On The Construction of Priority Queues with Good Worst Case Performance

Gerth S. Brodal
gerth@daimi.aau.dk

BRICS
Computer Science Department
University of Aarhus
Aarhus, Denmark

August 1995
Priority Queue Operations

- **MakeQueue**
- **FindMin**(Q)
- **Insert**(Q, e)
- **Meld**(Q₁, Q₂)
- **DeleteMin**(Q)
- **Delete**(Q, e)*
- **DecreaseKey**(Q, e, e'), e' ≤ e

*Assumes that it is known where the element e is stored in Q.*
Optimality?

Theorem:
If MELD can be performed in worst case time \( o(n) \) then DELETEMIN cannot be performed in worst case time \( o(\log n) \).

Proof:
For \( n = 2^k \) we otherwise by contradiction get

\[
T_{\text{Sorting}}(n) = nT_{\text{MAKEQUEUE}} + \sum_{i=0}^{k-1} 2^{k-1-i} T_{\text{MELD}}(2^i) + \sum_{i=1}^{n} T_{\text{DELETEMIN}}(i)
\]

\[= o(n \log n).\]
### Priority Queues without \texttt{DECREASE KEY}

<table>
<thead>
<tr>
<th></th>
<th>[W64]</th>
<th>[SS85]</th>
<th>[DGST88]</th>
<th>[V78]</th>
<th>[B95a]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Heaps</td>
<td>Merging Heaps</td>
<td>Relaxed Heaps</td>
<td>Binomial Queues*</td>
<td>New Result</td>
</tr>
<tr>
<td>\texttt{FINDMIN}</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>\texttt{INSERT}</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>\texttt{MELD}</td>
<td>O(n)</td>
<td>O(log^2 n)</td>
<td>O(log n)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>\texttt{DELETE(MIN)}</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(log n)</td>
</tr>
</tbody>
</table>

*Amortised bounds*
A priority queue is represented by a heap ordered tree where each node contains an element and has a rank assigned.

A node of rank $r$ has at most one son of type I and one, two or three sons of type II of rank $i$ for $i = 0, \ldots, r - 1$. 
Two trees of equal rank $r$ can be linked to one tree of rank $r + 1$ in worst case time $O(1)$.
Linking Example

Priority Queues with Good Worst Case Performance

Gerth S. Brodal
The Invariant

Between two ranks where three sons of type II have equal rank there is a rank of which there only is one son of type II.
**Implementation of MELD**

How to perform MELD in worst case time $O(1)$.

The case $e_1 \leq e_2 < e'_1 \leq e'_2$. 

Gerth S. Brodal
Implementation of **DELETEMIN**

How to perform **DELETEMIN** in worst case time $O(\log n)$.

$e_1, e_2$ and $e_3$ are the three smallest elements.
Each node is stored as a record having seven fields.
The Priority Queues Described...

- support **MAKEQUEUE**, **FINDMIN**, **INSERT** and **MELD** in worst case time $O(1)$,
- support **DELETE(MIN)** in worst case time $O(\log n)$,
- require linear space and
- can be implemented on a pointer machine.
- But they are **not purely functional**.
A Purely Functional Priority Queue
Joint work with Chris Okasaki

Purely functional priority queues support only the operations:

- **MAKEQUEUE**
- **FINDMIN**(Q)
- **INSERT**(Q, e)
- **MELD**(Q₁, Q₂)
- **DELETEMIN**(Q)

and can be implemented without using side effects
i.e. fully persistent priority queues.
A pure priority queue is represented by a heap ordered tree where each node has three fields: an element, a rank and a list of sons.

- The subtree rooted at a node $x$ has size $\Omega(2^{rank(x)})$.
- All nodes have degree $O(\log n)$.
- Arrangement of sons:

\[ \begin{array}{cccccccc}
4 & 3 & 0 & 2 & 0 & 1 & 1 & 3 \\
& 3 & 3 & 5 & 6 & 8 & 9 & \\
\end{array} \]

- Can have equal rank
- Zero and decreasing rank
- Increasing rank

- The root has rank zero.
A Purely Functional Priority Queue
— Operations —

- **MELD:**

- **DELETEMIN:**

- **MELD** requires time $O(1)$ and **DELETEMIN** time $O(\log n)$. 

Gerth S. Brodal
# Priority Queues with DecreaseKey

<table>
<thead>
<tr>
<th></th>
<th>[W64]</th>
<th>[DGST88]</th>
<th>[FT84]</th>
<th>[B95a]</th>
<th>[B95b]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Heaps</td>
<td>Relaxed Heaps</td>
<td>Fibonacci Queues*</td>
<td>New Result</td>
<td>New Result</td>
</tr>
<tr>
<td><strong>FindMin</strong></td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td><strong>Insert</strong></td>
<td>O(log n)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td><strong>Meld</strong></td>
<td>O(n)</td>
<td>O(log n)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td><strong>Delete(Min)</strong></td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(log n)</td>
</tr>
<tr>
<td><strong>DecreaseKey</strong></td>
<td>O(log n)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(log n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

*Amortised bounds

Gerth S. Brodal
Relaxed Heaps [DGST88]

- First data structure supporting \textsc{DecreaseKey} in worst case time $O(1)$. \textsc{Meld} requires time $O(\log n)$.

- Basic idea:

\begin{itemize}
  \item \textsc{DecreaseKey} creates at most one new active node.
  \item Transformations can eliminate $O(1)$ active nodes in worst case time $O(1)$.
\end{itemize}
**DecreaseKey and Meld in time O(1) [B95b]**

- Time bounds achieved:
  - FindMin, Insert, Meld, DecreaseKey in time $O(1)$ and
  - Delete(Min) in time $O(\log n)$.

- Basic idea:
  - To each node is associated a set of active nodes.
  - The active nodes associated to a node are larger than the node.
  - An active node belongs to at most one set.
The Sons of a Node

- Each node has a rank.
- The rank of a node is less than the rank of its father.
- All nodes except for the roots have a brother of equal rank.
- Leaves have rank zero. A node of rank $r \geq 1$ has at least two sons of rank $r - 1$.
- At most $O(1)$ brothers can have equal rank.
How to Reduce The Number of Active Nodes

Two active brothers of equal rank can be made good by creating at most one new active node by the following transformations:

- The two brothers are cut off and made good sons of the root of $T_1$.

- The two brothers and the father are cut off and made good sons of the root of $T_1$. The replacement becomes a new active node.
The basic idea behind the implementation of MELD is the linking below. At most two new active nodes are created.
Open problems

• Simplify the data structures.

• Construct efficiently purely functional priority queues supporting \texttt{DELETE} and \texttt{DECREASE\textsc{Key}}.

• Is it possible to support the operations in the presented time bounds without requiring the RAM model?

• How does the lower bound on \texttt{DELETE\textsc{Min}} relate to \texttt{DELETE} and \texttt{FIND\textsc{Min}}?