

A Generalization of Binomial Queues

Rolf Fagerberg *

Department of Mathematics and Computer Science
Odense University
Campusvej 55, DK-5230 Odense M, Denmark

Abstract

We give a generalization of binomial queues involving an arbitrary sequence $(m_k)_{k=0,1,2,\dots}$ of integers greater than one. Different sequences lead to different worst case bounds for the priority queue operations, allowing the user to adapt the data structure to the needs of a specific application. Examples include the first priority queue to combine a sub-logarithmic worst case bound for *Meld* with a sub-linear worst case bound for *Delete_min*.

Keywords: Data structures; Meldable priority queues.

1 Introduction

The binomial queue, introduced in 1978 by Vuillemin [14], is a data structure for meldable priority queues. In meldable priority queues, the basic operations are *insertion* of a new item into a queue, *deletion* of the item having minimum key in a queue, and *melding* of two queues into a single queue. The binomial queue is one of many data structures which support these operations at a worst case cost of $O(\log n)$ for a queue of n items. Theoretical [2] and empirical [9] evidence indicates that binomial queues are competitive in terms of the constant factors involved in the bounds. Furthermore, they are optimal in the amortized sense, having amortized complexities of $O(1)$ for *Insert* and *Meld*, and $O(\log n)$ for *Delete_min* [10].

Binomial queues have served as the basis for many further extensions [3, 5, 6, 7, 8, 10]. Of these, the most well-known is probably the Fibonacci heaps of Fredman and Tarjan [7], which allow decreasing the key of an item at $O(1)$ amortized cost, as well as supporting insertions and melds at $O(1)$ cost (amortized and worst case). Deletion of the minimum item is done at $O(\log n)$ amortized cost. Similar bounds are achieved in [5, 8, 13]. These data structures have theoretical properties superior to binomial queues, but they are also more complex, and may be less advantageous in practice. Several empirical investigations [4, 11, 12] confirm this in the case of Fibonacci heaps.

Here, we consider the original binomial queues, and show how a slight generalization of their definition allows us to vary the worst case bounds for the three basic operations, *Insert*, *Delete_min*, and *Meld*. More precisely, each sequence $(m_k)_{k=0,1,2,\dots}$ of integers greater than one defines a generalization, and different sequences result in different worst case bounds. By proper choice of sequence, one can adjust the data structure to the particular needs of an application. This situation is comparable to that of d -heaps (an implicit heap [15], based on a d -ary tree instead of a binary tree), which support *Insert* at $O(\log_d n)$ cost and *Delete_min* at $O(d \log_d n)$ cost. Here, however, the tradeoff possible between the bounds is not only in terms of constant factors. Rather, asymptotically better worst case bounds for *Insert* and

*E-mail address: rolf@imada.ou.dk. Supported by SNF grant 11-0575.

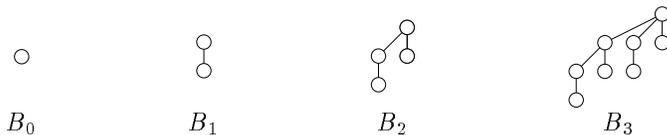
Meld can be achieved at the expense of the worst case bound for *Delete_min*, and the degree of this tradeoff can be varied. We refer to section 4 for examples. The generalization is simple, and does not incur much overhead compared to standard binomial queues, so the price for the extra flexibility is low.

All previous proposals for meldable priority queues have worst case bounds for *Meld* and *Delete_min* of either $O(\log n)$ for both operations, or of $O(1)$ and $O(n)$, respectively. Thus, we bridge this gap by providing the first examples of priority queues which support melding in sub-logarithmic worst case time, without allowing linear time for deletion.

ADDED IN PRINT: It should be noted that subsequent to the submission of this paper, a priority queue has been given [1] which achieves the optimal worst case complexities of $O(1)$ for *Insert* and *Meld*, and $O(\log n)$ for *Delete_min*.

2 Binomial Queues

We briefly review binomial queues. The family of *binomial trees* B_k , where $k = 0, 1, 2, \dots$, is defined inductively as follows: B_0 consists of just one node, and any other B_k is formed by *linking* two B_{k-1} trees together, where linking is defined as making one of the roots a child of the other. The integer k is called the *rank* of B_k . Below, the first few binomial trees are shown.



A tree containing keys is *heap ordered* if the key of any child is no less than that of its parent. Thus, in a heap ordered tree, the minimum key is at the root. A *binomial queue* is defined as a forest of heap ordered binomial trees containing zero or one tree of each rank.

Melding of two binomial queues is analogous to binary addition: If more than one B_0 is present in the two queues, they are linked together to form a B_1 , corresponding to a carry in binary addition. If now more than one B_1 is present, two are linked together to form a B_2 , and so on. Insertion of an item is just a meld with a queue of size one. Deletion of the minimum item is performed by locating the root with minimum key, removing it, making a new binomial queue of its children (which are B_0, B_1, \dots, B_{k-1} , for some k), and melding this queue with the rest of the original one.

As B_k contains 2^k nodes, the maximum k in a binomial queue is $\lceil \log n \rceil$.¹ This provides the $O(\log n)$ worst case bound for *Insert*, *Meld*, and *Delete_min*.

3 Generalized Binomial Queues

In a binomial queue, only zero or one tree of each rank is permitted. The idea we follow here is to generalize binomial queues such that the maximum number of trees of each rank k can be determined by *any* number m_k greater than one.

In this section, we give appropriate definitions for such a generalization, as well as an efficient implementation of them. In the next section, we look at what impact different sequences $(m_k)_{k=0,1,2,\dots}$ have on the worst case bounds of the priority queue operations.

We start by defining the *linking* of more than two trees in the obvious way:

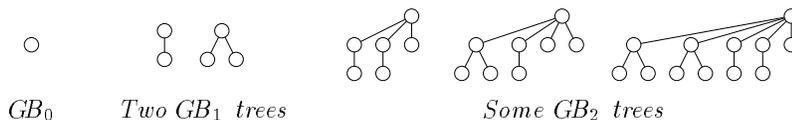
¹Unless otherwise indicated, logarithms and exponential functions are to base 2.

Definition 1 Two or more trees of the same rank are linked by making the root of one of them the parent of the rest.

The definition of the trees we use is less straightforward. For each sequence $(m_k)_{k=0,1,2,\dots}$ of integers greater than one, there is different family of trees, defined inductively as follows:

Definition 2 For a sequence $(m_k)_{k=0,1,2,\dots}$ of integers with $m_k > 1$, all k , a corresponding generalized binomial tree GB_k of rank k consists of one node when $k = 0$, and otherwise is formed by linking γ GB_{k-1} trees together, for some integer γ such that $m_{k-1} \leq \gamma \leq 2m_{k-1} - 1$.

Note that there are many different trees of the same rank, due to the latitude in the choice of γ . Examples of GB_0 , GB_1 , and GB_2 trees corresponding to the sequence $(m_k) = 2, 3, 4, 5, \dots$ are shown below.



Generalized binomial queues can now be defined:

Definition 3 Given a sequence $(m_k)_{k=0,1,2,\dots}$ of integers with $m_k > 1$, all k , a corresponding generalized binomial queue is a forest of heap ordered generalized binomial trees corresponding to the same sequence, containing fewer than m_k trees of rank k .

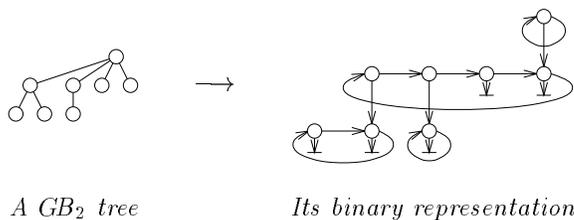
The following properties of generalized binomial trees will be used later.

Lemma 1 For a generalized binomial tree GB_k of rank k , we have:

- (i) The subtrees of the root have rank less than k , and if γ_i denotes the number having rank i , then $(m_i - 1) \leq \gamma_i \leq 2(m_i - 1)$ for $i = 0, 1, 2, \dots, k - 1$.
- (ii) If $|GB_k|$ denotes the number of nodes in GB_k , then $\prod_{i=0}^{k-1} m_i \leq |GB_k| \leq \prod_{i=0}^{k-1} (2m_i - 1)$.

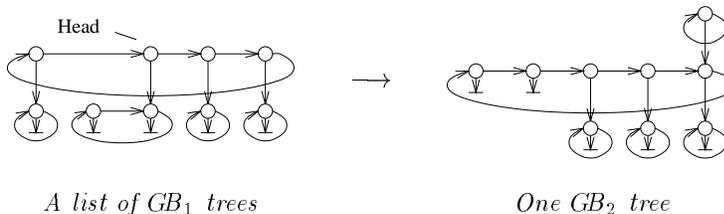
PROOF. Induction on k . □

Next, we describe a simple implementation of generalized binomial queues where the linking of all trees of the same rank can be done with $O(1)$ work. The implementation uses a variant of the standard representation of multi-way trees as binary trees. Each node contains two pointers, p_1 and p_2 , pointing to the rightmost child and the right sibling, respectively, of the node. The p_2 -pointer of the rightmost sibling points to the leftmost sibling, making all siblings appear in a singly linked, circular list (so both the leftmost and the rightmost child of a node can be easily accessed). An example of a multi-way tree and its binary representation is shown below.



Using the p_2 -pointers at the roots of the trees, we also keep all trees of the same rank in the queue in a singly linked, circular list. Hence, a queue is a collection of such lists, one for each rank k present. In each of these list, the tree containing the smallest key is referred to as the *head*. With each list, we store its length and a pointer to the tree immediately before the head (so the head can be removed easily).

With this implementation, the linking of all trees of the same rank in the queue is easy: remove the head from the list of trees, and concatenate the remainder of this list with the removed nodes list of children. Thus, the remaining trees in the list become the new rightmost children of the head. The linking of a list of GB_1 trees into one GB_2 tree is shown below.



This process requires $O(1)$ work, and heap order is preserved in the resulting tree.

As in standard binomial queues, *melding* of two queues consists of a sequence of iterations, one for each rank present, starting with the smallest. At each iteration, the trees may be linked to form a tree of rank one higher, analogous to a carry. An iteration for rank k starts by concatenating the two lists of trees of rank k and the possible carry from the previous iteration into a single list of trees. The new head is the minimum among the two old and the carry. The size of the resulting list is computed from the sizes of the two initial lists, and if there are m_k trees or more, we link them to form a tree of rank one higher. We can link *all* the trees of the list and still produce a legal tree of the next rank, as there can be no more than $2(m_k - 1) + 1$ trees in the list. This is the reason for letting γ vary in definition 2. A more direct generalization of binomial trees would allow only $\gamma = m_k$, but then lists must be split during linking, making the time used for linking depend on the bounds m_k .

Insertion of a new item is just a meld. For the *deletion* of the item with minimum key, we inspect the heads of the tree lists to find the appropriate root, remove it, build new queues of its subtrees, and finally meld the new queues with the remainder of the original queue. By Lemma 1, two new queues are sufficient to hold the released subtrees. To build the new queues, we note that the subtrees of the removed root appear with increasing rank in its list of children (since during links, new children are added to the right of existing children). Traversing this list, we can fill the list of trees in the new queues, using $O(1)$ work per tree. The only problem is recognizing the ranks of the trees during the traversal. Therefore, at each node we keep an integer storing the rank of the subtree rooted at that node. This integer must then be updated during links (only one integer needs to be updated per link). Actually, storing the rank modulo 2 is enough for the recognition of the ranks during the traversal, so one bit of extra information per node suffices.

The worst case bounds for these operations depend on the sequence (m_k) in the following way:

Lemma 2 *Given a sequence $(m_k)_{k=0,1,2,\dots}$ of integers with $m_k > 1$, all k , define two functions f and g by $f(n) = \max\{k \mid \prod_{i=0}^{k-1} m_i \leq n\}$ and $g(k) = \sum_{i=0}^{k-1} m_i$. The operations on generalized binomial queues corresponding to this sequence have the*

following worst case bounds

$$\text{Insert, Meld} \in O(f(n)), \quad \text{Delete_min} \in O(g(f(n))),$$

where n is the number of items in the queue (Meld: in the largest queue).

PROOF. If K denotes the maximum rank of a tree in a queue of n items, we have $\prod_{i=0}^{K-1} m_i \leq n$ by Lemma 1. Therefore $K \leq f(n)$. During melds, we perform $O(1)$ work for each rank from zero to the maximum rank present. This gives the bound for *Meld* and *Insert*. The bound for *Delete_min* is dominated by the handling of the released subtrees. As we do $O(1)$ work for each subtree, Lemma 1 shows that *Delete_min* is in $O(\sum_{i=0}^{K-1} m_i) = O(g(f(n)))$. \square

4 Examples

In this section, we first analyze the complexities for one specific sequence, and then discuss some more general cases. Finally, we prove that (unfortunately) sub-logarithmic complexities for *Insert* and *Meld* can only be obtained at the expense of a super-logarithmic complexity for *Delete_min*.

Theorem 1 *There is a data structure for meldable priority queues, having the following worst case bounds:*

$$\text{Insert, Meld} \in O\left(\frac{\log n}{\log \log \log n}\right), \quad \text{Delete_min} \in O\left(\frac{\log n \log \log n}{\log \log \log n}\right).$$

PROOF. Consider the sequence $(m_k) = 2, 3, 3, 4, 4, 4, 4, 5, \dots$, formally defined by $m_k = \lfloor \log(k+1) \rfloor + 2$. To find the function f of Lemma 2, we note that $\prod_{i=0}^{k-1} m_i = \exp(\sum_{i=0}^{k-1} \log m_i) \geq \exp(\sum_{i=2}^{k-1} \log m_i)$. Regarding sums as integrals of step functions, we have $\sum_{i=r}^s h(i) \geq \int_{r-1}^s h(t) dt$ for any increasing function h . By partial integration,

$$\begin{aligned} \int_1^{k-1} \log(\lfloor \log(t+1) \rfloor + 2) dt &\geq \int_2^k \log \log t dt \\ &= k \log \log k - \frac{1}{(\log_e 2)^2} \int_2^k \frac{1}{\log t} dt \\ &\geq k \log \log k - \frac{1}{(\log_e 2)^2} \int_2^k dt \\ &\geq k \log \log k - 2.1k. \end{aligned}$$

Combining, we get $\prod_{i=0}^{k-1} m_i \geq \exp(k(\log \log k - 2.1)) = (\log k / 2^{2.1})^k$. For any increasing function β with inverse β^{-1} and any function α , $\alpha(\beta(n)) \in \Theta(n)$ is equivalent to $\beta^{-1} \in \Theta(\alpha(n))$. For $\beta(n) = (\log n / c)^n$, c a constant, and $\alpha(n) = \log n / \log \log \log n$, the first condition is easily verified. Thus, the function $f(n)$ from Lemma 2 is in $O(\log n / \log \log \log n)$. As $\sum_{i=1}^k \log i$ is in $\Theta(k \log k)$, so is the function $g(k)$ from Lemma 2. It follows that $g(f(n)) \in O(\log n \log \log n / \log \log \log n)$. \square

In general, if the sequence $(m_k)_{k=0,1,2,\dots}$ is bounded from above, then the worst case complexities will be $O(\log n)$ for all operations. For instance, if $(m_k) = d, d, d, d, \dots$, then $f(n) = \lfloor \log_d n \rfloor$ and $g(k) = dk$, giving

Theorem 2 *For any integer $d \geq 2$, there is a data structure for meldable priority queues with the following worst case bounds:*

$$\text{Insert, Meld} \in O(\log_d n), \quad \text{Delete_min} \in O(d \log_d n).$$

These bounds are the same as for binomial queues, except that a tradeoff in terms of constant factors can be achieved, just as for d -heaps (but unlike d -heaps, the *Meld* operation is supported).

On the other hand, if $(m_k)_{k=0,1,2,\dots}$ is nondecreasing and unbounded, then $f(n) \in o(\log n)$, as in Theorem 1. We give two more examples of this.

Theorem 3 *For any integer $d \geq 1$, there is a data structure for meldable priority queues with the following worst case bounds:*

$$\text{Insert, Meld} \in O\left(\frac{\log n}{d \log \log n}\right), \quad \text{Delete_min} \in O\left(\left(\frac{\log n}{d \log \log n}\right)^{d+1}\right).$$

PROOF. For the polynomially growing sequence $(m_k) = 2^d, 3^d, 4^d, 5^d, \dots$, we have $\prod_{i=0}^{k-1} m_i = ((k+1)!)^d$, so f is the inverse of the factorial function, applied to $\sqrt[d]{n}$. Using Stirling's formula, $n! \sim \sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n}$, it is easily verified that $\log(n!)/\log \log(n!) \in \Theta(n)$. This implies (cf. the proof of Theorem 1) that $f \in \Theta(\log n / d \log \log n)$. As $g(k) = 2^d + 3^d + \dots + (k+1)^d \in \Theta(k^{d+1})$, the theorem follows. \square

Theorem 4 *For any integer $d \geq 2$, there is a data structure for meldable priority queues with the following worst case bounds:*

$$\text{Insert, Meld} \in O(\sqrt{\log_d n}), \quad \text{Delete_min} \in O(d^{\Theta(\sqrt{\log_d n})}).$$

PROOF. For the exponentially growing sequence $(m_k) = d^1, d^2, d^3, d^4, \dots$, we have $\prod_{i=0}^{k-1} m_i \in d^{\Theta(k^2)}$. Thus, $f(n)$ is in $\Theta(\sqrt{\log_d n})$. As $g(k) = d^1 + d^2 + \dots + d^k \in \Theta(d^k)$, the theorem follows. \square

These examples suggest that if for any sequence $f(n) \in o(\log n)$, then we must necessarily have $g(f(n)) \in \omega(\log n)$. This can indeed be proven:

Theorem 5 *If for any sequence $(m_k)_{k=0,1,2,\dots}$ the corresponding function f is in $o(\log n)$, then $g(f(n))$ is in $\omega(\log n)$.*

PROOF. For asymptotic behavior, it is enough to consider integers n such that $n = \prod_{i=0}^{k-1} m_i$ for some k . By the definition of f , $k = f(n)$. From the standard inequality $(\prod_{i=0}^{k-1} m_i)^{1/k} \leq (\sum_{i=0}^{k-1} m_i)/k$ between the geometric and the arithmetic means of k positive numbers, we get $f(n)n^{1/f(n)} \leq g(f(n))$. If f is in $o(\log n)$, then $\log n/f(n)$ is in $\omega(1)$. For any function $h \in \omega(1)$, we have $\exp(h(n)) \in \omega(h(n))$. Thus, $f(n)n^{1/f(n)} = f(n) \exp(\log n/f(n))$ is in $\omega(f(n) \log n/f(n)) = \omega(\log n)$. \square

5 Conclusion

As a final remark, it seems that besides adding flexibility to binomial queues, our generalization also takes care of one objection to them in [9]. In standard binomial queues, the set of trees for a queue of size n is uniquely determined by the binary expansion of n . If alternating *Insert* and *Delete_min* operations make the queue oscillate around a size $n = 111\dots 11_2$, worst case performance will arise at every operation, as the carry will propagate through all the ranks. In generalized binomial queues, there are for each n many possible sets of trees in the queue (since γ can take on many values in definition 2), so this problem seems unlikely to arise.

6 Acknowledgments

The author would like to thank the anonymous referee for suggestions on the exposition, and Peter Høyer for noting that one bit of extra information per node suffices.

References

- [1] Gerth S. Brodal. Fast meldable priority queues. In *Algorithms and Data Structures*, volume 955 of *LNCS*, pages 282–290. Springer-Verlag, 1995.
- [2] Mark R. Brown. Implementation and analysis of binomial queue algorithms. *SIAM J. Comput.*, 7(3):298–319, 1978.
- [3] Svante Carlsson, J. Ian Munro, and Patricio V. Poblete. An implicit binomial queue with constant insertion time. In *SWAT 88*, volume 318 of *LNCS*, pages 1–13. Springer-Verlag, 1988.
- [4] Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. Shortest paths algorithms: Theory and experimental evaluation. In *5th ACM-SIAM SODA'94*, pages 516–525, 1994.
- [5] James R. Driscoll, Harold N. Gabow, Ruth Shrairman, and Robert E. Tarjan. Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. *Comm. of the ACM*, 31(11):1343–1354, 1988.
- [6] Michael L. Fredman, Robert Sedgewick, Daniel D. Sleator, and Robert E. Tarjan. The pairing heap: A new form of self-adjusting heap. *Algorithmica*, 1:111–129, 1986.
- [7] Michael L. Fredman and Robert E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *Proc. 25th Annual IEEE Symp. on Foundations of Comp. Sci.*, pages 338–346, 1984.
- [8] Peter Høyer. A General Technique for Implementation of Efficient Priority Queues. In *Proc. 3rd Israel Symposium on Theory of Computing and Systems, Tel Aviv*, pages 57–66. IEEE Computer Society Press, 1995.
- [9] Douglas W. Jones. An empirical comparison of priority-queue and event-set implementations. *Comm. of the ACM*, 29(4):300–311, 1986.
- [10] C. M. Khoong and H. W. Leong. Double-ended binomial queues. In *Algorithms and Computation*, volume 762 of *LNCS*, pages 128–137. Springer-Verlag, 1993.
- [11] Andrew M. Liao. Three priority queue applications revisited. *Algorithmica*, 7:415–427, 1992.
- [12] Bernard M. E. Moret and Henry D. Shapiro. An empirical analysis of algorithms for constructing a minimum spanning tree. In *Algorithms and Data Structures*, volume 519 of *LNCS*, pages 400–411. Springer-Verlag, 1991.
- [13] Gary L. Peterson. A balanced tree scheme for meldable heaps with updates. Technical Report GIT-ICS-87-23, School of Information and Computer Science, Georgia Institute of Technology, 1987.
- [14] Jean Vuillemin. A data structure for manipulating priority queues. *Comm. of the ACM*, 21(4):309–315, 1978.
- [15] J. W. J. Williams. Algorithm 232: Heapsort. *Comm. of the ACM*, 7:347–348, 1964.