Two-tier relaxed heaps

Presented by Claus Jensen
Joint work with Amr Elmasry and Jyrki Katajainen
Slides available at
www.cphstl.dk

Heaps

- Our focus has been on the worst-case comparison complexity of the heap operations:
  - Find-min
  - Insert
  - Decrease(-key)
  - Delete
- The following heaps support decrease at O(1) cost in the worst case:
  - Run-relaxed heaps (Driscoll et al. 1988)
  - Fat heaps (Kaplan and Tarjan 1999)

Worst-case complexity bounds

- The number of element comparisons performed in the worst case

	Run-relaxed heaps	Fat heaps
Find-min	O(1)	O(1)
Insert	O(1)	O(1)
Decrease	O(1)	O(1)
Delete	3lg n + O(1)	2.6lg n + O(1)

Our two-tier relaxed heaps

- A two-tier relaxed heap has the following complexity bounds.

	Number of element comparisons	Worst-case cost
Find-min	O(1)	O(1)
Insert	O(1)	O(1)
Decrease	O(1)	O(1)
Delete	$\lg n + 3\lg\lg n + O(1)$	O(lg n)

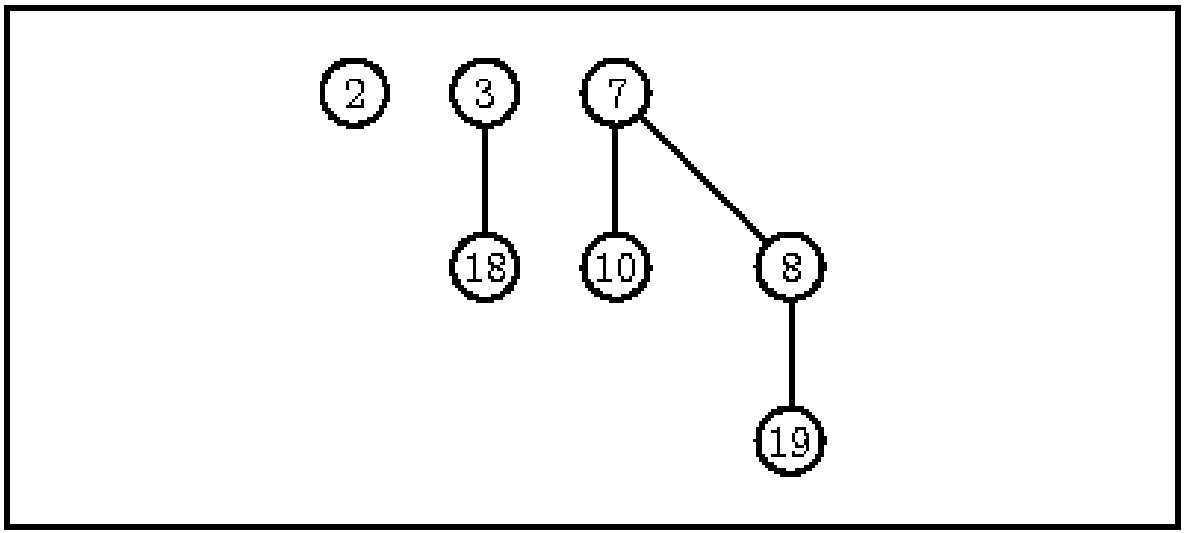
Two-tier relaxed heaps

- A two-tier relaxed heap is composed of two run-relaxed heaps relying on a redundant zeroless representation

- Next: Number representations vs. data structures

A binomial heap using binary representation

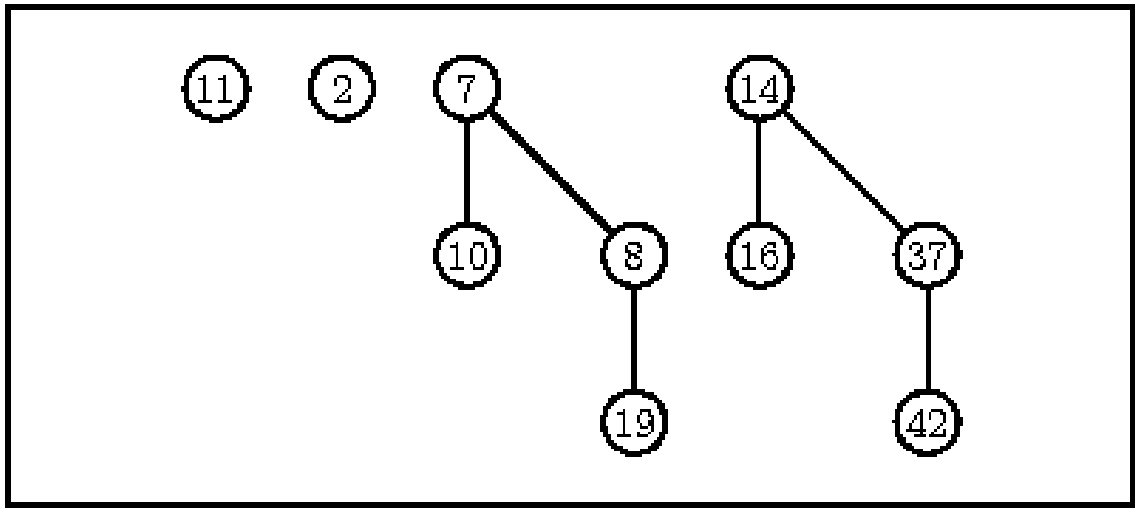
- $M = 111_{two}$ Digits = {0, 1}



- A binomial heap storing 7 integers which are kept in 3 binomial trees and stored in heap order within the trees
- The rank of a binomial tree is equal to the number of children of the root

A binomial heap using redundant binary representation

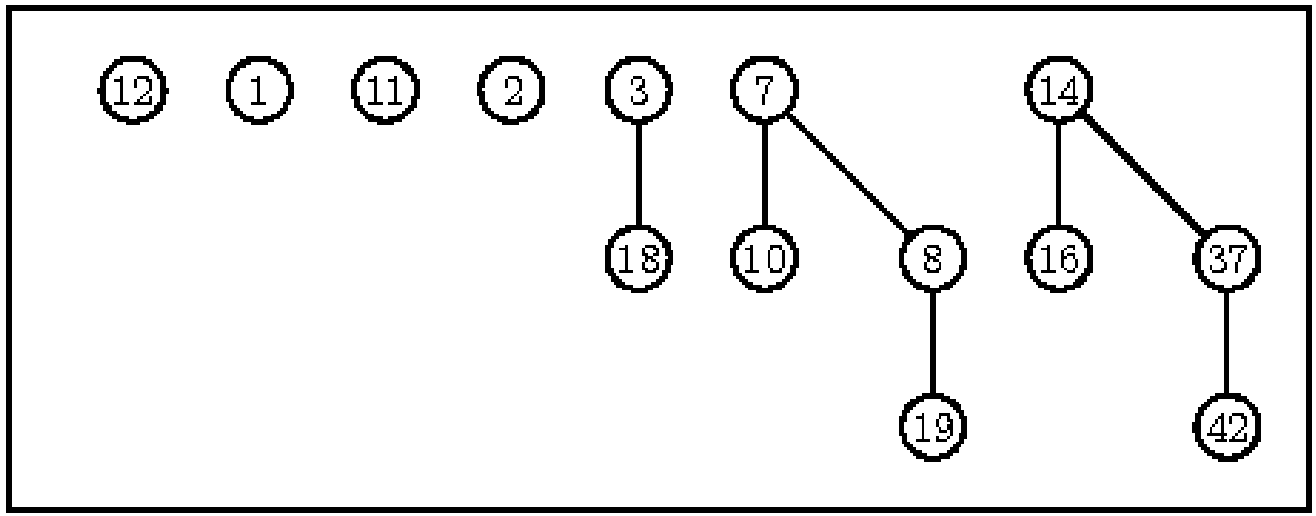
- $M = 202_{\text{redundant two}}$ Digits = {0, 1, 2}



- A binomial heap storing 10 integers

A binomial heap using redundant zeroless representation

- $M = 214_{\text{redundant four}}$ Digits = {1, 2, 3, 4}



- A binomial heap storing 14 integers

Zeroless number system

- Addition
  - M + 1
  - Corresponds to inserting a node into the binomial heap
- Subtraction
  - M – 1
  - Corresponds to extracting a node from the binomial heap

Insert and extract

- Using functionality which incrementally handles carries and borrows in the zeroless number system, both addition and subtraction can be accomplished at constant cost
- When using a carry/borrow stack to provide this functionality in the context of binomial heaps, both insert and extract can be performed at constant worst-case cost

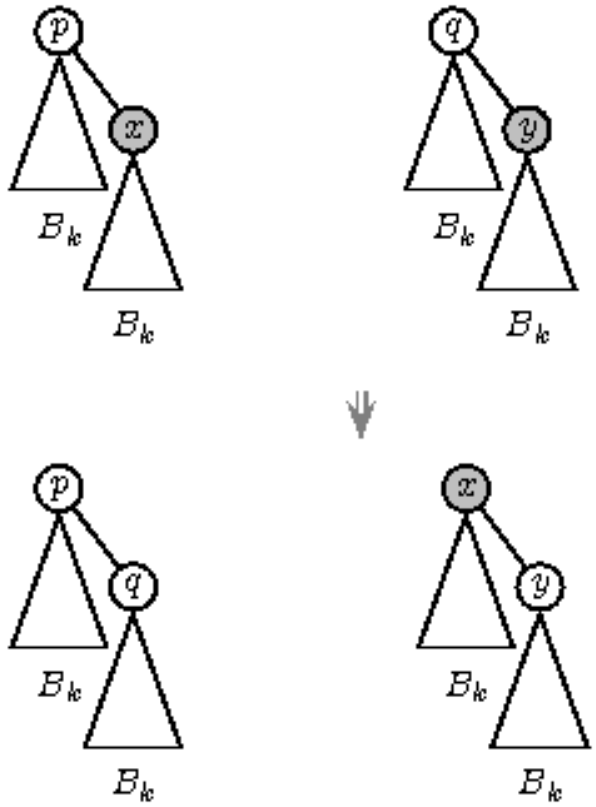
- Next: Run-relaxed heaps

Run-relaxed heaps

- A run-relaxed heap is composed of relaxed binomial trees which are almost heap-ordered binomial trees
- Some nodes can be marked active indicating that there may be a heap-order violation
- The maximum number of active nodes is $\lfloor \lg n \rfloor$ (Driscoll et al. 1988)
- A singleton is an active node which has no active siblings
- A run is a sequence of consecutive active siblings

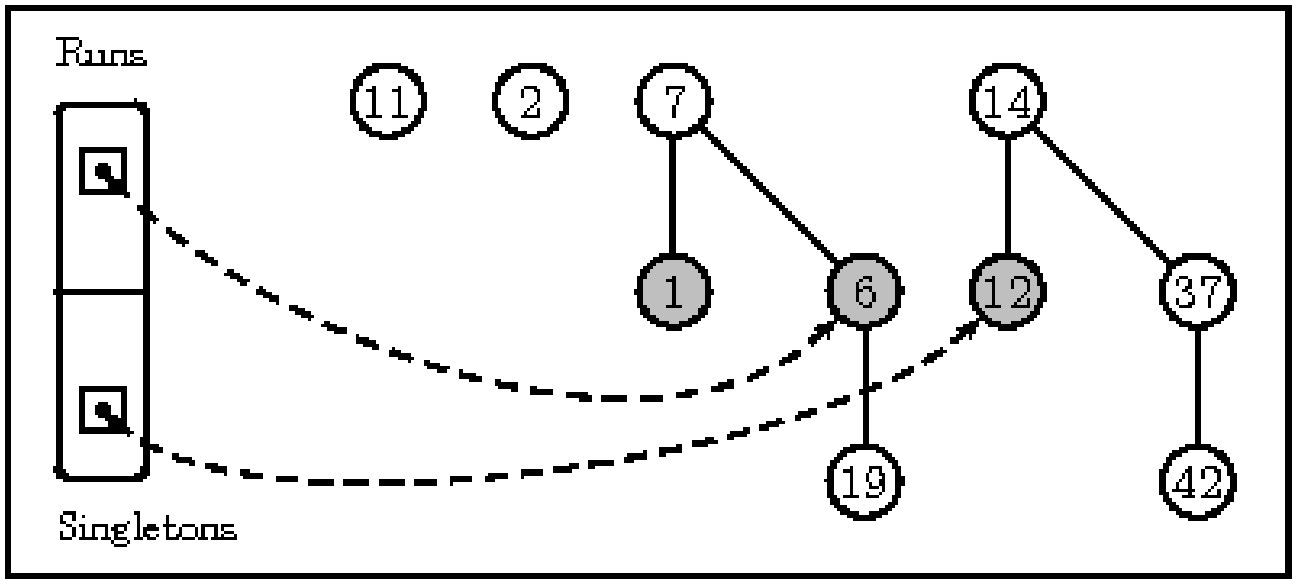
Run-relaxed heaps

- A series of run/singleton transformations is used to reduce the number of active nodes



- Illustrated with one of the singleton transformations

A run-relaxed heap

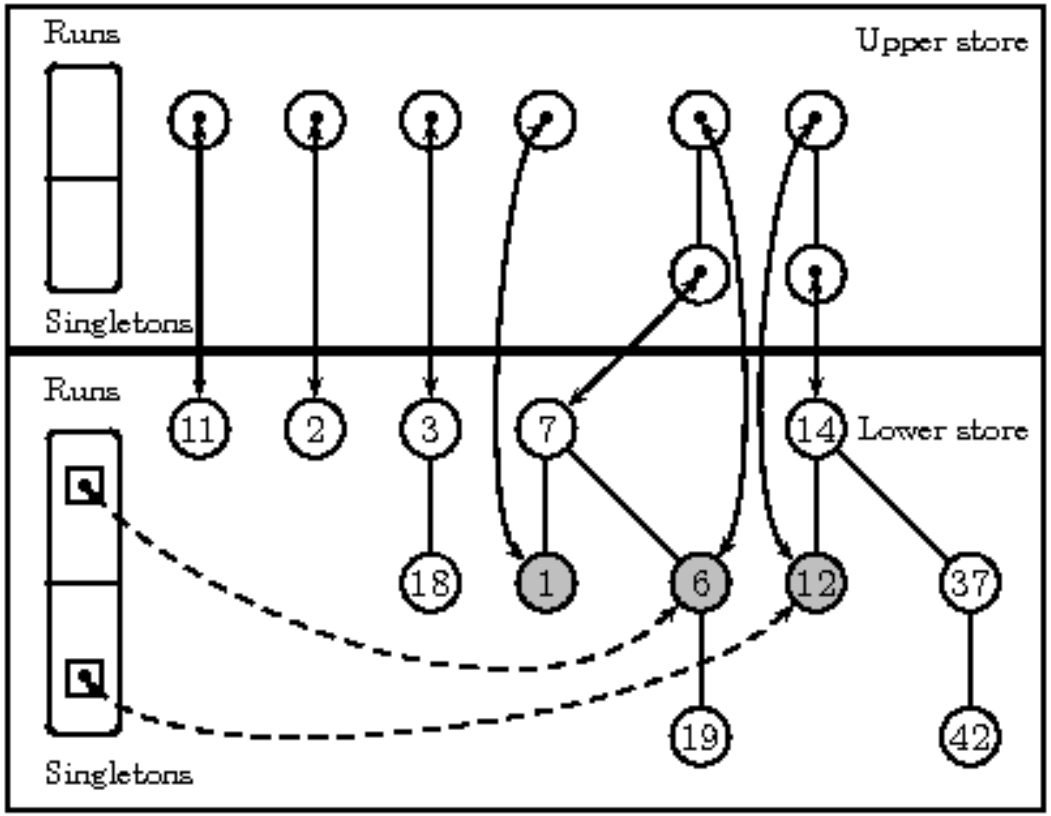


- A run-relaxed heap storing 10 integers. The relaxed binomial trees are drawn in schematic form and active nodes are drawn in grey

Two-tier relaxed heap components

- A two-tier relaxed heap has the following components:
  - An upper store which contains pointers to roots and active nodes held in the lower store
  - A lower store which contains the elements of the heap
- The two components are run-relaxed heaps using the redundant zeroless number system

A two-tier relaxed heap



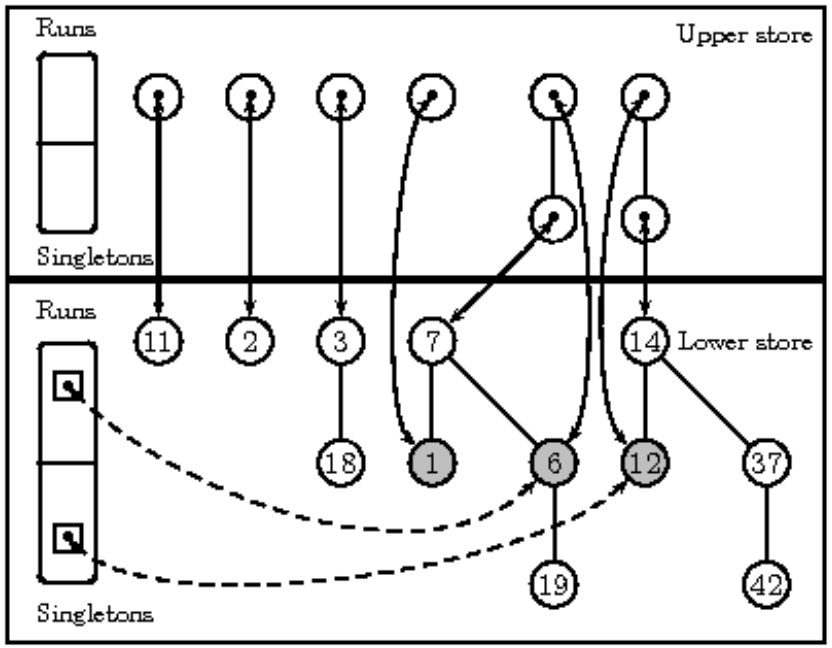
- A two-tier relaxed heap storing 12 integers

Extra functionality at the upper store

- Lazy deletions: Sometimes nodes in the upper store are marked instead of being deleted
  - When the number of marked nodes reaches half of the total number of nodes, the marked nodes are removed by an incremental global rebuilding

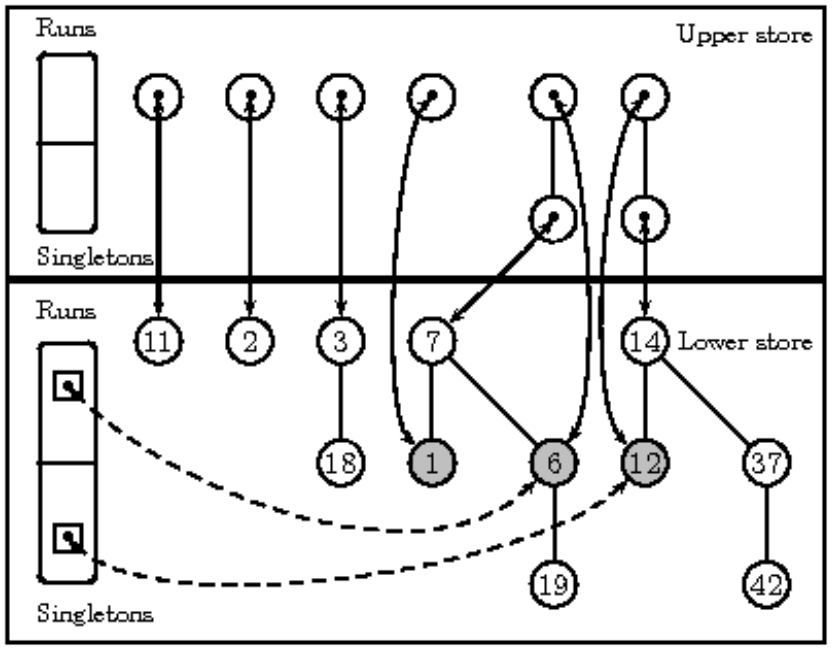
Heap operations

- Find-min:
  - The upper-store find-min operation is called, and this operation returns either one of the roots or one of the active nodes



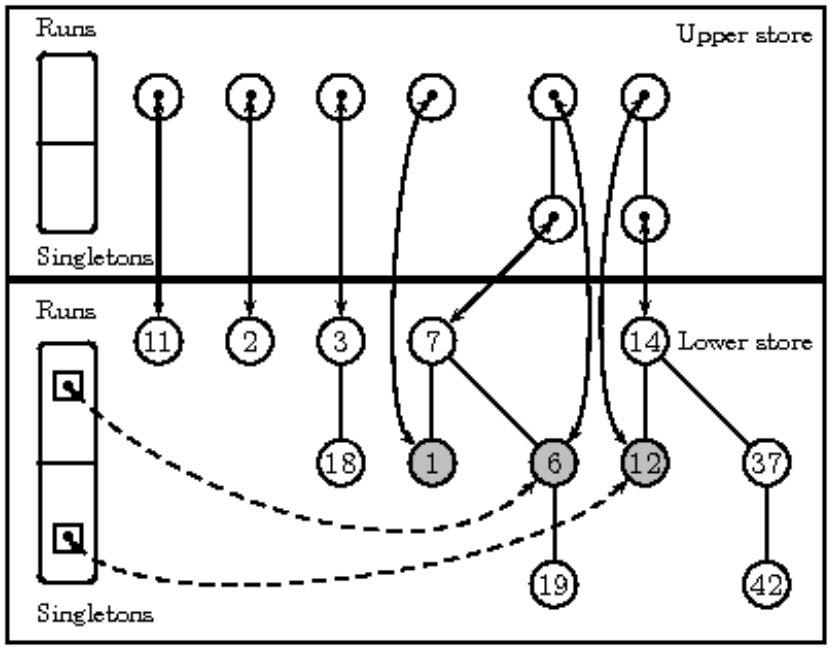
Heap operations

- Insert:
  - The lower-store insert operation is called which adds a new node to the lower store
  - A pointer to that node is inserted into the upper store
  - If due to a join of two trees a node is no longer a root, the pointer in the upper store associated with that node is marked



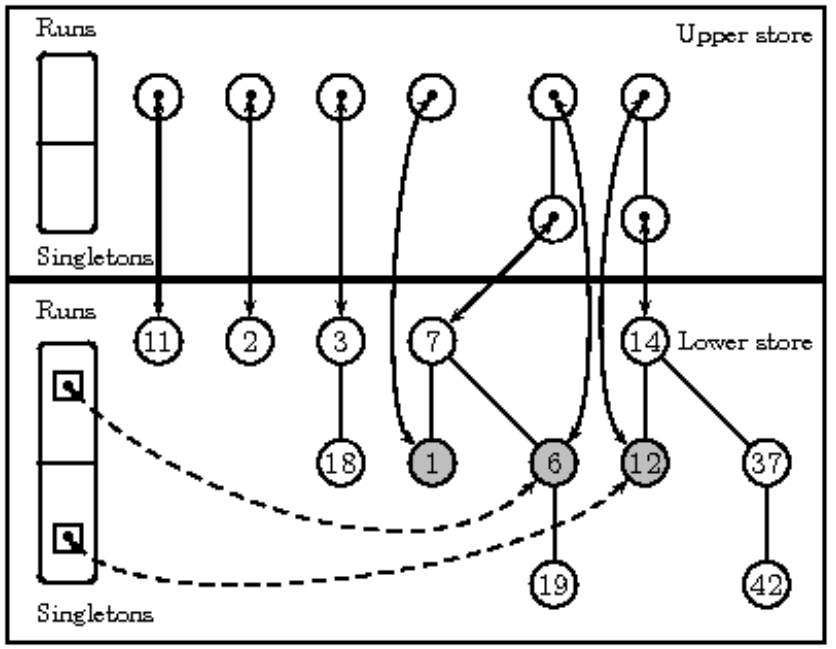
Heap operations

- Decrease:
  - After the element replacement the node is made active
  - If the number of active nodes has become too large, a constant number of transformations (singleton or run) is performed
  - A pointer to the node is inserted into the upper store



Heap operations

- Delete:
  - Lower store: The node is removed. A node is extracted, and this node and the roots of the former subtrees are joined together into a new tree
  - If deletion involves an internal node, the new root of the subtree is made active
  - Upper store: A pointer to the new root is inserted. If there exists a pointer to the removed node, it is deleted.



New contribution

- The introduction of extract
- Improved constant factors for delete

Open problems

- Constant factors for the find-min, insert, and decrease operations should be improved
- Can the bound for delete be improved to log n + O(1)? (this can be obtained when decrease is not supported)

Related work

- Amr Elmasry, Claus Jensen, and Jyrki Katajainen. *Relaxed weak queues: an alternative to run-relaxed heaps*. CPH STL Report 2005-2
- Amr Elmasry, Claus Jensen, and Jyrki Katajainen. *On the power of structural violations in priority queues*. CPH STL Report 2005-3
- Both available at www.cphstl.dk