Pairing Heaps with Costless Meld

Amr Elmasry

Max-Planck Institut für Informatik and University of Copenhagen
elmasry@mpi-inf.mpg.de

Abstract. Improving the structure and analysis in [1], we give a variation of pairing heaps that has amortized zero cost per meld (compared to an $O(\lg \lg n)$ in [1]) and the same amortized bounds for other operations. More precisely, the new pairing heap requires: no cost per meld, $O(1)$ per find-min and insert, $O(\lg n)$ per delete-min, and $O(\lg \lg n)$ per decrease-key, where $n$ is the size of the priority queue at the time the operation is performed. These bounds are the best known for any self-adjusting heap, and match the lower bound established by Fredman for a family of such priority queues. Moreover, our structure is even simpler than that in [1].

1 Introduction

The pairing heap [5] is a self-adjusting heap that is implemented as a single heap-ordered multiway tree. A primitive operation is the join operation in which two trees are combined by linking the root with the larger key value as the leftmost child of the other. The following operations are defined for the standard implementation of pairing heaps:

- find-min. Return the value at the root of the heap.
- insert. Create a single-node tree and join it with the main tree.
- decrease-key. Decrease the value of the corresponding node. If this node is not the root, cut its subtree and join the two resulting trees.
- meld. Join the two trees representing the two priority queues.
- delete-min. Remove the root of the heap and return its value. The resulting trees are then combined to form a single tree. The joins are performed in two passes. In the first pass, called the pairing pass, the trees are joined in pairs from left to right (pairing these trees from right to left achieves the same amortized bounds). In the second pass, called the right-to-left incremental-pairing pass, the resulting trees are joined in order from right to left, where each tree is joined with the combined tree resulting from the joins of the trees to its right. (Other variants with different delete-min implementation were given in [1,2,3,5].)

The original analysis of pairing heaps [5] established $O(\lg n)$ amortized cost for all operations. Around the same time, the skew heap, another self-adjusting heap

* Supported by Alexander von Humboldt Fellowship and VELUX Fellowship.
that has $O(\lg n)$ amortized cost per operation, was also introduced \[11\]. Theoretical results concerning pairing heaps and their variants were later obtained through the years. The amortized bounds for the standard implementation were improved by Iacono \[7\] to: $O(1)$ per insert, and zero cost per meld. Pettie \[10\] proved amortized costs of: $O(\lg n)$ per delete-min, and $O(2^{\sqrt{\lg \lg n}})$ for other operations including decrease-key. Some variants were also introduced. Stasko and Vitter \[12\] suggested a variant that achieves $O(1)$ amortized cost per insert. Elmasry \[1\] introduced a variant that achieves the following amortized bounds: $O(1)$ per insert, $O(\lg n)$ per delete-min, and $O(\lg \lg n)$ per decrease-key and meld. See Table \[1\] Fredman \[4\] showed that $\Omega(\lg \lg n)$ amortized comparisons, in the decision-tree model, would be necessary per decrease-key operation for a family of priority queues that generalizes pairing heaps.

Several experiments were conducted on pairing heaps, either comparing their performance with other priority queues \[8,9\] or with some variants \[2,3,12\]. Such experiments illustrate that pairing heaps are practically efficient and superior to other priority queues, including Fibonacci heaps \[6\].

In this paper, we give another variant of pairing heaps that achieves the best bounds known for any self-adjusting heap. Namely, our amortized bounds are: zero cost per meld, $O(1)$ per find-min and insert, $O(\lg n)$ per delete-min, and $O(\lg \lg n)$ per decrease-key. To achieve these bounds, we apply similar ideas to those in \[1\] adapted to efficiently support meld. In addition, we avoid using an insertion buffer, which makes the new structure even simpler than that in \[1\]. To prove our bounds, we use similar ideas to those of Iacono’s analysis to the standard implementation \[7\], and tailor them for our new structure.

We describe the data structure in Section 2, explain our accounting strategies in Section 3, prove the time bounds in Section 4, and conclude the paper with three open questions in Section 5.

### 2 The Data Structure

Our priority-queue structure is composed of two components: the main tree and the decrease pool. The decrease pool holds at most $\lceil \lg n \rceil$ heap-ordered trees of various sizes, which are the subtrees that have been cut by decrease-key operations since the last call to the consolidate operation (see below). Similar to the

|               | insert | delete-min | decrease-key | meld |
|---------------|--------|------------|--------------|------|
| Fredman et al. \[5\] | $O(\lg n)$ | $O(\lg n)$ | $O(\lg n)$ | $O(\lg n)$ |
| Iacono \[7\]     | $O(1)$  | $O(\lg n)$ | $O(\lg n)$ | zero |
| Pettie \[10\]    | $O(2^{2\sqrt{\lg \lg n}})$ | $O(\lg n)$ | $O(2^{\sqrt{\lg \lg n}})$ | $O(2^{\sqrt{\lg \lg n}})$ |
| Stasko and Vitter \[12\] | $O(1)$  | $O(\lg n)$ | $O(\lg n)$ | $O(\lg n)$ |
| Elmasry \[1\]    | $O(1)$  | $O(\lg n)$ | $O(\lg \lg n)$ | $O(\lg \lg n)$ |
| This paper       | $O(1)$  | $O(\lg n)$ | $O(\lg \lg n)$ | zero |

Table 1. Upper bounds on the operations of the pairing heap (first rows) and its variants (last rows). All the time bounds are amortized.