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(Q_1, Q_2)
- DeleteMin(Q)
- Delete(Q, e)*
- DecreaseKey(Q, e, e')*, $\quad e' \leq e$

*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

$$\begin{aligned} & T_{\text{SORTING}}(n) \\ & \quad = nT_{\text{MAKEQUEUE}} + \sum_{i=0}^{k-1} 2^{k-1-i} T_{\text{MELD}}(2^i) + \sum_{i=1}^{n} T_{\text{DELETEMIN}}(i) \\ & \quad = o(n \log n). \end{aligned}$$

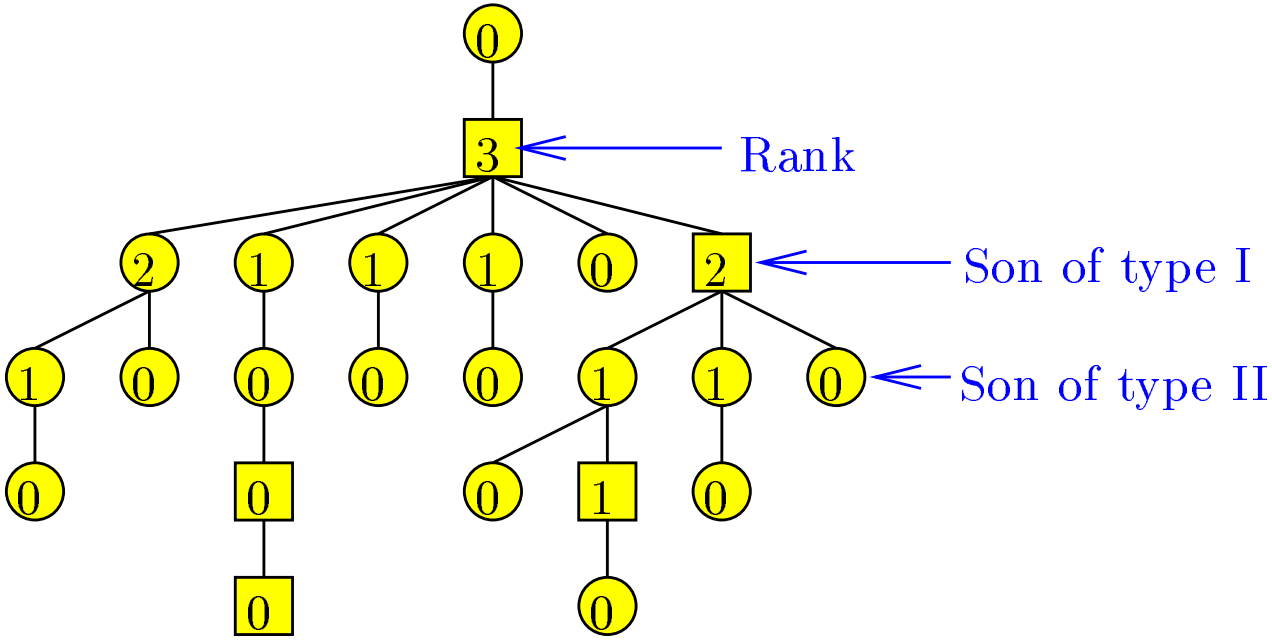
□

Priority Queues without DecreaseKey

	[W64] Heaps	[SS85] Merging Heaps	[DGST88] Relaxed Heaps	[V78] Binomial Queues*	[B95a] New Result
FindMin	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Insert	$O(\log n)$	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$
Meld	$O(n)$	$O(\log^2 n)$	$O(\log n)$	$O(1)$	$O(1)$
Delete(Min)	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

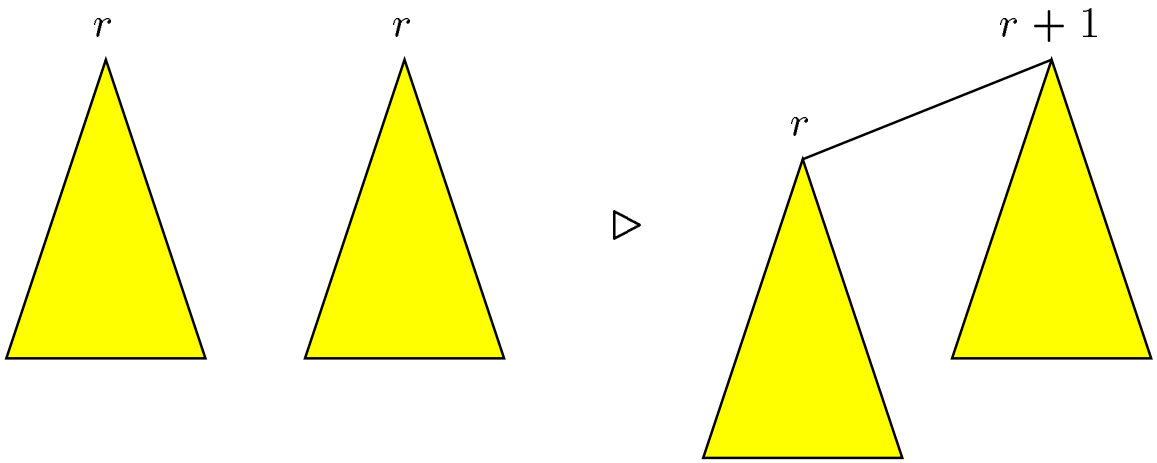
*Amortised bounds

The Data Structure



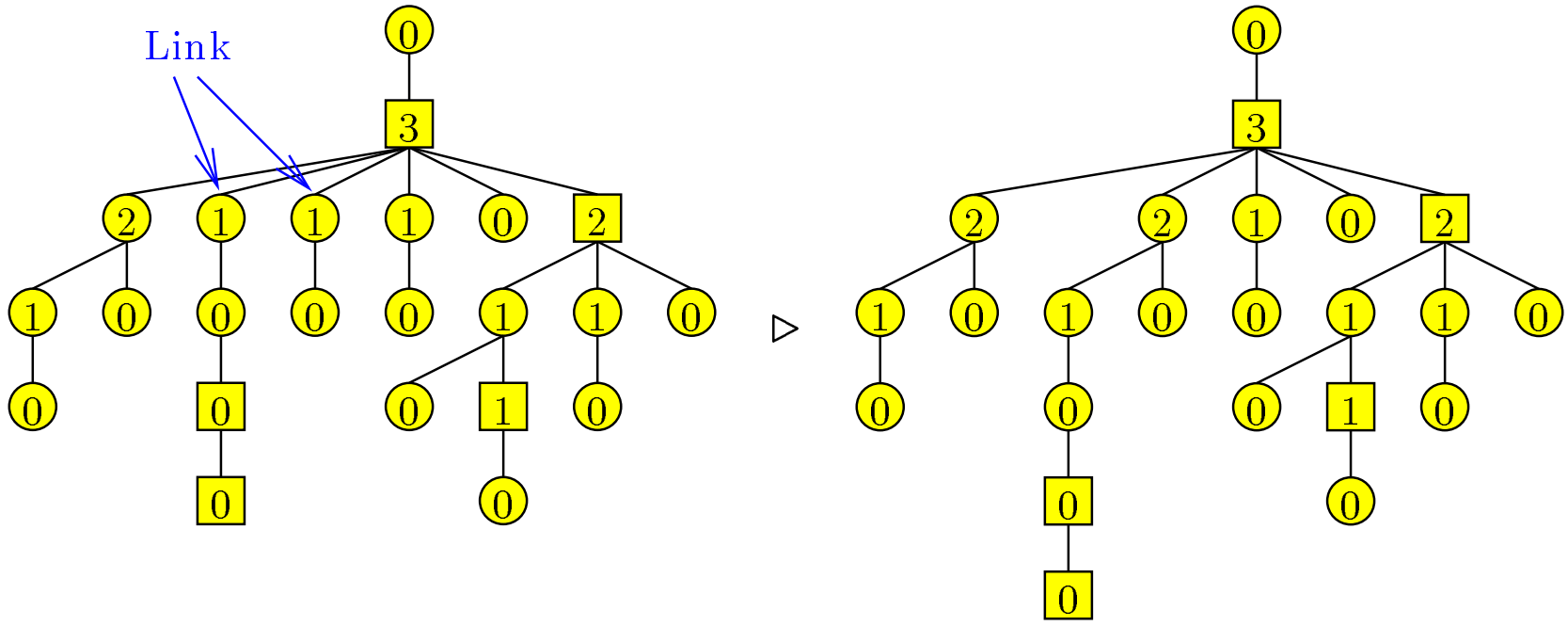
- A priority queue is represented by a heap ordered tree where each node contains an element and has a rank assigned.
- A node of rank r has at most one son of type I and one, two or three sons of type II of rank i for $i = 0, \ldots, r-1$.

Linking

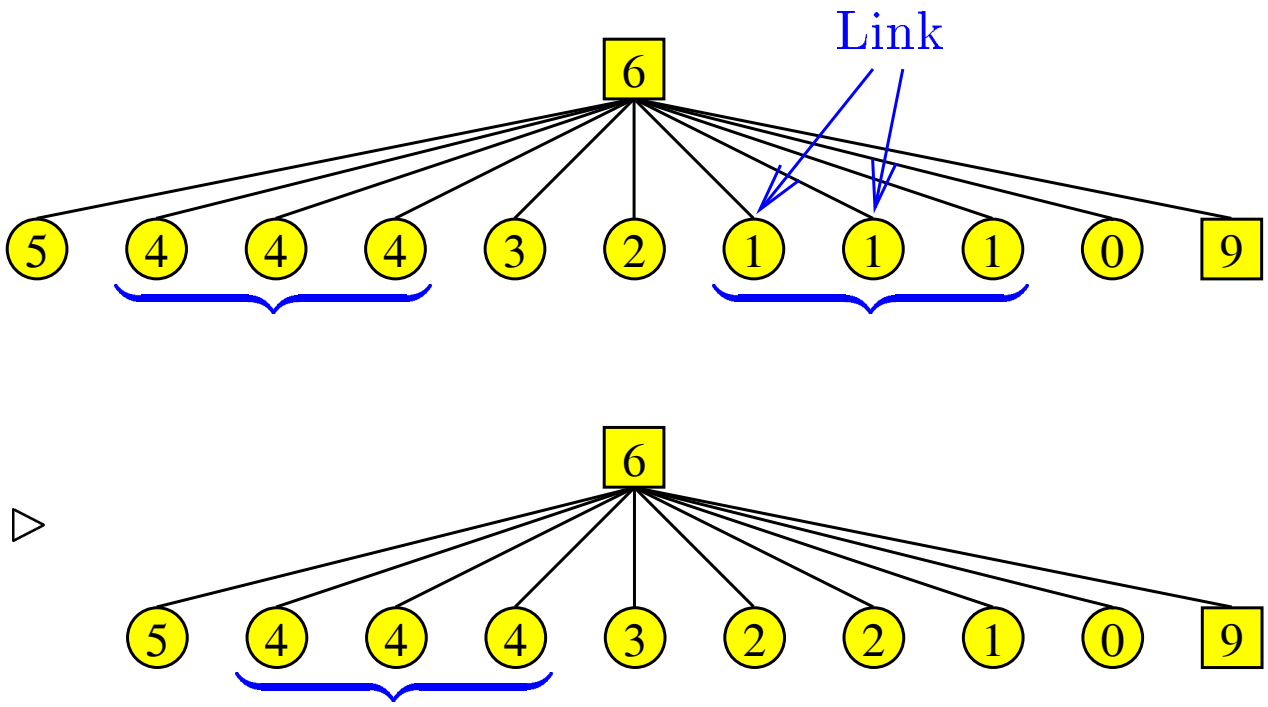


Two trees of equal rank r can be linked to one tree of rank $r + 1$ in worst case time O(1).

Linking Example

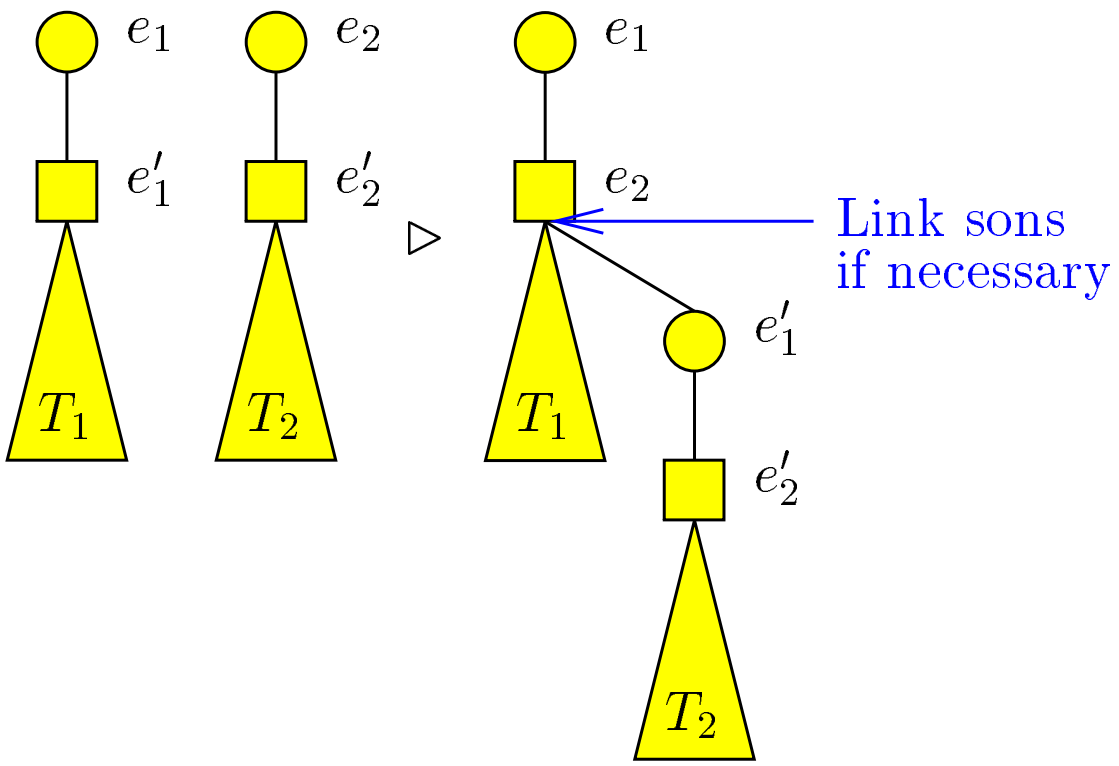


The Invariant



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.

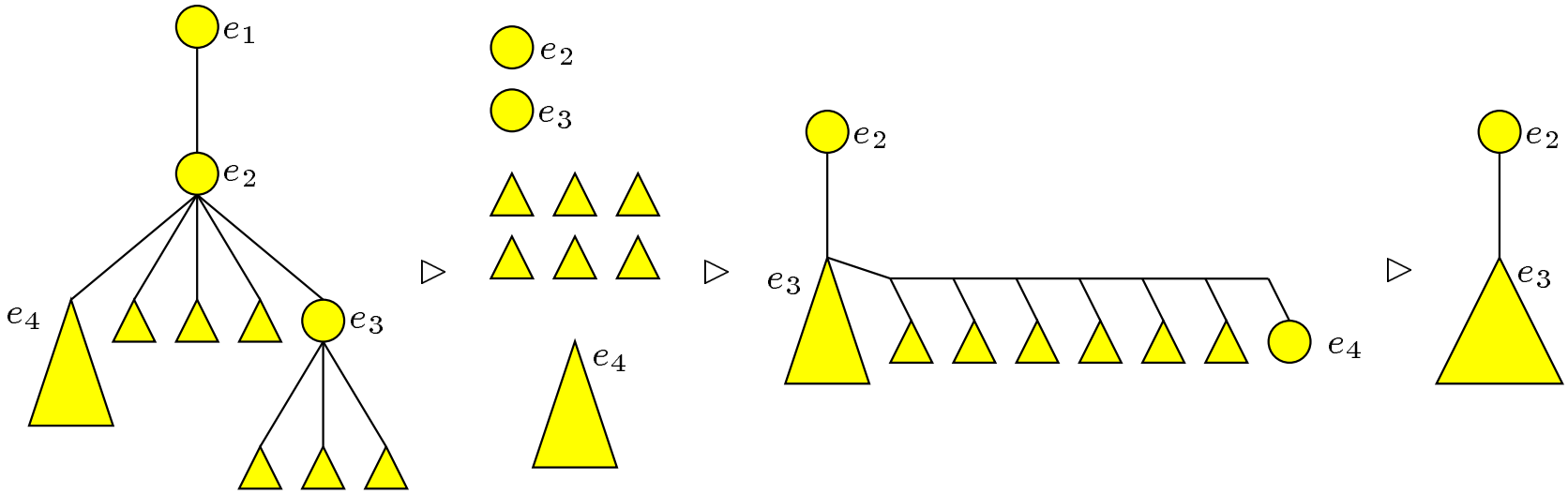
Implementation of MELD



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

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

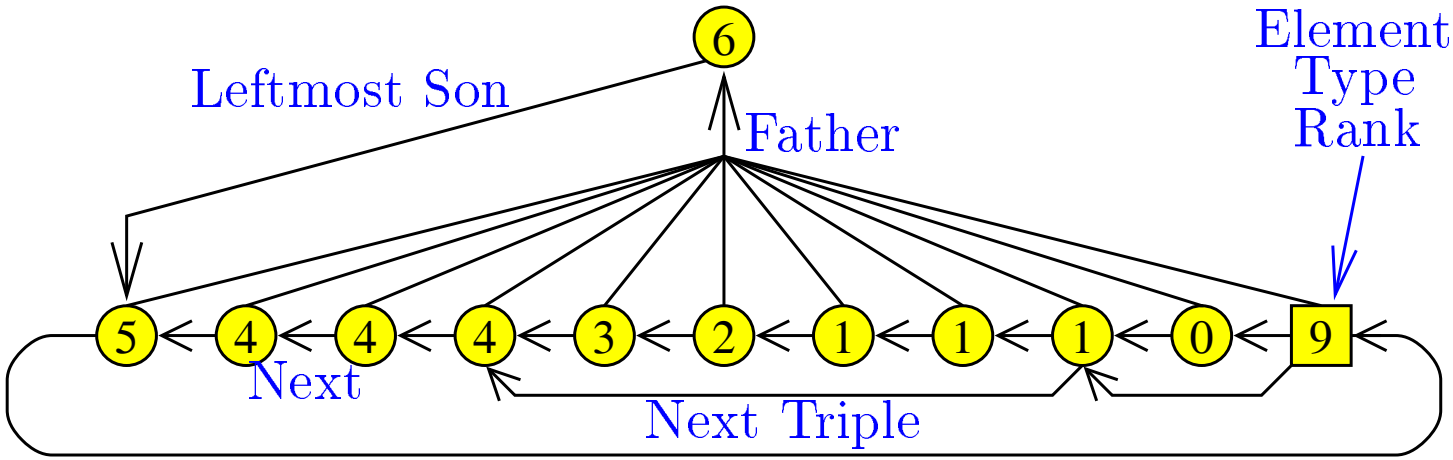
Implementation of DeleteMin



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

e_1, e_2 and e_3 are the three smallest elements.

Implementation Details



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

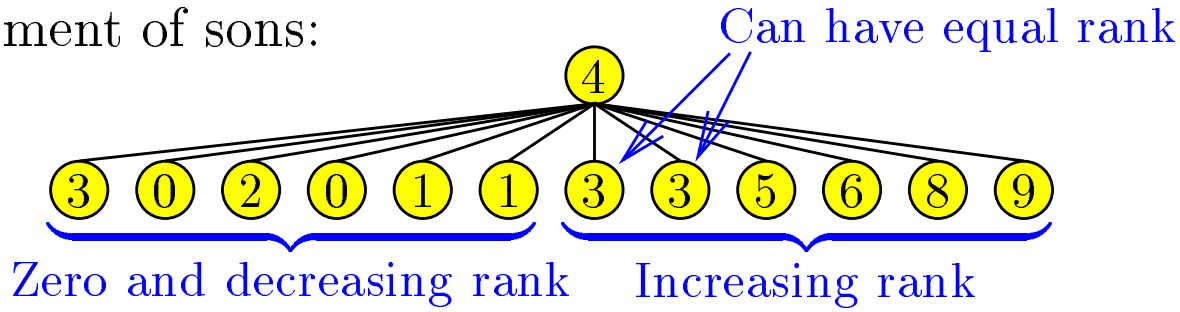
Purely functional priority queues support only the operations:

- MakeQueue
- FindMin(Q)
- Insert(Q, e)
- Meld(Q_1, Q_2)
- DeleteMin(Q)

and can be implemented without using side effects
i.e. fully persistent priority queues.

A Purely Functional Priority Queue

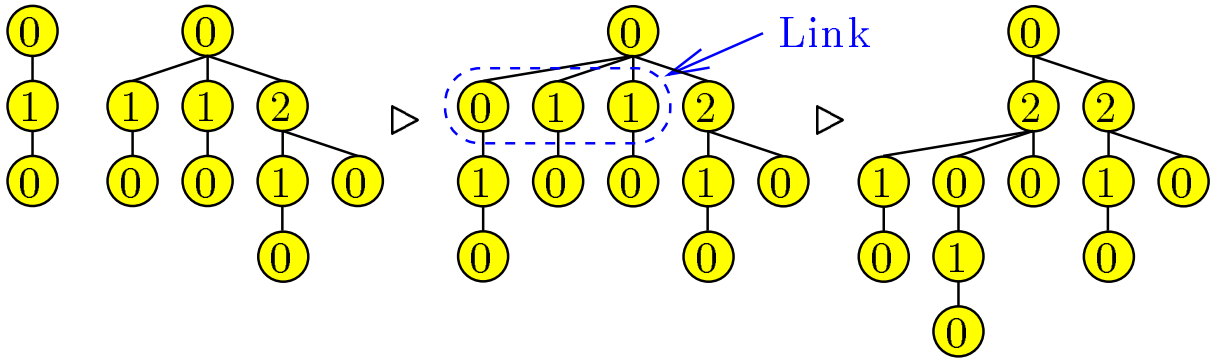
- A priority queue is represented by a heap ordered tree where each node has three fields: an element, a rank and a list of sons.
- The subtree rooted at a node x has size $\Omega(2^{rank(x)})$.
- All nodes have degree $O(\log n)$.
- Arrangement of sons:



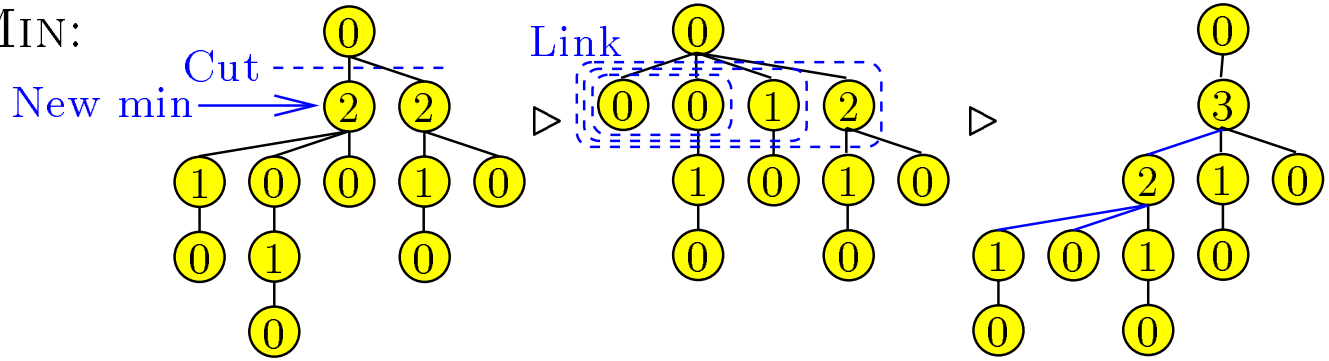
- The root has rank zero.

A Purely Functional Priority Queue
— Operations —

- MELD:



- DELETEMIN:



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

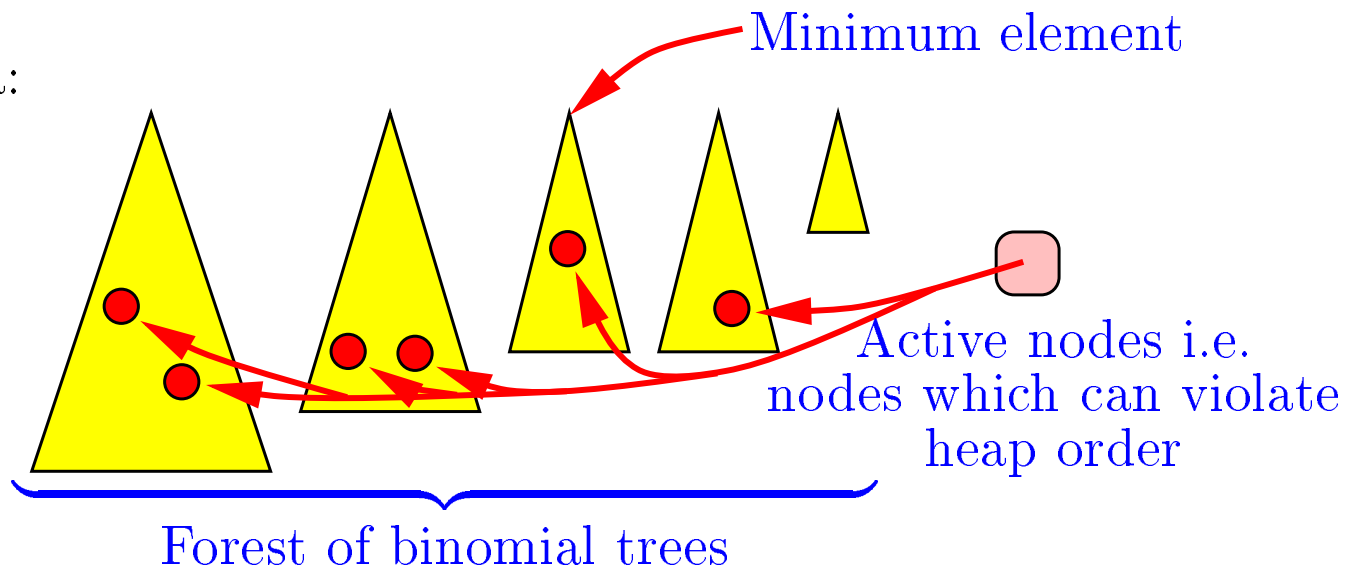
Priority Queues with DecreaseKey

	[W64] Heaps	[DGST88] Relaxed Heaps	[FT84] Fibonacci Queues*	[B95a] New Result	[B95b] New Result
FindMin	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Insert	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Meld	$O(n)$	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$
Delete(Min)	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
DecreaseKey	$O(\log n)$	$O(1)$	$O(1)$	$O(\log n)$	$O(1)$

*Amortised bounds

Relaxed Heaps [DGST88]

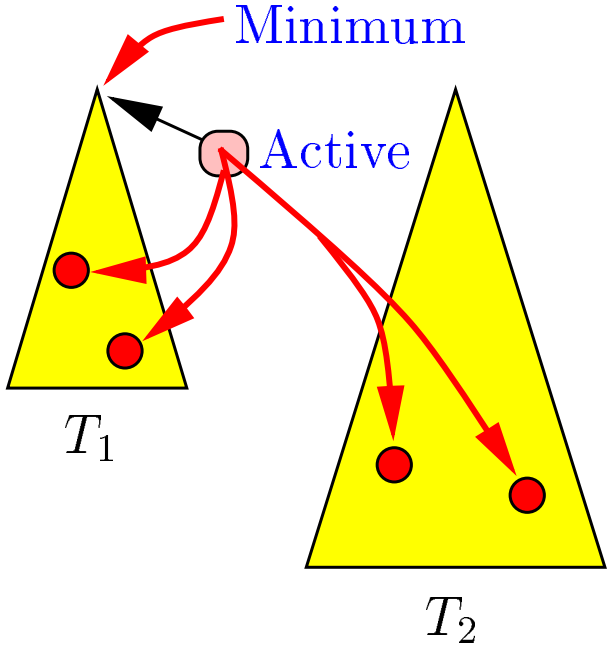
- First data structure supporting DecreaseKey in worst case time O(1). Meld requires time O($\log n$).
- Basic idea:



- DecreaseKey creates at most one new active node.
- Transformations can eliminate O(1) active nodes in worst case time O(1).

DecreaseKey and Meld in time O(1) [B95b]

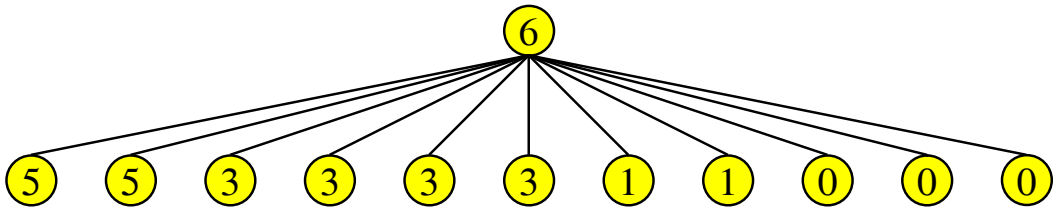
- Time bounds achieved:
  - FindMin, Insert, Meld, DecreaseKey in time $O(1)$ and
  - Delete(Min) in time $O(\log n)$.
- Basic idea:



  - 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

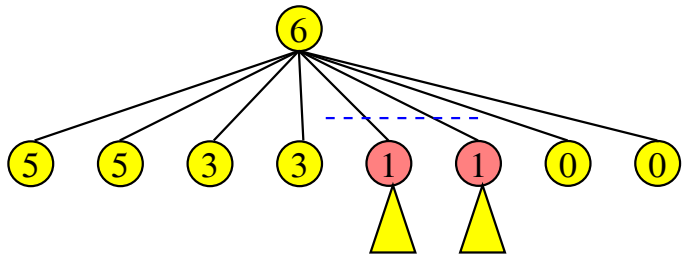
- Each node has a rank.
- The rank of a node is less than the rank of its father.
- All nodes except for the roots have a brother of equal rank.
- Leaves have rank zero. A node of rank $r \geq 1$ has at least two sons of rank $r - 1$.
- At most O(1) brothers can have equal rank.



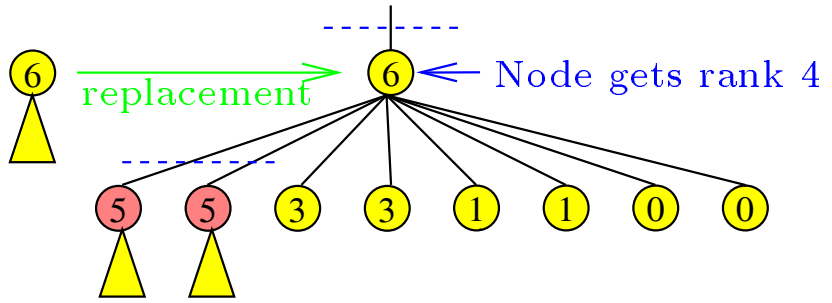
How to Reduce The Number of Active Nodes

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.

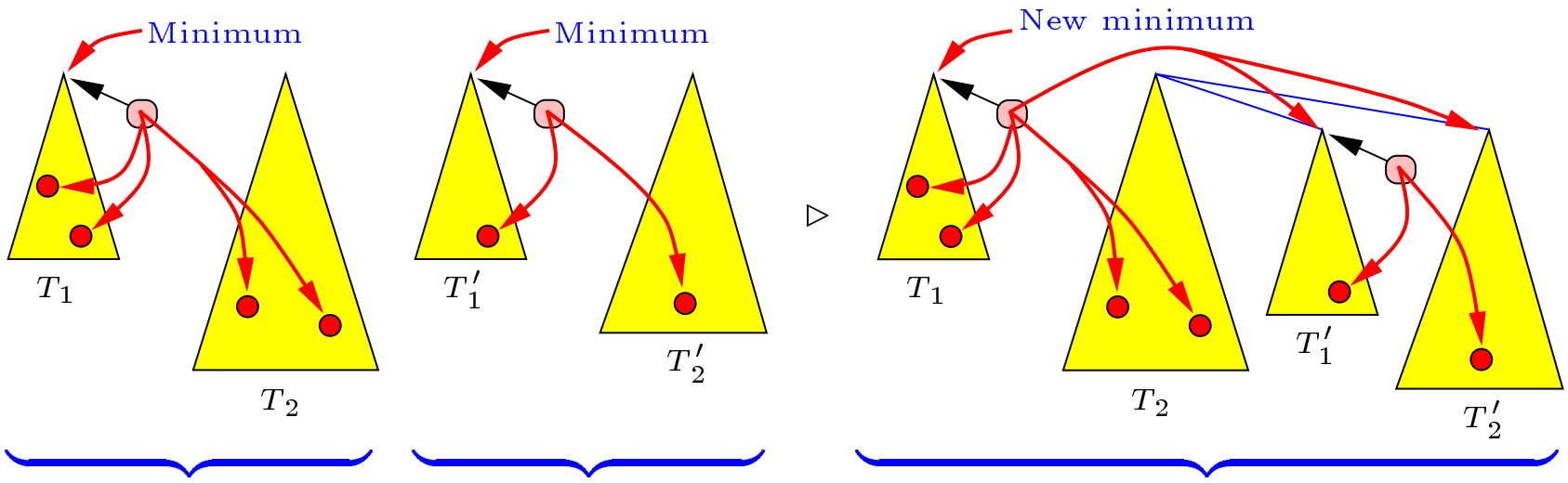


- The two brothers and the father are cut off and made good sons of the root of T_1. The replacement becomes a new active node.



Meld

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 Delete and DecreaseKey.
- Is it possible to support the operations in the presented time bounds without requiring the RAM model?
- How does the lower bound on DeleteMin relate to Delete and FindMin?