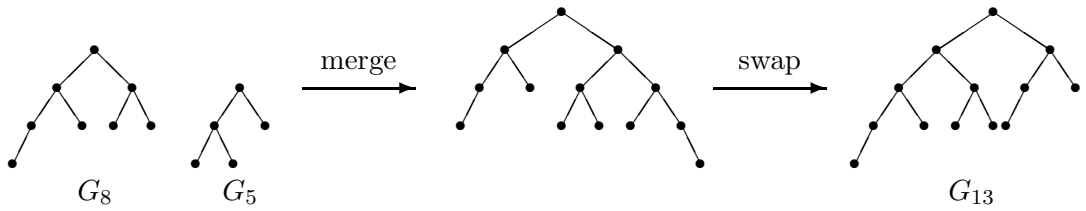

Figure 3: Operation $G_8 \bowtie G_5$ viewed as first merging the rightmost paths and then swapping the subtrees of all nodes on the rightmost path, resulting in G_{13}.

$$
\begin{array}{ll}
= & \{ \text{ Lemma 1, using } F_{k-2} \geq 2 \ \} \\
& F_{k-2} + L(F_{k-2} - 2) \\
= & \{ \text{ induction hypothesis } \} \\
& F_{k-2} + F_{k-3} - 1 \\
= & \{ \text{ definition } F \ \} \\
& F_{k-1} - 1.
\end{array}
$$

□

Lemma 9 *$\|G_n\| = k-2$, for $n \geq 0$, where k is the unique integer satisfying $F_k \leq |G_n| < F_{k+1}$.*
Proof By induction on n. Clearly true for $n = 0$ (and $k = 2$). For $n \geq 1$ (hence $k \geq 3$), we note that $\|G_n\| = \|G_{L(n-1)}\|+1$. From the induction hypothesis we get that $\|G_{L(n-1)}\| = k-3$, so the result follows provided $F_{k-1} \leq L(n-1)+1 < F_k$ is implied by $F_k \leq n+1 < F_{k+1}$. For the lower bound we note that $F_k - 2 \leq n-1$ implies that $F_{k-1} - 1 \leq L(n-1)$, using Lemma 8. For the upper bound we note that $n-1 \leq F_{k+1}-3$ implies that $L(n-1) \leq F_k - 2$, again using Lemma 8. □

This leads to the following conclusion. Let $F_k \leq n+1 < F_{k+1}$ and consider the computation of the meld of $G_{L(n)}$ and $G_{R(n)}$. On account of Lemma 9 the actual cost of this operation is $\|G_n\| = k-2$. The upper bound (Theorem 1) for the amortized cost of this operation is $\log_\phi |G_n| = \log_\phi(n+1)$, which is bounded by approximately $k - 0.67$, using that $F_{k+1} \approx \phi^{k+1}/\sqrt{5}$. So, the actual cost differs at most a small constant from the alloted amortized cost for such a meld operation.

3 Labelled case

The central properties achieved in the previous section are the facts that tree G_n can be written both as $G_n = G_{L(n)} \bowtie G_{R(n)}$ and as $G_n = \langle G_{L(n-1)}, G_{R(n-1)} \rangle$, $n \geq 1$, and that $\|G_n\|$ is approximately equal to $\log_\phi n$. In this section we use these results to construct a worst-case sequence of skew heap operations. The plan is to consider a particular sorting program and to see to it that each meld and delmin takes maximal time.

$$
\begin{array}{rcll}
sort.N & = & h.(g.N) & \\
g.0 & = & \mathsf{empty} & \\
g.1 & = & \mathsf{single}.some & \\
g.n & = & \mathsf{meld}.(g.L(n)).(g.R(n)), & n \geq 2 \\
h.x & = & [\,], & \mathsf{isempty}.x \\
h.x & = & \mathsf{min}.x \vdash h.(\mathsf{delmin}.x), & \neg\mathsf{isempty}.x
\end{array}
$$

Function g first builds a skew heap, where the elements for the singleton heaps are chosen appropriately. We will show that it is possible to choose each singleton element such that each $g.n$ heap is a G_n-tree, and moreover such that each tree to which h is applied, is a G_n-tree as well. It then follows from the results for the unlabelled case that the applications of meld and delmin all take maximal time.

For this to hold it suffices to show the existence of a labelling for which each application of meld and delmin simulates the behaviour of $\bowtie$. The next lemma captures the essence, where we limit ourselves to labellings without duplicates.

Lemma 10 *Consider any labelled G_n-heap x, $n \geq 0$. Then there exist a labelled $G_{L(n)}$-heap t and a labelled $G_{R(n)}$-heap u such that* meld.$t.u = x$.

Proof On account of Lemma 7, we know that $G_n = G_{L(n)} \bowtie G_{R(n)}$. This defines a one-to-one mapping between the nodes of G_n on the one hand and the nodes of $G_{L(n)}$ and $G_{R(n)}$ on the other hand. If we now copy the labels of x to trees t and u according to this mapping, we know that both t and u are heaps, and that indeed meld.$t.u = x$ provided that the labels of the rightmost paths of t and u alternate, starting with t. This is indeed true because the leftmost path of x forms an increasing list. □

We work backwards to show that the labels in the program *sort* (in single.*some*) can be picked such that each application of delmin and meld simulates $\bowtie$. First consider a labelled version of the G_n trees, such that delmin.$G_n = G_{n-1}$ and min.$G_n = -n$. The sequence is defined inductively by $G_0 = \langle\,\rangle$ and $G_n = \langle t, -n, u\rangle$, $n \geq 1$, where t is a labelled $G_{L(n-1)}$ tree and u is a labelled $G_{R(n-1)}$ tree such that meld.$t.u = G_{n-1}$. The existence of these labelled trees is guaranteed by Lemma 10. This fixes all the trees to which h is applied, hence tree $g.N$ as well. On account of Lemma 10 it follows that there exist labelled versions of $g.L(N)$ and $g.R(N)$ for which meld yields $g.N$. Repeating this argument it then follows that a (unique) assigment of the inputs to single exists that satisfies our requirements. As the argument holds for any N, $N \geq 0$, we have proved that:

Theorem 2 *There exist sequences of operations on top-down skew heaps such that the heaps may get arbitrarily large and for which the actual costs satisfy (in terms of comparisons):* empty, isempty.x, min.x, *and* single.a *cost 0,* delmin.x *costs at least* $\log_\phi |x| - c$, *and* meld.$x.y$ *costs at least* $\log_\phi(|x| + |y|) - c$, *where* $\phi = (\sqrt{5}+1)/2$ *denotes the golden ratio and c is a small constant, $c < 2$.*

4 Concluding remarks

As defined in Figure 1 operation meld.$x.y$ terminates as soon as the end of the rightmost path of either x or y is reached. It is also possible to define meld such that melding continues until both rightmost paths are completely traversed. The same bounds apply to this version of meld. (Actually, this version was analyzed in [4], and in [8] it was shown that the same upper bounds hold for both versions.)

It would be interesting to extend our results to bottom-up skew heaps as well. In [8] several sets of amortized bounds have been derived. Just as for top-down skew heaps, operations empty, isempty, single, and min all take $O(1)$ time. One set of bounds [8, Lemma 9.10] says that it is possible to amortize the costs (counting comparisons) such that the amortized costs are at most 3 for meld and at most $1 + 2\log_2 |x|$ for delmin.x. This improves upon the original

bounds by Sleator and Tarjan of [9] by a factor of two. A new parameterized set of bounds that is incomparable with previous bounds [8, Lemma 9.12] says that the amortized costs can be chosen at most $1+\varepsilon\log_\beta(|x|+|y|)$ for $\mathsf{meld}.x.y$ and at most $1+(\varepsilon+2)\log_\beta|x|$ for $\mathsf{delmin}.x$, where $\beta=\frac{(\varepsilon+1)^{\varepsilon+1}}{\varepsilon^\varepsilon}$, for any $\varepsilon>0$.

Picking $\varepsilon=\phi$ in the latter case yields as upper bounds $\frac{\phi}{\phi+2}\log_\phi(|x|+|y|)$ for $\mathsf{meld}.x.y$ and $\log_\phi|x|$ for $\mathsf{delmin}.x$. Since we now know that the bounds of Theorem 1 are tight, it follows that bottom-up skew heaps outperform top-down skew heaps, as the bound for meld is better. It is an interesting open problem whether the bounds for bottom-up skew heaps are tight as well.

References

[1] M.J. Fischer, Efficiency of equivalence algorithms, in: R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations* (Plenum Press, New York, 1972) 153–168.

[2] D.R. Hofstadter, *Gödel, Escher, Bach: an Eternal Golden Braid*, Basic Books (1979).

[3] D.W. Jones, Concurrent operations on priority queues, *Communications of the ACM* **32** (1989) 132–137.

[4] A. Kaldewaij and B. Schoenmakers, The derivation of a tighter bound for top-down skew heaps, *Information Processing Letters* **37** (1991) 265–271.

[5] D. Knuth, *The Art of Computing Programming*, Volume 3, Sorting and Searching, Addison Wesley (1975).

[6] C. Okasaki, Amortization, lazy evaluation, and persistence: Lists with catenation via lazy linking, in: *IEEE Symposium on Foundations of Computer Science* (October 1995) 646–654.

[7] C. Okasaki, The role of lazy evaluation in amortized data structures, in: *ACM SIGPLAN International Conference on Functional Porgramming* (May 1996) 62–72.

[8] B. Schoenmakers, *Data Structures and Amortized Complexity in a Functional Setting*, Ph.D. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (1992).

[9] D.D. Sleator and R.E. Tarjan, Self-adjusting heaps, *SIAM Journal on Computing* **15** (1986) 52–69.

[10] N.J.A. Sloane and S. Plouffe, *The Encyclopedia of Integer Sequences*, Academic Press (1995).

[11] R.E. Tarjan, Efficiency of a good but not linear set union algorithm, *Journal of the ACM* **22** (1975) 215–225.

[12] R.E. Tarjan and J. van Leeuwen, Worst-case analysis of set union algorithms, *Journal of the ACM* **31** (1984) 245–281.