Comparison complexity of priority-queue operations

Jyrki Katajainen (University of Copenhagen)

Joint work with Amr Elmasry (Max-Planck-Institut für Informatik) and Claus Jensen (University of Copenhagen)

16 January 2009
Updated 31 January 2009

These slides are available at
http://cphstl.dk

Priority-queue kernel

$insert(Q, p)$. Add an element with locator p to Q.

$extract(Q)$. Extract an **unspecified** element from Q and return a locator to that element. Precondition: $Q \neq \emptyset$.

$delete(Q, p)$. Remove the element with locator p from Q (without destroying it).

$meld(Q, R)$. Move all elements from Q and R to a new priority queue S, destroy Q and R, and return S.

$decrease(Q, p, x)$. . . . not relevant for this talk. . .

$create(Q)$. Create Q. Postcondition: $Q = \emptyset$.

$destroy(Q)$. Destroy Q. Precondition: $Q = \emptyset$.

$find\text{-}min(Q)$. Return a locator to an element that, of all elements in Q, has a minimum value. Precondition: $Q \neq \emptyset$.

$size(Q)$. Return the number of elements stored in Q.

$swap(Q, R)$. Make Q refer to the data structure referred to by R, and vice versa.

Focus

- Comparison complexity
- Worst-case efficiency
- Constant factors

God's rule 1: Do care about the size of $\mathcal{O}$!

Read [Schönhage et al. 1994]

I make no claims about the practical utility of the data structures discussed, even though the development of a library component that provides good practical performance is a challenging task.

Bug report

The binary-heap implementation in LEDA 6.1 is broken.

1. The time bounds are guaranteed in the amortized or average-case sense, not in the worst-case sense, contrary to what is claimed in the documentation.
2. $insert$ is extremely slow; it does not take $\mathcal{O}$ expected time as I would expect.

Research question

Assumptions: Trivial operations have $\mathcal{O}$ worst-case running time and perform no element comparisons. The worst-case running time of non-trivial operations is proportional to the number of element comparisons performed. (If not, specify explicitly.)

Question: What are the best worst-case bounds for the number of element comparisons performed by the non-trivial operations $insert$, $extract$, $delete$, and $meld$?

Answer: It is still open whether optimal bounds $\mathcal{O}$ for $insert$, $extract$, and $meld$; and $\lg n + \mathcal{O}$ for $delete$ can be achieved or not.

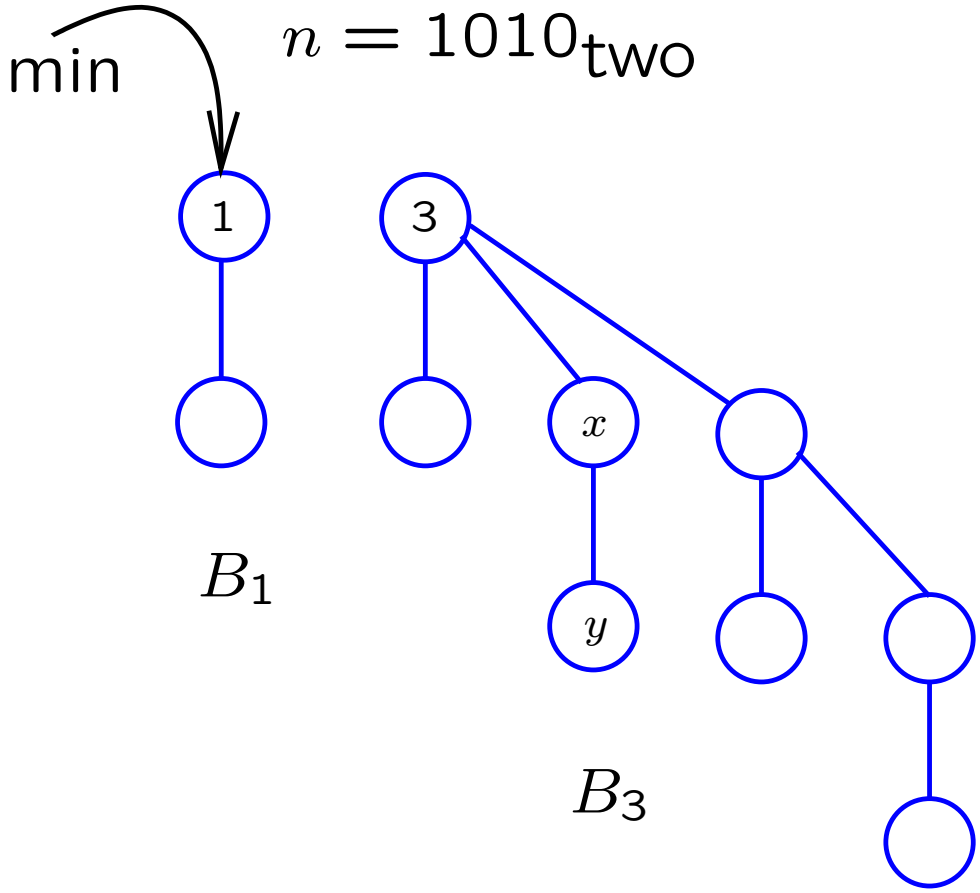
n, m: # of elements stored just prior to an operation

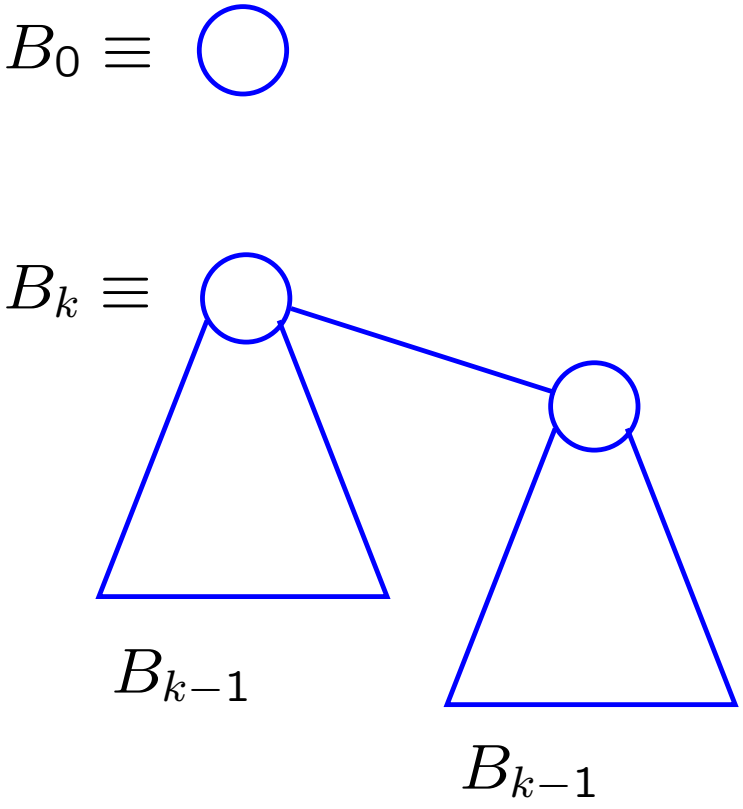
Biased history

Year	Inventor	*insert*	*extract*	*delete*	*meld*
1964	Williams	$\lg n + \mathcal{O}$	$\mathcal{O}$	$2\lg n + \mathcal{O}$	$\mathcal{O}\lg n \lg m$
1978	Vuillemin	$\lg n + \mathcal{O}$	$\mathcal{O}\lg n$ time	$2\lg n + \mathcal{O}$	$\lg n + \mathcal{O}$
1988	Driscoll & al.	$\mathcal{O}$	$\lg n + \mathcal{O}$	$3\lg n + \mathcal{O}$	$\lg m + \mathcal{O}$
1995	Brodal	$\mathcal{O}$	$\mathcal{O}\lg n$ time	$6\lg n + \mathcal{O}$	$\mathcal{O}$
1996	Brodal & Okasaki	$\mathcal{O}$	$\mathcal{O}\lg n$ time	$4\lg n + \mathcal{O}$	$\mathcal{O}$
2004	Elmasry	$\mathcal{O}$	–	$1.44\lg n + \mathcal{O}\lg\lg n$	–
	Jensen & me	$\mathcal{O}$	$\mathcal{O}$	$\lg n + \mathcal{O}\lg\lg n$	–
	Elmasry	$\mathcal{O}$	$\mathcal{O}$	$\lg n + \mathcal{O}$	–
Now	New results	$\mathcal{O}$	$\mathcal{O}$	$\lg n + \mathcal{O}$	$2\lg n + 2\lg m + \mathcal{O}$
		$\mathcal{O}$	$\lg n + \mathcal{O}$	$3\lg n + \mathcal{O}$	$\mathcal{O}$

- $\mathcal{O} \leq 10$; $n \geq m$.
- Bounds displayed in red are not proved in the original papers.
- See our joint paper *Multipartite priority queues* in *ACM Transactions on Algorithms* **5**,1 (2008), Article 14.

Binomial queues





- heap-ordered $x \leq y$
- at most $\lfloor \lg n \rfloor + 1$ binomial trees

Read [Cormen et al. 2001]

Redundant binary numbers

Binary representation:

$$n = \sum_{i=0}^{\lfloor \lg n \rfloor} d_i 2^i,$$

where $d_i \in \{0, 1\}$ for all $i \in \{0, 1, \ldots, \lfloor \lg n \rfloor\}$.

Redundant representation:

$$n = \sum_{i=0}^{\lfloor \lg n \rfloor} d_i 2^i,$$

where $d_i \in \{0, 1, 2\}$ for all $i \in \{0, 1, \ldots, \lfloor \lg n \rfloor\}$.

Read [Okasaki 1998]

$$\begin{array}{rcccc} & & 1 & 1 & 1 \\ + & & & & 1 \\ \hline & 1 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{rcccc} & & 1 & 1 & 1 \\ + & & & & 1 \\ \hline & & 1 & 1 & 2 \end{array}$$

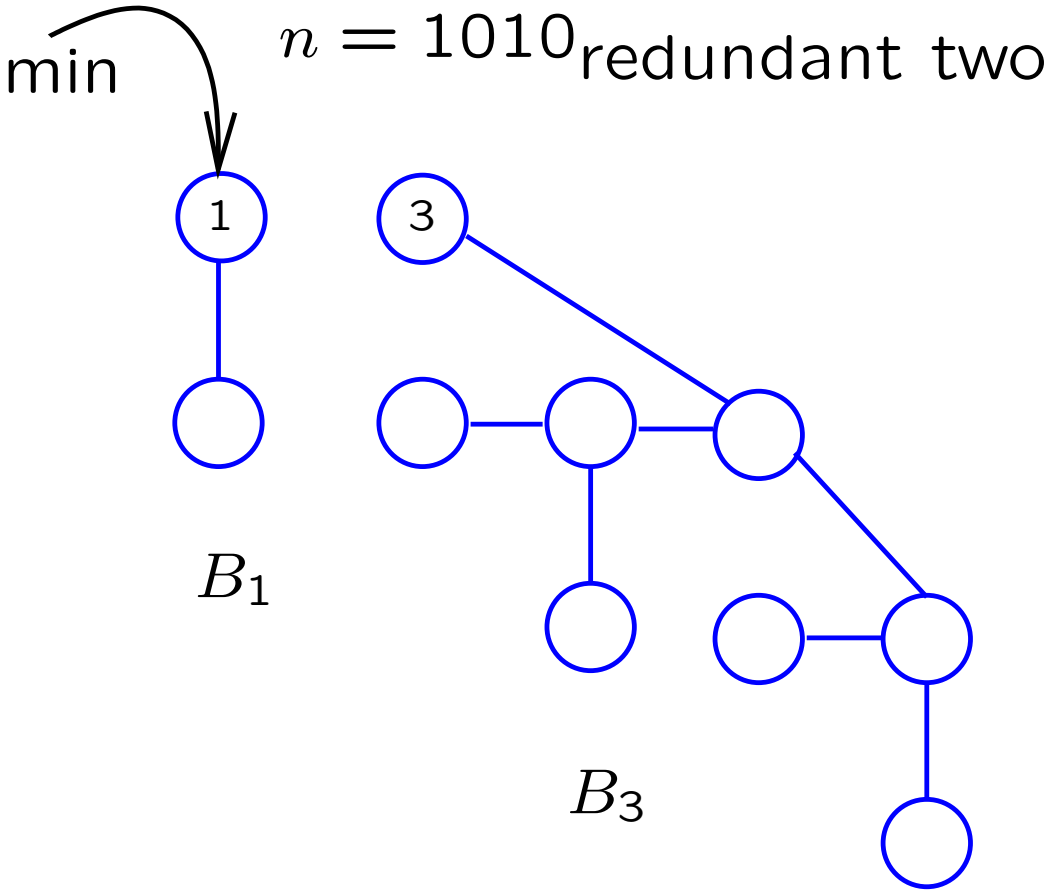
Assuming a **carry stack** is available, ++ is performed as follows:

1. Fix the topmost carry if the stack is not empty.
2. Add one as desired.
3. If the least significant digit becomes 2, push this carry onto the stack.

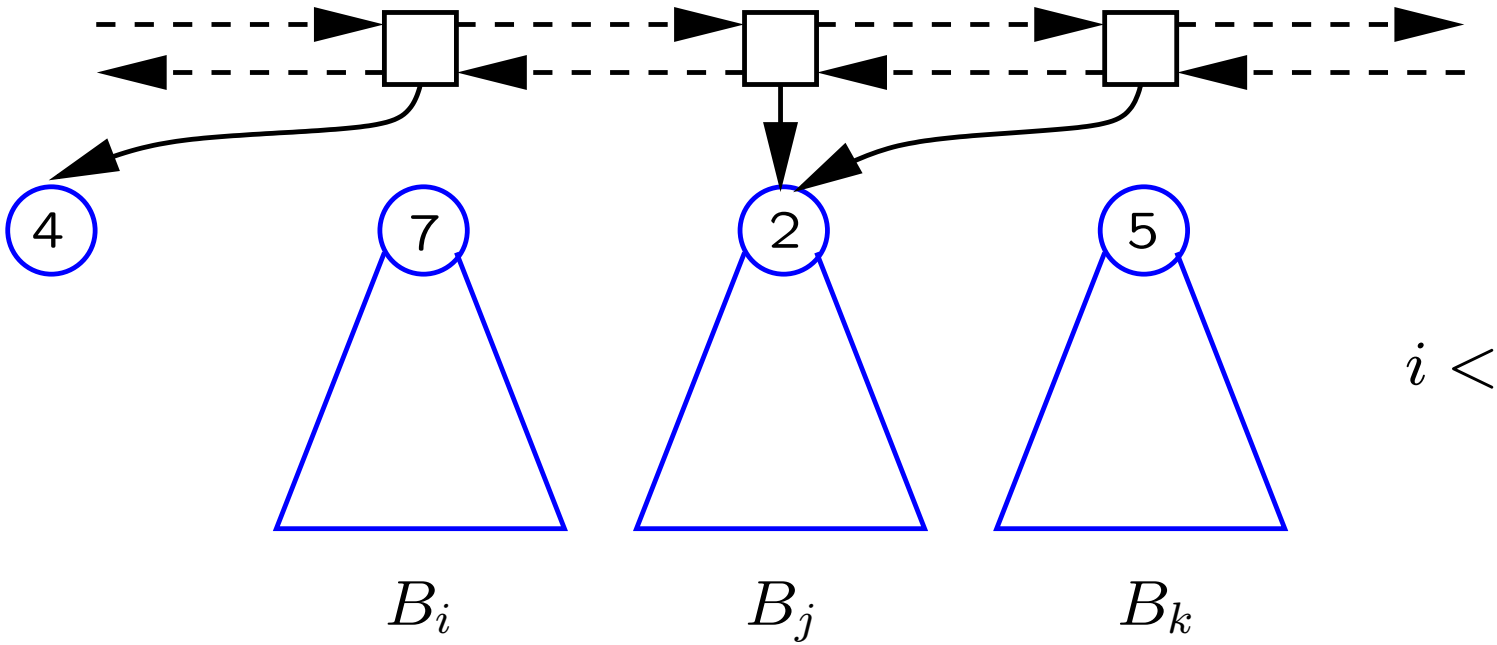
Bug report

The binomial-queue implementation in [Cormen et al. 2001] can be improved:

1. By maintaining a pointer to the minimum, the running time of *find-min* can be improved from $\Theta \lg n$ to Θ.
2. With the redundant binary representation, the running time of *insert* can be improved from $\Theta \lg n$ to Θ.
3. It would be easier to maintain referential integrity if parent pointers were only maintained for the largest children.



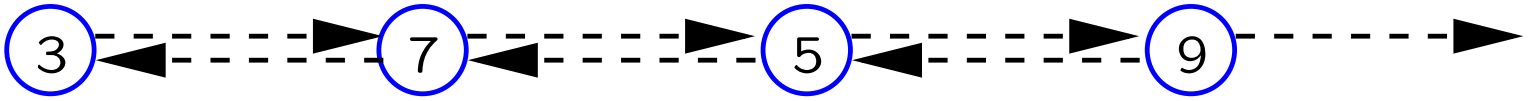
Prefix-minimum pointers



The key observation, what I call the $\uparrow^{\rightarrow}$ **property**, is that when a value in tree B_j is updated, the number of element comparisons made to fix B_j is $j + \mathcal{O}$ and the number of element comparisons made to fix the prefix-minimum pointers at higher trees is at most $\lg n - j + \mathcal{O}$.

Read [Elmasry et al. 2008]

Delivery buffer



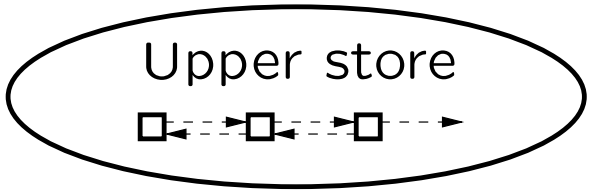
Maintain a buffer whose size is between 1 and $\lg n + \mathcal{O}$. The first node stores the minimum element held in the buffer. The buffer can be used for **borrowing** and **inserting** elements. Special handling is necessary when the buffer becomes too small or too large.

Incremental processing

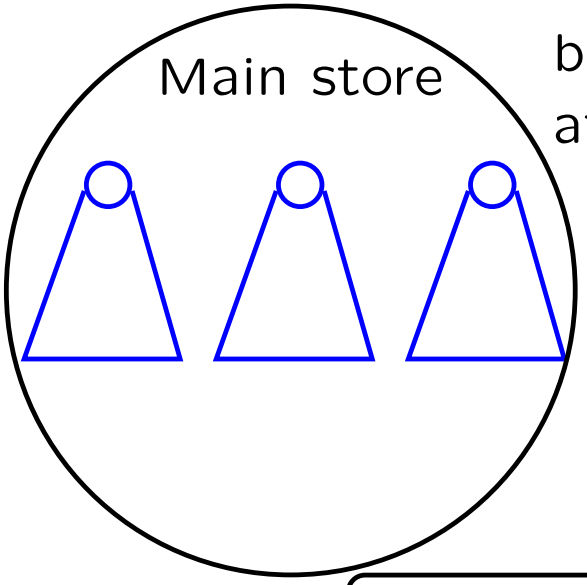
Perform a task taking $\mathcal{O}\,t$ time incrementally during the forthcoming t modifying operations ($insert$, $extract$, $delete$, and $meld$) so that each subtask is about equal in size. It is important that there is at most one incremental process running at a time.

The simplest way of implementing this idea is to divide the task into primitive steps and execute a fixed number of primitive steps in connection with each subtask. An alternative way is to use **coroutines** in languages that support them.

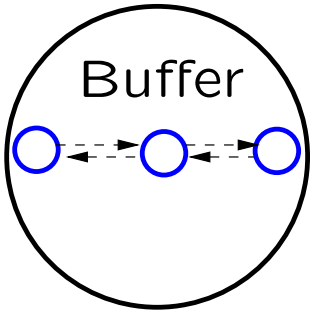
Multipartite priority queues



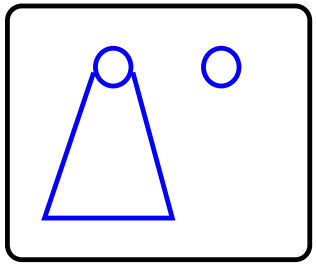
doubly-linked list
at most $\lg n + \mathcal{O}$ prefix-minimum pointers



binomial queue (binary)
at most n elements

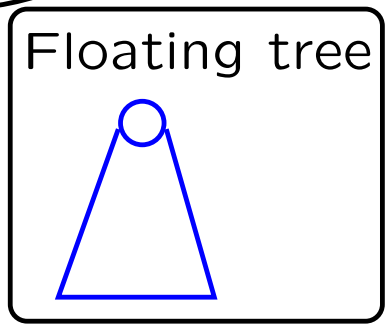


doubly-linked list
$1..\lg n + \mathcal{O}$ elements



binomial queue (redundant)
$0..\frac{1}{2}\lg n + \mathcal{O}$ elements

Recall where the overall minimum is!



binomial tree
at most n elements

Priority-queue operations (Part I)

insert:

– Insert the given node into the buffer.

– If the buffer is larger than $\frac{7}{8}\lg n + \mathcal{O}$, construct B_k, where $\frac{1}{4}\lg n + \mathcal{O} \leq 2^k < \frac{1}{2}\lg n + \mathcal{O}$, and unite it with the main store.

[Done incrementally during the next $\frac{1}{8}\lg n + \mathcal{O}$ operations!]

extract:

– Remove a node from the buffer.

– If the buffer is smaller than $\frac{1}{8}\lg n + \mathcal{O}$, take the smallest tree from the main store and split it (if possible). If the smallest tree is of size 1, move it to the buffer. After such a move, update the upper store accordingly. Do this splitting repeatedly until the buffer contains more than $\frac{1}{4}\lg n + \mathcal{O}$ elements.

[Done incrementally during the next $\frac{1}{8}\lg n + \mathcal{O}$ operations!]

Priority-queue operations (Part II)

delete:
– If the removed node is in the buffer, remove it and update the buffer minimum. Stop.
– Swap the removed node with its parent until it reaches the root of its tree B_k.
– Borrow a node from the buffer.
– Remove the root of B_k.
– Rebuild B_k using the root's subtrees and the borrowed node.
– If the removed node was in the main store, update the prefix-minimum pointers for trees larger than B_k, if necessary.

meld(Q, R): (if $size(Q) \leq size(R)$)
– Throw away both upper stores.
– Concatenate the two buffers.
– If Q had a floating tree under construction, complete this.
– If R had a floating tree under construction, continue this.
– If the buffer is still too large, start the creation of a new floating tree.
– Unite the two main stores, the two new trees (if any), and the two floating trees (if any).
– Create an upper store for the new main store.

Bug report

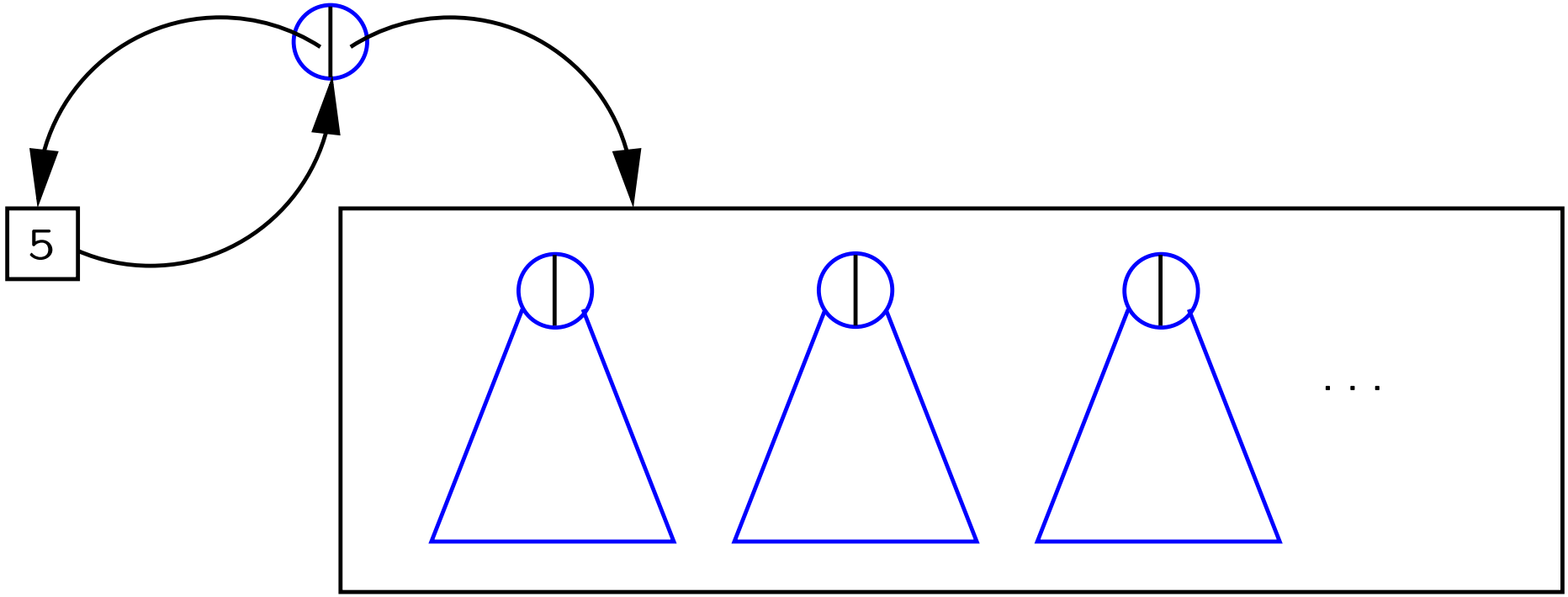
I claimed that $\mathcal{O} \leq 10$!

1. Construction of a binomial queue of size $\frac{1}{2}\lg n + \mathcal{O}$ requires $\frac{1}{2}\lg n + \mathcal{O}$ element comparisons.
2. Union of the floating tree and a binomial queue of size n can require up to $\lg n + \mathcal{O}$ element comparisons.
3. Finding the minimum of 4 elements requires 3 element comparisons.
4. This work is shared by $\frac{1}{8}\lg n + \mathcal{O}$ modifying operations. Because of the incremental process itself, $\mathcal{O} > 15$.

How to recover from this error?

Bootstrapping

The nodes contain (element, priority queue) pairs.



The element at a node is a representative for the whole group.

Bootstrapped priority queues

The whole priority queue is represented as a pair locator. From the outside, this priority queue is accessed with element locators. The priority queues inside are accessed with pair locators.

Implement the priority queues inside using binomial queues (redundant).

# of element comparisons without a minimum pointer:

find-min	*insert*	*extract*	*delete*	*meld*
$\lg n + \mathcal{O}$	$\mathcal{O}$	$\mathcal{O}$	$\lg n + \mathcal{O}$	$\lg n + \mathcal{O}$

Priority-queue operations (Part I)

$insert(Q, x)$:

$(y, R) \leftarrow pair[Q]$
if $element[x] < element[y]$
  $p \leftarrow (y, -)$
  $insert(R, p)$
  $pair[Q] \leftarrow (x, R)$
else
  $p \leftarrow (x, -)$
  $insert(R, p)$
  $pair[Q] \leftarrow (y, R)$

$extract(Q)$:

$(x, R) \leftarrow pair[Q]$
$(y, S) \leftarrow extract(R)$
$T \leftarrow meld(R, S)$
$pair[Q] \leftarrow (x, T)$
return y

Priority-queue operations (Part II)

$delete(Q, x)$:

$(y, R) \leftarrow pair[Q]$
if $x = y$ **and** $size(R) = 0$
  $pair[Q] \leftarrow (-, -)$
else if $x = y$ **and** $size(R) > 0$
  $(z, S) \leftarrow p \leftarrow \mathit{find\text{-}min}(R)$
  $delete(R, p)$
  $T \leftarrow meld(R, S)$
  $pair[Q] \leftarrow (z, T)$
else
  $(x, S) \leftarrow p \leftarrow pair[x]$
  $delete(R, p)$
  $T \leftarrow meld(R, S)$
  $pair[Q] \leftarrow (y, T)$

$meld(Q,\ R)$:

$(x, S) \leftarrow pair[Q]$
$(y, T) \leftarrow pair[R]$
if $element[x] < element[y]$
  $p \leftarrow (y, T)$
  $insert(S, p)$
  $pair[Q] \leftarrow (x, S)$
else
  $p \leftarrow (x, S)$
  $insert(T, p)$
  $pair[Q] \leftarrow (y, T)$
return Q

Open problems

- What is the best bound for $delete$ if $insert$, $extract$, and $meld$ run in $\mathcal{O}$ worst-case time?
- Another story if $decrease$ is to be supported in $\mathcal{O}$ worst-case time. Read [Brodal 1996].
- How to implement a worst-case efficient priority queue in an industry-strength program library?