CSCE 750
10/26/2023

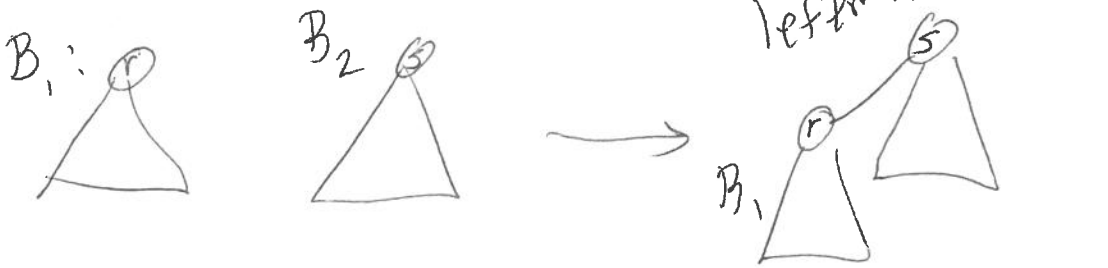
Binomial Heaps — implements mergeable heap

Structure — A binomial heap is a sequence of binomial trees of strictly increasing degree.

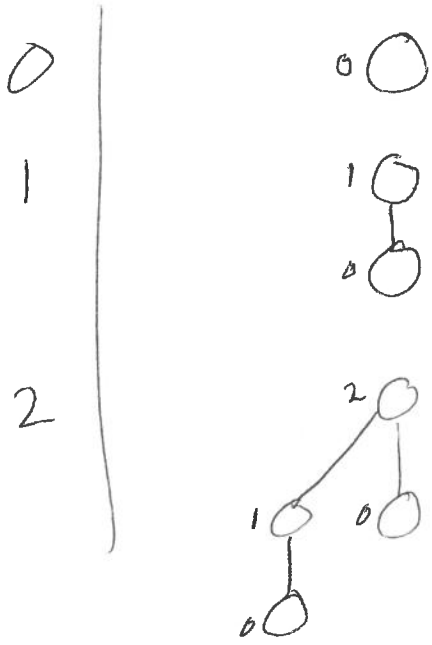
Binomial tree (inductive def):

— A single node (root). Degree 0

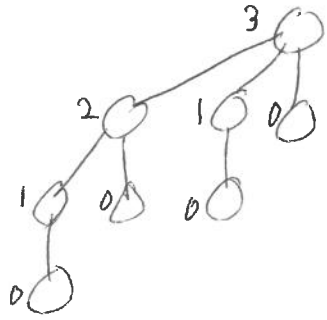
— Given binomial trees of the same degree d, can combine into a single binomial tree of degree $d+1$ as follows



degree:



3



Size of a tree of degree d is exactly 2^d (Proof by induction on d)

Also degree = # children of the root.

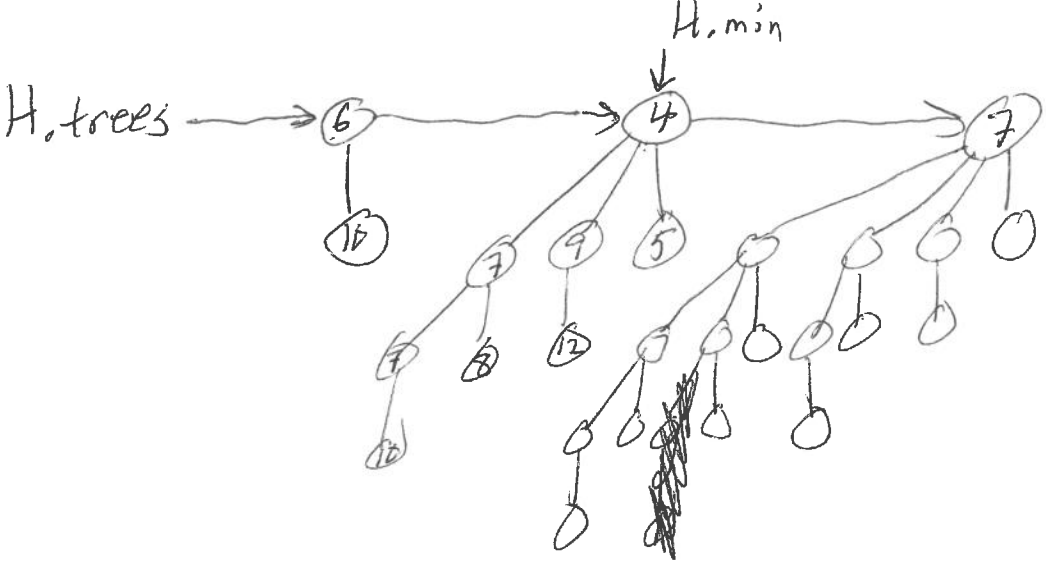
Note: every node in a binomial tree is the root of a binomial tree. The degree of a node is the number of its children.

Children of a degree d node have degrees $d-1, d-2, \ldots, 0$ from left to right.

[Induction on d]

(Min) binomial heap: sequence of binomial trees of strictly increasing degree, where keys (and other data) are stored in the nodes of the trees, and each tree is in min-heap order.

Ex: H min-binomial-heap



H.size = 26

= 11010

4 3 2 1 0

Binomial heaps of the same size have the same shape exactly: degrees of trees correspond to positions of 1's in the binary rep of the size.

Implementing a binomial tree

A node ~~with~~ will have the following fields:

key (or data) — key and including any other satellite data

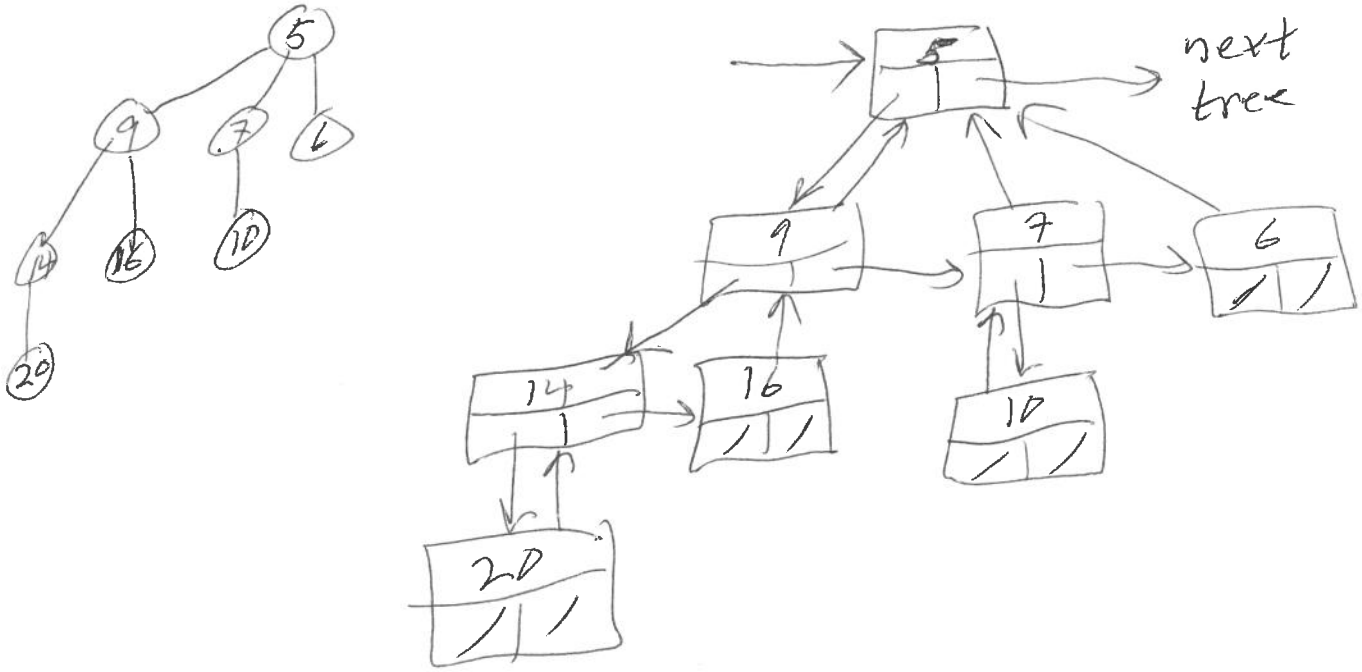
degree — the degree of the node

parent — points to the parent (Null if root)

~~leftmost~~

leftmost_child — points to leftmost child (Null if leaf)

right_sibling — points to the sibling to the right (if root, points to next tree in the heap, if there is one) Null if rightmost child or root of the rightmost tree.



Let n be size of H.

Then H has $\leq \lg n + 1$ trees

A tree of degree d has depth d.

~~$d = 0$~~ $d \leq \lg n$

$2^d \approx$ tree size $\leq n$

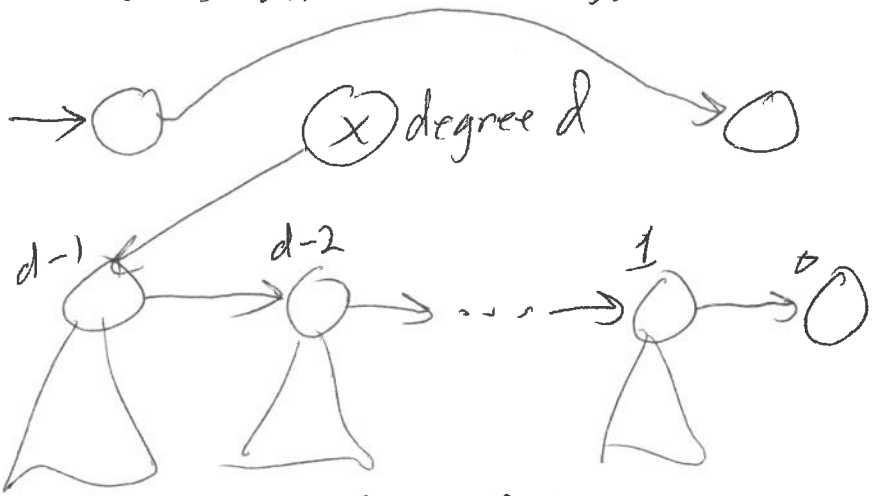
Delete(x, H)

- DecreaseKey$(H, x, -\infty)$
- DeleteMin(H)

DeleteMin(H)

- let $x :=$ FindMin(H)

x is the root of some tree



- Remove x from list of H.trees
- Reverse the list of its children, call that H'.trees

H.trees points to leftmost root in H

optional [H.min " " min key node (must be a root)]

Basic Operations & their runtimes

FindMin(H) — Return H.min if this field is there, otherwise, traverse the roots looking for the minimum.

O(lg n) = O(#trees)

Insert(H, x) — Insert key x into H:

— Create a size-1 binomial Heap

H': → (x)

— merge H' with H, returning the result

Merge(H, H')

DecreaseKey(H, x, k) — decrease the key of node x from x down to k.

{assume k < x}

O(lg n) = O(depth of biggest tree in the heap)

— change key x to k

— "bubble up":

while k < parent key

swap k with parent

keep going

- Merge(H, H')

Merge(H_1, H_2)

1) Merge tree lists H_1.trees & H_2.trees into a single list L of tree with ascending degree (not strictly).

Initially For any d, there are ≤ 2 trees in L of degree d (adjacent)

Ex: