Open access • Book Chapter • DOI:10.1007/978-3-642-40273-9_11

# A Survey on Priority Queues — Source link

Gerth Stølting Brodal

**Institutions:** Aarhus University

**Published on:** 01 Jan 2013

**Topics:** Priority queue and Binary heap

# A Survey on Priority Queues

Gerth Stølting Brodal

MADALGO*, Department of Computer Science, Aarhus University, gerth@cs.au.dk

**Abstract.** Back in 1964 Williams introduced the binary heap as a basic priority queue data structure supporting the operations Insert and ExtractMin in logarithmic time. Since then numerous papers have been published on priority queues. This paper tries to list some of the directions research on priority queues has taken the last 50 years.

## 1 Introduction

In 1964 Williams introduced "Algorithm 232" [125]—a data structure later known as *binary heaps*. This data structure essentially appears in all introductory textbooks on Algorithms and Data Structures because of its simplicity and simultaneously being a powerful data structure.

A tremendous amount of research has been done within the design and analysis of priority queues over the last 50 years, building on ideas originating back to the initial work of Williams and Floyd in the early 1960s. In this paper I try to list some of this work, but it is evident by the amount of research done that the list is in no respect complete. Many papers address several aspects of priority queues. In the following only a few of these aspects are highlighted.

## 2 The beginning: Binary heaps

Williams' binary heap is a data structure to store a dynamic set of elements from an ordered set supporting the insertion of an element (Insert) and the deletion of the minimum element (ExtractMin) in $O(\lg n)$ time, where $n$ is the number of elements in the priority queue[1]. Williams' data structure was inspired by Floyd's 1962 paper on sorting using a tournament tree [65], but compared to Floyd's earlier work a binary heap is *implicit*, i.e. the data structure only uses one array of size $n$ for storing the $n$ elements without using any additional space[2]. For a set of size $n$ it is simply an array $A[1..n]$ storing the $n$ elements in a permutation implicitly representing a binary tree satisfying *heap order*, i.e. $A[i] \leq A[2i]$ and

* Center for Massive Data Algorithms, a Center of the Danish National Research Foundation.

[1] We let $\lg x$ denote the binary logarithm of $x$.

[2] In the following we denote a data structure storing $O(1)$ words of $\lg n$ bits between the operations also as being implicit. The additional space will be stated explicitly in these cases.

$A[i] \leq A[2i+1]$ for all $1 \leq i \leq n$ (provided $2i \leq n$ and $2i+1 \leq n$, respectively). Williams gave an algorithm for constructing a heap from an array of $n$ elements in $O(n \lg n)$ time [125]. This was subsequently improved by Floyd [66] to $O(n)$.

The average case performance of the operations on a binary heap was studied in a sequence of papers. Porter and Simon [111] introduced the random heap model, and proved that random insertions into a random heap require about 1.61 element exchanges. Bollobás and Simon [10] considered inserting a random sequence of elements into an initially empty binary heap and proved that the average number of exchanges per element inserted is about 1.7645. Doberkat studied the average number of exchanges for ExtractMin in a random heap [38] and for Floyd's heap construction algorithm [39].

Gonnet and Munro considered the constants in the number of comparisons to maintain a binary heap [80], and gave upper and lower bounds proving that Insert requires $\lg \lg n \pm O(1)$ comparisons (the upper bound still requiring $O(\lg n)$ elements to be moved) and ExtractMin requires $\lg n + \lg^* n \pm O(1)$ comparisons. An $1.5n - O(\lg n)$ lower bound for the number of comparisons for constructing a heap was proved by Carlsson and Chen in [22].

Sack and Strothotte [113] showed how to support the merging of two binary heaps (Meld) of size $n$ and $k$ in time $O(k + \lg n \cdot \lg k)$, where $k \leq n$. If the heaps are represented as binary trees using pointers the additive "$k$" term disappears from their bound.

It is obvious that the $k$ smallest elements in a heap can be extracted by $k$ applications of ExtractMin in $O(k \lg n)$ time. Frederickson [73] proved that the partial order given by a binary heap (or more generally, from any heap ordered binary tree) allows us to select the $k$ smallest elements in $O(k)$ time (reported in arbitrary order).

## 3 Reducing the number of comparisons

All the results in Section 2 assume the partial order of the original binary heaps of Williams. In this section we summarize work on lowering the constants in the number of comparisons by considering priority queues with alternative requirements with respect to the order maintained.

Johnson [88] generalized binary heaps to implicit $d$-ary heaps with $O(\lg_d n)$ and $O(d \lg_d n)$ time for Insert and ExtractMin, respectively. By setting $d = \omega(1)$, $d$-ary heaps achieve sublogarithmic insertion time. Subsequently many priority queues achieved constant time Insert and logarithmic time ExtractMin, surpassing the bounds of Johnson.

The weak heaps of Peterson and Dutton [108, 41] are not completely implicit like binary heaps. They store the input elements in one array and require one additional bit per element. Edelkamp and Wegener [47] proved that sorting $n$ elements using a weak heap uses $n \lg n + 0.09n$ comparisons, getting very close to the information theoretic lower bound of $n \lg n - 1.44n$ comparisons [67]. A refinement of weak heap sorting performs at most $n \lg n - 0.9n$ comparisons [46].

Edelkamp *et al.* [43] studied variations of weak heaps, in particular they reduced the cost of INSERT to be constant.

Elmasry *et al.* [56] studied how to reduce the number of comparisons for priority queues to get close to the optimal number of comparisons, and presented a pointer based spriority queue implementation supporting INSERT with $O(1)$ comparisons and EXTRACTMIN with $\lg n + O(1)$ comparisons. Edelkamp *et al.* [45] recently achieved the same bounds by an implicit priority queue using $O(1)$ extra words of $\lg n$ bits.

## 4 Double-ended priority queues

Atkinson *et al.* [8] generalized the partial order of binary heaps and introduced the implicit Min-Max heaps supporting both EXTRACTMIN and EXTRACTMAX in logarithmic time and having linear construction time. Essentially Min-Max heaps and all the following double-ended priority queues maintain a Min-heap and a Max-heap for some partition of the elements stored.

Carlsson introduced the implicit Deap [21] as an alternative to Min-Max heaps, improving the number of comparisons for EXTRACTMIN/EXTRACTMAX from $\frac{3}{2}\lg n + \lg\lg n$ to $\lg n + \lg\lg n$. Carlsson *et al.* [23] gave a proof that a Deap can be constructed in linear time.

General techniques to convert single ended priority queues into double-ended priority queues were presented by Chong and Sahni [32] and Elmasry *et al.* [57]. Alternative implementations of implicit double-ended queues include [106, 26, 37, 99, 6]. Ding and Weiss [35] presented an implicit double-ended priority queue for multi-dimensional data.

Double-ended priority queues supporting MELD were presented by Ding and Weiss [36], Khoong and Leong [96], and Cho and Sahni [31], which are based on min-max heaps, binomial queues, and leftist trees, respectively.

## 5 Implicit data structures

The original implicit heaps of Williams require $O(\lg n)$ worst-case time for INSERT and EXTRACTMIN. Similar bounds are achieved by several of the above mentioned implicit double-ended priority queues. Carlsson *et al.* [24] described an implicit heap with $O(1)$ time INSERT and $O(\lg n)$ time EXTRACTMIN, storing $O(1)$ extra words of $\lg n$ bits between operations. Edelkamp *et al.* [45] presented an implicit priority queue with the same time bounds and also using $O(1)$ extra words, but only requiring $\lg n + O(1)$ comparisons per EXTRACTMIN. A priority queue with amortized $O(1)$ time INSERT and $O(\lg n)$ time EXTRACTMIN was presented by Harvey and Zatloukal [84], that does not store any additional information between operations.

The existence of efficient implicit priority queue data structures implied the canonical question if efficient implicit dynamic dictionaries also existed. The study of implicit dynamic dictionaries was initiated by Munro and Suwanda [104]

who proved tight $\Theta(\sqrt{n})$ bounds on implicit dictionaries satisfying a fixed partial order. The bounds for implicit dictionaries were subsequently improved by Frederickson [71] who achieved logarithmic time searches and $O(n^{\sqrt{2/\lg n}} \cdot \lg^{3/2} n)$ time updates, and the first polylogarithmic bounds were given by Munro in [101] achieving $O(\lg^2 n)$ time for both updates and searches by encoding the bits of pointers for an AVL-tree by the relative order of pairs of elements. Munro and Poblete [103] presented a semi-dynamic implicit dictionary with $O(\lg^2 n)$ time insertions and $O(\lg n)$ time searches. Subsequently update time was improved to $O(\lg^2 n / \lg n)$ by implicit B-trees [69], and eventually logarithmic time bounds where obtained by Franceschini and Grossi [68]. Franceschini and Munro [70] furthermore reduced the number of exchanges to $O(1)$ while keeping the number of comparisons per operation logarithmic (update bounds being amortized).

## 6 DecreaseKey and Meld

Dijkstra's single source shortest paths algorithm makes essential use of priority queues, and in particular the primitive of lowering the priority of an existing element in the priority queue. Fredman and Tarjan [77] introduced the DecreaseKey operation for this and presented Fibonacci heaps, supporting DecreaseKey in amortized constant time implying a running time of $O(m+n \lg n)$ for Dijkstra's algorithm, improving the previous bound of $O(m \lg_{m/n} n)$ achieved using an $m/n$-ary heap [89]. Fibonacci heaps also resulted in improved algorithms for computing minimum trees in weighted graphs with running time $O(m \lg^* n)$. Fibonacci heaps are a generalization of the binomial queues of Vuillemin [123], which achieve the same performance as Fibonacci heaps except for the DecreaseKey operation. In particular both data structures support Meld in amortized constant time. The worst-case time for Meld in a binomial queue is $\Theta(\lg n)$, but the amortized time was proven to be constant by Khoong and Leong [96].

A sequence of priority queues achieves the same amortized performance as Fibonacci heaps. Peterson [108] gave a solution based on AVL-trees, Driscoll *et al.* [40] presented the rank-relaxed heaps, Kaplan and Tarjan [94] presented the thin heaps, Chan [25] presented the quake heaps, Haeupler *et al.* [81] presented the rank-pairing heaps, and Elmasry [53] presented the violation heaps. Høyer [85] presented a general technique to construct different data structures achieving time bounds matching those of Fibonacci heaps, using red-black, AVL-trees and $(a, b)$-trees. Elmasry improved the number of comparisons of Fibonacci heaps by a constant factor [48].

A direction of research has been to develop priority queues with worst-case time guarantees for the operations supported by Fibonacci heaps. The run-relaxed heaps by Driscoll *et al.* [40] achieve worst-case constant time DecreaseKey operations, but Meld takes logarithmic time. The same result was achieved by Kaplan and Tarjan [93] with fat heaps. Elmasry *et al.* presented two-tier relaxed heaps [58] in which the number of comparisons for ExtractMin is reduced to $\lg n + 3 \lg \lg n + O(1)$. Elmasry *et al.* [55] achieve similar bounds where

DecreaseKey operations are supported with $\lg n + O(\lg \lg n)$ comparisons by introducing structural violations instead of heap order violations. The first priority queue with worst-case $o(\lg n)$ time Meld was a generalization of binomial queues by Fagerberg [63], supporting Meld in $o(\lg n)$ time and ExtractMin in time $\omega(\lg n)$. A priority queue with constant time Insert and Meld, and logarithmic time ExtractMin and DecreaseKey was presented by Brodal [11]. A purely functional implementation of [11] (without DecreaseKey) was given by Brodal and Okasaki in [17].

Comparison based priority queues with worst-case constant time Insert, DecreaseKey and Meld and logarithmic time ExtractMin were presented by Brodal [12], assuming the RAM model. Similar bounds in the RAM model were achieved by Elmasry and Katajainen [59]. Brodal *et al.* [16] recently achieved matching bounds in the pointer machine model

Common to many of the priority queues achieving good worst-case bounds for Meld and/or DecreaseKey are that they use some redundant counting scheme [33] to control the number of trees in a forest of heap ordered trees, the number of structural violations and/or heap order violations.

Kaplan *et al.* [92] emphasized the requirement that the DecreaseKey operation as arguments must take both the element to be deleted and a reference to the priority queue containing this element, since otherwise FindMin, DecreaseKey, or Meld must take non-constant time.

Chazelle [28] introduced the soft heap, a priority queue specialized toward minimum spanning tree computations that is allowed to perform a limited number of internal errors. A simplified version of soft heaps was given by Kaplan and Zwick [95]. Minimum spanning tree algorithms using soft heaps were presented by Chazelle [27] and Pettie and Ramachandran [110], where [110] is an optimal minimum spanning tree algorithm but with unknown complexity.

Mortensen and Pettie [43] presented an implicit priority queue supporting Insert and DecreaseKey in amortized constant time and ExtractMin in logarithmic time, using $O(1)$ words of extra storage.

## 7 Self-adjusting priority queues

Crane [34] introduced the leftist heaps. The leftist heaps of Crane are height balanced heap ordered binary trees, where for each node the height of the left subtree is at least the height of the right subtree. Cho and Sahni [30] introduced a weight-balanced version of leftist trees. Okasaki [105] introduced maxiphobic heaps as a very pedagogical and easy to understand priority queue where operations are based on the recursive melding of binary heap ordered trees.

Sleator and Tarjan introduced the skew heaps [117] as a self-adjusting version of leftist heaps [34], i.e. a data structure where no balancing information is stored at the nodes of the structure and where the structure is adjusted on each update according to some local updating role. A tight analysis was given in [91, 115] for the amortized number of comparisons performed by ExtractMin and Meld in a skew heap, showing that the amortized number of comparisons

is approximately $\lg_\phi n$, where $\phi = (\sqrt{5}+1)/2$ is the golden ratio. The upper bound was given by Kaldewaij and Schoenmakers [91] and the matching lower bound was given by Schoenmakers [115].

Pairing heaps [76] were introduced as a self-adjusting version of Fibonacci heaps, but the exact asymptotic amortized complexity of pairing heaps remains unsettled. Stasko and Vitter [118] did an early experimental evaluation showing that DecreaseKey was virtually constant. Fredman later disproved this by showing a lower bound of amortized time $\Omega(\lg \lg n)$ for the DecreaseKey operation on pairing heaps [74]. Iacono [86] gave an analysis of pairing heaps achieving amortized constant Insert and Meld, and logarithmic ExtractMin and DecreaseKey operations. Pettie [109] proved an upper bound of amortized $O(2^{2\sqrt{\lg \lg n}})$ time for Insert, Meld and DecreaseKey, and amortized $O(\lg n)$ time for ExtractMin.

Variations of pairing heaps were considered in [118, 51, 52], all achieving amortized constant time Insert. Stasko and Vitter [118] achieved that Meld, DecreaseKey, and ExtractMin all take amortized $O(\lg n)$ time. Elmasry in [49] examined parameterized versions of skew heaps, pairing heaps, and skew-pairing heaps, both theoretically and experimentally, and in [51] and [52] showed how to improve the time bound for DecreaseKey to amortized $O(\lg \lg n)$ and the time bounds for Meld to amortized $O(\lg \lg n)$ and amortized $O(1)$, respectively.

## 8 Distribution sensitive priority queues

Priority queues with distribution-sensitive performance have been designed and analyzed (similarly to the working-set properties of splay trees for dictionaries [116]). Fischer and Paterson's fishspear priority queue [64] supports a sequence of Insert and ExtractMin operations, where the amortized cost for handling an element is logarithmic in the "max-depth" of the element, i.e. over time the largest number elements less than the element simultaneously in the priority queue. Iacono [86] proved that for pairing heaps ExtractMin on an element takes amortized logarithmic time in the number of operations performed since the insertion of the element. The funnel-heap of Brodal and Fagerberg [13] achieves ExtractMin logarithmic in the number of insertions performed since the element to be deleted was inserted. Elmasry [50] described a priority queue where ExtractMin takes time logarithmic in the number of elements inserted after the element to be deleted was inserted and are still present in the priority queue. Iacono and Langerman [87] introduced the Queap priority queue where ExtractMin takes time logarithmic in the number of elements inserted *before* the element to be deleted and still present in the priority queue, a property denoted "queueish". Elmasry *et al.* [54] describe a priority queue with a unified property covering both queueish and working set properties.

## 9 RAM priority queues

Priority queues storing non-negative integers and where the running time depends on the maximal possible value $N$ stored in the priority queue were presented by van Emde Boas *et al.* [60, 61], who achieved INSERT and EXTRACTMIN in time $O(\lg\lg N)$ using space $O(N \lg\lg N)$ and $O(N)$ in [61] and [60], respectively. Using dynamic perfect hashing, the Y-fast tries of Willard [124] reduces the space to $O(n)$, by making the time bounds amortized randomized $O(\lg\lg N)$.

Subsequent work initiated by the fusion trees of Fredman and Willard [78] has explored the power of the RAM model to develop algorithms with $o(\lg n)$ time priority queue operations and being independent of the word size $w$ (assuming that elements stored are integers in the range $\{0, 1, \ldots, 2^w - 1\}$). Fusion trees achieve $O(\lg n / \lg\lg n)$ time INSERT and EXTRACTMIN using linear space

Thorup [120] showed how to support INSERT and EXTRACTMIN in $O(\lg\lg n)$ time for $w$-bit integers on a RAM with word size $w$ bits (using superlinear space or linear space using hashing). Linear space deterministic solutions using $O((\lg\lg n)^2)$ amortized and worst-case time were presented by Thorup [119] and Andersson and Thorup [3], respectively. Raman [112] presented a RAM priority queue supporting DECREASEKEY, resulting in an $O(m + n\sqrt{\lg n \lg\lg n})$ time implementation of Dijkstra's single source shortest path algorithm.

That priority queues can be used to sort $n$ numbers is trivial. Thorup in [122] proved that the opposite direction also applies: Given a RAM algorithm that sorts $n$ words in $O(n \cdot S(n))$ time, Thorup describes how to support INSERT and EXTRACTMIN in $O(S(n))$ time, i.e. proving the equivalence between sorting and priority queues. Using previous deterministic $O(n \lg\lg n)$ time and expected $O(n\sqrt{\lg\lg n})$ time RAM sorting algorithms by Han [82] and Han and Thorup [83], respectively, this implies deterministic and randomized priority queues with INSERT and EXTRACTMIN in $O(\lg\lg n)$ and expected $O(\sqrt{\lg\lg n})$ time, respectively. Thorup [121] presented a RAM priority queue supporting INSERT and DECREASEKEY in constant time and EXTRACTMIN in $O(\lg\lg n)$ time, resulting in an $O(m + n \lg\lg n)$ time implementation of Dijkstra's single source shortest path algorithm.

A general technique to convert non-meldable priority queues with INSERT operations taking more than constant time to a corresponding data structure with constant time INSERT operations was presented by Alstrup *et al.* [2]. A general technique was described by Mendelson *et al.* [100] to convert non-meldable priority queues without DECREASEKEY into a priority queue supporting MELD in constant time and with an additive $\alpha(n)$ cost in the time for the DELETE operation, i.e. the operation of deleting an arbitrary element given by a reference. Here $\alpha$ is the inverse of the Ackermann function.

Brodnik *et al.* studied the power of the RAMBO model (random access machine with byte overlap). In [18] they showed how to support INSERT and EXTRACTMIN in constant time (and in [19] they showed how to perform constant time queries and updates for the dynamic prefix sum problem).

## 10 Hierarchical memory models

Early work on algorithm design in the 60s and 70s made the (by then realistic) assumption that running time was bound by the number of instructions performed, and the goal was to construct algorithms minimizing the number of instructions performed. On modern computer architectures the running time of an algorithm implementation is often not dominated by the number of instructions performed, but by other factors such as the number of cache faults, page faults, TLB misses, and branch mispredictions. This has lead to computational models such as the I/O-model of Aggarwal and Vitter [1] and the cache-oblivious model Frigo *et al.* [79], modeling that the bottleneck in a computation is the number of cache-line or disk-block transfers performed by an algorithm. The I/O-model assumes that the parameters $M$ and $B$ are known to the algorithm, where $M$ and $B$ are the capacity in elements of the memory and a disk block, respectively. In the cache-oblivious model the block and memory parameters are not known by the algorithm, with the consequence that a cache-oblivious algorithm with good I/O-performance automatically achieves good I/O-performance on several levels.

Fadel *et al.* [62] described an amortized I/O-optimal priority queue by adopting binary heaps to external memory by letting each node store $\Theta(M)$ elements and the degree of each node be $\Theta(M/B)$. An alternative solution with the same amortized performance was achieved by Arge [4] using a "buffer tree". An external memory priority queue with worst-case bounds matching the previous structures amortized bounds was presented in [15].

Cache-oblivious priority queues were presented by Arge *et al.* [5] and Brodal and Fagerberg [13]. External memory and cache oblivious priority queues supporting an adapted version of DecreaseKey to solve the single source shortest path problem on undirected graphs with $O(n + \frac{m}{B} \lg \frac{m}{B})$ I/Os were presented by Kumar and Swabe [97] and Brodal *et al.* [14], respectively.

Fischer and Paterson [64] introduced the Fishspear priority queue designed for sequential storage such as a constant number of stacks or tapes, and using amortized $O(\lg n)$ time per Insert and ExtractMin operation.

## 11 Priority queues for sorting with limited space

Since the seminal paper by Munro and Patterson [102] on sorting and selection for read-only input memory with a limited read-write working space (and write-only output memory for the case of sorting), a sequence of papers have presented priority queues for sorting in this model. Frederickson [72] achieved a time-space product of $O(n^2 \lg n)$ for sorting, and [107] and [7] achieved an $O(n^2)$ time-space product for a wide-range of working space sizes, which was proven to be optimal by Beame [9].

## 12 Empirical investigations

Many experimental evaluations of priority queues have been done, e.g. [20, 90, 98, 30, 75, 29, 44]. The importance of cache misses were observed in [98], and an implementation adopted to be cache efficient based on merging sorted lists and making efficient use of registers was presented by Sanders [114].

Modern machines are complex and an efficient implementation is not necessarily an implementation performing the fewest possible instructions. As mentioned, other parameters that are important to reduce are e.g. the number of cache misses, number of TLB misses, number of branch mispredictions, and number of branches performed. Memory hierarchy issues can be addressed on the algorithm design level, other issues such as branch mispredictions can be reduced by using special CPU instructions such as predicated instructions such as conditional move instruction (e.g. the CMOV instruction available on the Intel Pentium II and later processors), and exploiting parallelism using e.g. SIMD instructions. Some recent work considering priority queues in this context was done by Edelkamp *et al.* [42].

## 13 Concluding remarks

As stated in the introduction, this paper lists some of the research done related to priority queues, but the list is not expected to be complete. A lot of branches of related work have not been discussed. Examples are: Work on discrete event simulation that makes heavy use of priority queues, and where a lot of work on specialized priority queues has been done; priority queues in parallel models, both practical and theoretical work; concurrency issues for parallel access to a priority queue; results on sorting based on priority queues; just to mention few.

#### Acknowledgment

The author would like to thank Rolf Fagerberg, Andy Brodnik, Jyrki Katajainen, Amr Elmasry, Jesper Asbjørn Sindahl Nielsen and the anonymous reviewers for valuable input.

## References

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM 31(9), 1116–1127 (1988)
2. Alstrup, S., Husfeldt, T., Rauhe, T., Thorup, M.: Black box for constant-time insertion in priority queues (note). ACM Trans. Algorithms 1(1), 102–106 (2005)
3. Andersson, A., Thorup, M.: Tight(er) worst-case bounds on dynamic searching and priority queues. In: Proc. 32nd ACM Symposium on Theory of Computing. pp. 335–342. ACM (2000)
4. Arge, L.: The buffer tree: A technique for designing batched external data structures. Algorithmica 37, 1–24 (2003)

5. Arge, L., Bender, M.A., Demaine, E.D., Holland-Minkley, B., Munro, J.I.: An optimal cache-oblivious priority queue and its application to graph algorithms. SIAM J. Comput. 36(6), 1672–1695 (2007)
6. Arvind, A., Rangan, C.P.: Symmetric min-max heap: A simpler data structure for double-ended priority queue. Inf. Process. Lett. 69(4), 197–199 (1999)
7. Asano, T., Elmasry, A., Katajainen, J.: Priority queues and sorting for read-only data. In: Proc. 10th International Conference Theory and Applications of Models of Computation. Lecture Notes in Computer Science, vol. 7876, pp. 32–41. Springer (2013)
8. Atkinson, M.D., Sack, J.R., Santoro, N., Strothotte, T.: Min-max heaps and generalized priority queues. Commun. ACM 29(10), 996–1000 (1986)
9. Beame, P.: A general sequential time-space tradeoff for finding unique elements. SIAM J. Comput. 20(2), 270–277 (1991)
10. Bollobás, B., Simon, I.: Repeated random insertion into a priority queue. J. Algorithms 6(4), 466–477 (1985)
11. Brodal, G.S.: Fast meldable priority queues. In: Proc. 4th International Workshop Algorithms and Data Structures. Lecture Notes in Computer Science, vol. 955, pp. 282–290. Springer (1995)
12. Brodal, G.S.: Worst-case efficient priority queues. In: Proc. 7th ACM-SIAM Symposium on Discrete Algorithms. pp. 52–58. SIAM (1996)
13. Brodal, G.S., Fagerberg, R.: Funnel heap - a cache oblivious priority queue. In: Proc. 13th International Symposium on Algorithms and Computation, Lecture Notes in Computer Science, vol. 2518, pp. 219–228. Springer (2002)
14. Brodal, G.S., Fagerberg, R., Meyer, U., Zeh, N.: Cache-oblivious data structures and algorithms for undirected breadth-first search and shortest paths. In: Proc. 9th Scandinavian Workshop on Algorithm Theory. Lecture Notes in Computer Science, vol. 3111, pp. 480–492. Springer (2004)
15. Brodal, G.S., Katajainen, J.: Worst-case efficient external-memory priority queues. In: Proc. 6th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science, vol. 1432, pp. 107–118. Springer (1998)
16. Brodal, G.S., Lagogiannis, G., Tarjan, R.E.: Strict Fibonacci heaps. In: Proc. 44th ACM Symposium on Theory of Computing. pp. 1177–1184. ACM (2012)
17. Brodal, G.S., Okasaki, C.: Optimal purely functional priority queues. J. Funct. Program. 6(6), 839–857 (1996)
18. Brodnik, A., Carlsson, S., Fredman, M.L., Karlsson, J., Munro, J.I.: Worst case constant time priority queue. J. Systems and Software 78(3), 249–256 (2005)
19. Brodnik, A., Karlsson, J., Munro, J.I., Nilsson, A.: An $O(1)$ solution to the prefix sum problem on a specialized memory architecture. In: Proc. 4th IFIP International Conference on Theoretical Computer Science (TCS 2006), IFIP 19th World Computer Congress, TC-1 Foundations of Computer Science. vol. 209, pp. 103–114. Springer (2006)
20. Brown, M.R.: Implementation and analysis of binomial queue algorithms. SIAM J. Comput. 7(3), 298–319 (1978)
21. Carlsson, S.: The deap - a double-ended heap to implement double-ended priority queues. Inf. Process. Lett. 26(1), 33–36 (1987)
22. Carlsson, S., Chen, J.: The complexity of heaps. In: Proc. 3rd ACM-SIAM Symposium on Discrete Algorithms. pp. 393–402. SIAM (1992)
23. Carlsson, S., Chen, J., Strothotte, T.: A note on the construction of data structure "deap". Inf. Process. Lett. 31(6), 315–317 (1989)

24. Carlsson, S., Munro, J.I., Poblete, P.V.: An implicit binomial queue with constant insertion time. In: Proc. 1st Scandinavian Workshop on Algorithm Theory. Lecture Notes in Computer Science, vol. 318, pp. 1–13. Springer (1988)
25. Chan, T.M.: Quake heaps: a simple alternative to Fibonacci heaps. Manuscript (2009)
26. Chang, S.C., Du, M.W.: Diamond deque: A simple data structure for priority deques. Inf. Process. Lett. 46(5), 231–237 (1993)
27. Chazelle, B.: A minimum spanning tree algorithm with inverse-ackermann type complexity. J. ACM 47(6), 1028–1047 (2000)
28. Chazelle, B.: The soft heap: an approximate priority queue with optimal error rate. J. ACM 47(6), 1012–1027 (2000)
29. Cherkassky, B.V., Goldberg, A.V., Silverstein, C.: Buckets, heaps, lists, and monotone priority queues. SIAM J. Comput. 28(4), 1326–1346 (1999)
30. Cho, S., Sahni, S.: Weight-biased leftist trees and modified skip lists. ACM J. Experimental Algorithmics 3, 2 (1998)
31. Cho, S., Sahni, S.: Mergeable double-ended priority queues. Int. J. Found. Comput. Sci. 10(1), 1–18 (1999)
32. Chong, K., Sahni, S.: Correspondence-based data structures for double-ended priority queues. ACM J. Experimental Algorithmics 5, 2 (2000)
33. Clancy, M.J., Knuth, D.E.: A programming and problem-solving seminar. Tech. Rep. Technical Report STAN-CS-77-606, Computer Science Department, Stanford University (1977)
34. Crane, C.A.: Linear lists and priority queues as balanced binary trees. Ph.D. thesis, Stanford University, Stanford, CA, USA (1972)
35. Ding, Y., Weiss, M.A.: The K-D heap: An efficient multi-dimensional priority queue. In: Proc. 3rd Workshop on Algorithms and Data Structures. Lecture Notes in Computer Science, vol. 709, pp. 302–313. Springer (1993)
36. Ding, Y., Weiss, M.A.: The relaxed min-max heap. Acta Inf. 30(3), 215–231 (1993)
37. Ding, Y., Weiss, M.A.: On the complexity of building an interval heap. Inf. Process. Lett. 50(3), 143–144 (1994)
38. Doberkat, E.E.: Deleting the root of a heap. Acta Inf. 17, 245–265 (1982)
39. Doberkat, E.E.: An average case analysis of floyd's algorithm to construct heaps. Information and Control 61(2), 114–131 (1984)
40. Driscoll, J.R., Gabow, H.N., Shrairman, R., Tarjan, R.E.: Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation. Commun. ACM 31(11), 1343–1354 (1988)
41. Dutton, R.D.: Weak-heap sort. BIT 33(3), 372–381 (1993)
42. Edelkamp, S., Elmasry, A., Katajainen, J.: A catalogue of algorithms for building weak heaps. In: Proc. 23rd International Workshop on Combinatorial Algorithms. Lecture Notes in Computer Science, vol. 7643, pp. 249–262. Springer (2012)
43. Edelkamp, S., Elmasry, A., Katajainen, J.: The weak-heap data structure: Variants and applications. J. Discrete Algorithms 16, 187–205 (2012)
44. Edelkamp, S., Elmasry, A., Katajainen, J.: The weak-heap family of priority queues in theory and praxis. In: Proc. 18th Computing: The Australasian Theory Symposium. CRPIT, vol. 128, pp. 103–112. Australian Computer Society (2012)
45. Edelkamp, S., Elmasry, A., Katajainen, J.: Ultimate binary heaps (2013), submitted
46. Edelkamp, S., Stiegeler, P.: Implementing HEAPSORT with $(n \log n - 0.9n)$ and QUICKSORT with $(n \log n + 0.2n)$ comparisons. ACM J. Experimental Algorithmics 7, 5–24 (2002)

47. Edelkamp, S., Wegener, I.: On the performance of weak-heapsort. In: Proc. 17th Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 1770, pp. 254–266. Springer (2000)
48. Elmasry, A.: Layered heaps. In: Proc. 9th Scandinavian Workshop on Algorithm Theory. Lecture Notes in Computer Science, vol. 3111, pp. 212–222. Springer (2004)
49. Elmasry, A.: Parameterized self-adjusting heaps. J. Algorithms 52(2), 103–119 (2004)
50. Elmasry, A.: A priority queue with the working-set property. Int. J. Found. Comput. Sci. 17(6), 1455–1466 (2006)
51. Elmasry, A.: Pairing heaps with $O(\log \log n)$ decrease cost. In: Proc. 20th ACM-SIAM Symposium on Discrete Algorithms. pp. 471–476. SIAM (2009)
52. Elmasry, A.: Pairing heaps with costless meld. In: Proc. 18th European Symposium on Algorithms, Part II. Lecture Notes in Computer Science, vol. 6347, pp. 183–193. Springer (2010)
53. Elmasry, A.: The violation heap: A relaxed Fibonacci-like heap. In: Proc. 16th International Conference on Computing and Combinatorics. Lecture Notes in Computer Science, vol. 6196, pp. 479–488. Springer (2010)
54. Elmasry, A., Farzan, A., Iacono, J.: A priority queue with the time-finger property. J. Discrete Algorithms 16, 206–212 (2012)
55. Elmasry, A., Jensen, C., Katajainen, J.: On the power of structural violations in priority queues. In: Proc. 13th Computing: The Australasian Theory Symposium. CRPIT, vol. 65, pp. 45–53. Australian Computer Society (2007)
56. Elmasry, A., Jensen, C., Katajainen, J.: Multipartite priority queues. ACM Trans. Algorithms 5(1) (2008)
57. Elmasry, A., Jensen, C., Katajainen, J.: Two new methods for constructing double-ended priority queues from priority queues. Computing 83(4), 193–204 (2008)
58. Elmasry, A., Jensen, C., Katajainen, J.: Two-tier relaxed heaps. Acta Inf. 45(3), 193–210 (2008)
59. Elmasry, A., Katajainen, J.: Worst-case optimal priority queues via extended regular counters. In: Proc. 7th International Computer Science Symposium in Russia Computer Science - Theory and Applications. Lecture Notes in Computer Science, vol. 7353, pp. 125–137. Springer (2012)
60. van Emde Boas, P.: Preserving order in a forest in less than logarithmic time and linear space. Inf. Process. Lett. 6(3), 80–82 (1977)
61. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. Mathematical Systems Theory 10, 99–127 (1977)
62. Fadel, R., Jakobsen, K.V., Katajainen, J., Teuhola, J.: Heaps and heapsort on secondary storage. Theoretical Computer Science 220(2), 345–362 (1999)
63. Fagerberg, R.: A generalization of binomial queues. Inf. Process. Lett. 57(2), 109–114 (1996)
64. Fischer, M.J., Paterson, M.: Fishspear: A priority queue algorithm. J. ACM 41(1), 3–30 (1994)
65. Floyd, R.W.: Algorithm 113: Treesort. Commun. ACM 5(8), 434 (1962)
66. Floyd, R.W.: Algorithm 245: Treesort3. Commun. ACM 7(12), 701 (1964)
67. Ford, Lester R., J., Johnson, S.M.: A tournament problem. The American Mathematical Monthly 66(5), 387–389 (1959)
68. Franceschini, G., Grossi, R.: Optimal worst-case operations for implicit cache-oblivious search trees. In: Proc. 8th International Workshop on Algorithms and

Data Structures. Lecture Notes in Computer Science, vol. 2748, pp. 114–126. Springer (2003)
69. Franceschini, G., Grossi, R., Munro, J.I., Pagli, L.: Implicit B-trees: a new data structure for the dictionary problem. J. Comput. Syst. Sci. 68(4), 788–807 (2004)
70. Franceschini, G., Munro, J.I.: Implicit dictionaries with $O(1)$ modifications per update and fast search. In: Proc. 17th ACM-SIAM Symposium on Discrete Algorithm. pp. 404–413. SIAM (2006)
71. Frederickson, G.N.: Implicit data structures for the dictionary problem. J. ACM 30(1), 80–94 (1983)
72. Frederickson, G.N.: Upper bounds for time-space trade-offs in sorting and selection. J. Comput. Syst. Sci. 34(1), 19–26 (1987)
73. Frederickson, G.N.: An optimal algorithm for selection in a min-heap. Inf. Comput. 104(2), 197–214 (1993)
74. Fredman, M.L.: On the efficiency of pairing heaps and related data structures. J. ACM 46(4), 473–501 (1999)
75. Fredman, M.L.: A priority queue transform. In: Proc. 3rd International Workshop on Algorithm Engineering. Lecture Notes in Computer Science, vol. 1668, pp. 244–258. Springer (1999)
76. Fredman, M.L., Sedgewick, R., Sleator, D.D., Tarjan, R.E.: The pairing heap: A new form of self-adjusting heap. Algorithmica 1(1), 111–129 (1986)
77. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM 34(3), 596–615 (1987)
78. Fredman, M.L., Willard, D.E.: Surpassing the information theoretic bound with fusion trees. J. Comput. Syst. Sci. 47(3), 424–436 (1993)
79. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. 40th Foundations of Computer Science. pp. 285–297. IEEE (1999)
80. Gonnet, G.H., Munro, J.I.: Heaps on heaps. SIAM J. Comput. 15(4), 964–971 (1986)
81. Haeupler, B., Sen, S., Tarjan, R.E.: Rank-pairing heaps. SIAM J. Comput. 40(6), 1463–1485 (2011)
82. Han, Y.: Deterministic sorting in $O(n \log\log n)$ time and linear space. In: Proc. 34th ACM Symposium on Theory of Computing. pp. 602–608. ACM (2002)
83. Han, Y., Thorup, M.: Integer sorting in $O(n\sqrt{\log\log n})$ expected time and linear space. In: Proc. 43rd Foundations of Computer Science. pp. 135–144. IEEE (2002)
84. Harvey, N.J.A., Zatloukal, K.C.: The post-order heap. In: Proc. 3rd International Conference on Fun with Algorithms (2004)
85. Høyer, P.: A general technique for implementation of efficient priority queues. In: Proc. 3rd Israel Symposium on Theory of Computing and Systems. pp. 57–66. IEEE (1995)
86. Iacono, J.: Improved upper bounds for pairing heaps. In: Proc. 7th Scandinavian Workshop on Algorithm Theory. Lecture Notes in Computer Science, vol. 1851, pp. 32–45. Springer (2000)
87. Iacono, J., Langerman, S.: Queaps. Algorithmica 42(1), 49–56 (2005)
88. Johnson, D.B.: Priority queues with update and finding minimum spanning trees. Inf. Process. Lett. 4(3), 53–57 (1975)
89. Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. J. ACM 24(1), 1–13 (1977)
90. Jones, D.W.: An empirical comparison of priority-queue and event-set implementations. Commun. ACM 29(4), 300–311 (1986)

91. Kaldewaij, A., Schoenmakers, B.: The derivation of a tighter bound for top-down skew heaps. Inf. Process. Lett. 37(5), 265–271 (1991)
92. Kaplan, H., Shafrir, N., Tarjan, R.E.: Meldable heaps and boolean union-find. In: Proc. 34th ACM Symposium on Theory of Computing. pp. 573–582. ACM (2002)
93. Kaplan, H., Tarjan, R.E.: New heap data structures. Tech. Rep. TR-597-99, Department of Computer Science, Princeton University (1999)
94. Kaplan, H., Tarjan, R.E.: Thin heaps, thick heaps. ACM Trans. Algorithms 4(1) (2008)
95. Kaplan, H., Zwick, U.: A simpler implementation and analysis of chazelle's soft heaps. In: Proc. 20th ACM-SIAM Symposium on Discrete Algorithms. pp. 477–485. SIAM (2009)
96. Khoong, C.M., Leong, H.W.: Double-ended binomial queues. In: Proc. 4th International Symposium on Algorithms and Computation. Lecture Notes in Computer Science, vol. 762, pp. 128–137. Springer (1993)
97. Kumar, V., Schwabe, E.J.: Improved algorithms and data structures for solving graph problems in external memory. In: Proc. 8th Symposium on Parallel and Distributed Processing. pp. 169–177. IEEE (1996)
98. LaMarca, A., Ladner, R.E.: The influence of caches on the performance of heaps. ACM J. Experimental Algorithmics 1, 4 (1996)
99. van Leeuwen, J., Wood, D.: Interval heaps. Comput. J. 36(3), 209–216 (1993)
100. Mendelson, R., Tarjan, R.E., Thorup, M., Zwick, U.: Melding priority queues. ACM Trans. Algorithms 2(4), 535–556 (2006)
101. Munro, J.I.: An implicit data structure supporting insertion, deletion, and search in $O(\log^2 n)$ time. J. Comput. Syst. Sci. 33(1), 66–74 (1986)
102. Munro, J.I., Paterson, M.: Selection and sorting with limited storage. Theoretical Computer Science 12, 315–323 (1980)
103. Munro, J.I., Poblete, P.V.: Searchability in merging and implicit data structures. BIT 27(3), 324–329 (1987)
104. Munro, J.I., Suwanda, H.: Implicit data structures for fast search and update. J. Comput. Syst. Sci. 21(2), 236–250 (1980)
105. Okasaki, C.: Alternatives to two classic data structures. In: Proc. 36th SIGCSE Technical Symposium on Computer Science Education. pp. 162–165. ACM (2005)
106. Olariu, S., Overstreet, C.M., Wen, Z.: A mergeable double-ended priority queue. Comput. J. 34(5), 423–427 (1991)
107. Pagter, J., Rauhe, T.: Optimal time-space trade-offs for sorting. In: Proc. 39th Foundations of Computer Science. pp. 264–268. IEEE (1998)
108. Peterson, G.L.: A balanced tree scheme for meldable heaps with updates. Tech. Rep. GIT-ICS-87-23, School of Informatics and Computer Science, Georgia Institute of Technology (1987)
109. Pettie, S.: Towards a final analysis of pairing heaps. In: Proc. 46th Foundations of Computer Science. pp. 174–183. IEEE (2005)
110. Pettie, S., Ramachandran, V.: An optimal minimum spanning tree algorithm. J. ACM 49(1), 16–34 (2002)
111. Porter, T., Simon, I.: Random insertion into a priority queue structure. IEEE Trans. Software Eng. 1(3), 292–298 (1975)
112. Raman, R.: Priority queues: Small, monotone and trans-dichotomous. In: Proc. 4th European Symposium on Algorithms. Lecture Notes in Computer Science, vol. 1136, pp. 121–137. Springer (1996)
113. Sack, J.R., Strothotte, T.: An algorithm for merging heaps. Acta Inf. 22(2), 171–186 (1985)

114. Sanders, P.: Fast priority queues for cached memory. ACM J. Experimental Algorithmics 5, 7 (2000)
115. Schoenmakers, B.: A tight lower bound for top-down skew heaps. Inf. Process. Lett. 61(5), 279–284 (1997)
116. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM 32(3), 652–686 (1985)
117. Sleator, D.D., Tarjan, R.E.: Self-adjusting heaps. SIAM J. Comput. 15(1), 52–69 (1986)
118. Stasko, J.T., Vitter, J.S.: Pairing heaps: experiments and analysis. Commun. ACM 30(3), 234–249 (1987)
119. Thorup, M.: Faster deterministic sorting and priority queues in linear space. In: Proc. 9th ACM-SIAM Symposium on Discrete Algorithms. pp. 550–555. SIAM (1998)
120. Thorup, M.: On RAM priority queues. SIAM J. Comput. 30(1), 86–109 (2000)
121. Thorup, M.: Integer priority queues with decrease key in constant time and the single source shortest paths problem. J. Comput. Syst. Sci. 69(3), 330–353 (2004)
122. Thorup, M.: Equivalence between priority queues and sorting. J. ACM 54(6) (2007)
123. Vuillemin, J.: A data structure for manipulating priority queues. Commun. ACM 21(4), 309–315 (1978)
124. Willard, D.E.: Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. Inf. Process. Lett. 17(2), 81–84 (1983)
125. Williams, J.W.J.: Algorithm 232: Heapsort. Commun. ACM 7(6), 347–348 (1964)