Heaps

(Version of 21 November 2005)

- A *min-heap* (resp. *max-heap*) is a data structure with fast extraction of the smallest (resp. largest) item (in $O(\lg n)$ time), as well as fast insertion (also in $O(\lg n)$ time), at the expense of slow search (in only $O(n)$ time).
- To make things easier, we talk about heaps of *integers*, with no satellite data. Abstraction and higher-order functions allow us to implement heaps of items of any *ordered* data structure.
- Heaps are frequently used in software. A particular structure is the *priority queue*, where items are added to a pool and assigned a priority. The item with the lowest/highest priority gets extracted first. In a real-time system, this extraction operation must be implemented efficiently.

Heaps and Binomial Trees

Def. A *binary heap* is a completely filled binary tree, except possibly at the lowest level, which is filled from the left, such that the key of each non-root node is at least the key of its parent (*heap property*).

Def. A *binomial tree* is recursively defined as follows:

- A binomial tree of rank 0 (denoted by B_0) has a single node.
- A binomial tree of rank k (denoted by B_k) is formed by linking together two binomial trees of rank $k-1$, making one of them the leftmost child of the other one.

Note that binomial trees are *not* binary trees.

Prop. A binomial tree of rank k has height k (in number of edges), has 2^k nodes in total, and has $\binom{k}{i}$ nodes at depth i (hence its name!). Its root has degree k and its children have degrees $k-1, k-2, \ldots, 0$.

Representation of Binomial Trees and Heaps

We represent binomial trees by *labelled* trees, such that:

```
datatype binoTree = Node of int * int * binoTree list
REP. CONV. & INV.: the first integer, k, is the rank
of the tree, the second integer is the key at its root,
and the list has k child trees, ordered by decreasing
ranks k-1, k-2, ..., 1, 0.
```

Def. A *binomial heap* is a list of binomial trees, such that:

```
type binoHeap = binoTree list
REPRESENTATION INVARIANT: in each binomial tree, the key
of each non-root node is at least the key of its parent
(heap property) (hence the root of each tree contains
its minimum key); the trees have increasing ranks.
```

Consequences of the Properties

Reminder of some properties:

- A binomial tree of rank/degree k contains 2^k nodes.
- In a heap, no two binomial trees have the same rank/degree.

Consider binary arithmetic:

$$22_{10} = 10110_2$$

A binomial heap of 22 items is built from one binomial tree of rank 1, one binomial tree of rank 2, and one binomial tree of rank 4.

A binomial heap of n items has at most $\lfloor \lg n \rfloor + 1$ binomial trees, hence its minimum item can be found in $O(\lg n)$ time.

Linking Two Binomial Trees

When constructing binomial trees or heaps, we often have to link two binomial trees of the *same* rank r (this is a pre-condition!) in order to form a new binomial tree of rank $r + 1$; this takes $\Theta(1)$ time, no matter what the sizes of the trees are:

```
fun link(t1 as Node(r,x1,c1) , t2 as Node(r,x2,c2)) =
    if x1 < x2 then
        Node(r+1,x1,t2::c1)
    else
        Node(r+1,x2,t1::c2)
```

Note that the resulting binomial tree can become part of a heap, as it respects the heap property (in the representation invariant).

Inserting a Tree or Item into a Binomial Heap

Inserting a *binomial tree* of rank r into a binomial heap of n items, whose binomial trees have ranks $r' \geq r$ (pre!), takes $O(\lg\, n)$ time, maintaining the list of binomial trees ordered by increasing ranks:

```
fun rank (Node(r,x,c)) = r


fun insTree(t,[]) = [t]
  | insTree(t, ts as t'::ts') =
    if rank t < rank t' then t::ts
    else
         insTree(link(t,t'),ts')
```

Inserting an *item* into a binomial heap of n items takes $O(\lg\, n)$ time:

```
fun insert(x,ts) = insTree(Node(0,x,[]),ts)
```

Merging Two Binomial Heaps

Merging two binomial heaps with a total of n items takes $O(\lg n)$ time:

```
fun merge(ts1,[]) = ts1
  | merge([],ts2) = ts2
  | merge(ts1 as t1::ts1' , ts2 as t2::ts2') =
    if rank t1 < rank t2 then
        t1::merge(ts1',ts2)
    else if rank t2 < rank t1 then
             t2::merge(ts1,ts2')
         else
             insTree(link(t1,t2) , merge(ts1',ts2'))
```

If this operation is not needed, then binary heaps perform better.

Finding/Deleting the Minimum of a Bino. Heap

Finding or deleting the minimum item of a binomial heap with $n > 0$ items takes $O(\lg n)$ time:

```
fun root (Node(r,x,c)) = x
fun removeMinTree [t] = (t,[])
  | removeMinTree (t::ts) =
    let val (t',ts') = removeMinTree ts
    in if root t < root t' then (t,ts) else (t',t::ts') end
fun findMin ts =
    let val (t,_) = removeMinTree ts
    in root t end
fun deleteMin ts =
    let val (Node(_,_,ts1),ts2) = removeMinTree ts
    in merge(rev ts1, ts2) end
```