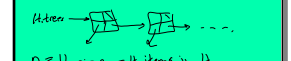
Binomial Heaps — basic ops

H — binomial heap

H.trees — list of trees



n = H.size = # items in H

tree degrees uniquely determined by n

Ex: n = 13: $13 = 2^0 + 2^2 + 2^3$

3 trees: deg 0, deg 2, deg 3.

FindMin(H) — return the min root.

Time = $\Theta(\log n) = \Theta(\#$ of trees$)$

[H.min is an optional attribute that points to the min root.]

DecreaseKey(H, x, k)

change x.key to k

(assuming $k < x.key$)

// cascade x up in its tree:

Time $\Theta(\log n)$ worst case

x.key := k

while k < x.parent.key

swap x with x.parent

Insert(H, x) — insert x into H.

Create a new heap H' whose only element is x.

H := Merge(H, H')

Time $= \Theta($time to merge$) \leq \Theta(\log n)$

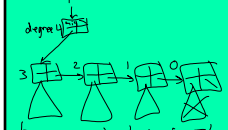
DeleteMin(H)

$\Theta(\log n)$ T := FindMin(H)

// T is the root of some tree

$O(1)$ Remove T from H.trees

[need prev pointer for this]



$\Theta(\log n)$ Reverse the list of T's children:

create new H' with H'.trees pointing to the reverse list

H'.size $= 2^d - 1$ $(d = deg(T))$

$\Theta(\log n)$ H := Merge(H, H')

return T.data

H_1, H_2 binomial heaps

Merge(H_1, H_2):

$n_1 := H_1$.size

$n_2 := H_2$.size

carry := Nil

outlist := Nil

while $n_1 > 0$ and $n_2 > 0$:

if carry ≠ Nil and

n_1 is odd and n_2 is odd:

remove [1st tree in H_2.trees] & add it to outlist

carry := MergeTree(carry, H_2.trees)

// unlink 1st tree in H_2

H_2.trees := H_2.trees.right

else if carry ≠ Nil and

n_1 is odd and n_2 is even:

carry := MergeTree(carry, H_1.trees)

remove 1st tree from H_1

else // similar if have 2 trees of the same degree

:

else if carry = Nil and

n_1, n_2 both even

put carry onto outlist

carry := Nil

else // similar if only one tree of degree d

end-if

$n_1 := \lfloor n_1/2 \rfloor$

$n_2 := \lfloor n_2/2 \rfloor$

end while

if $n_1 > 0$:

insert remaining H_1 trees onto outlist

if $n_2 > 0$: // similar for H_2

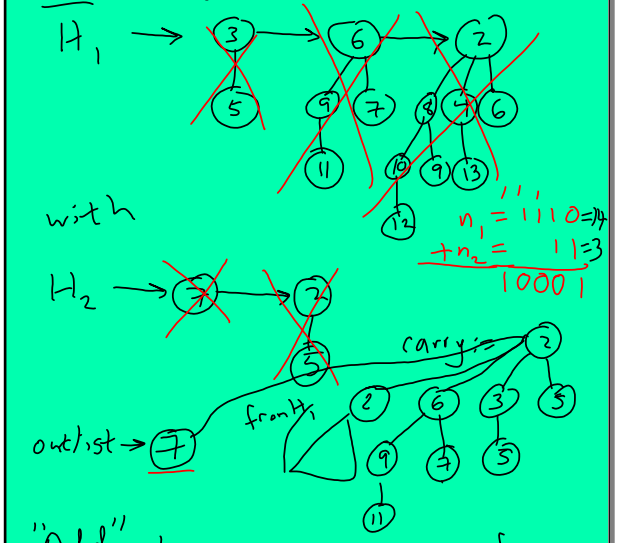
New heap H

H.size := H_1.size + H_2.size

H.trees := reverse(outlist)

return H

Ex: Merge



"Add" trees, merging trees of the same degree, making that the carry.

Time to Merge is $\Theta(\lg n)$ worst-case,

MergeTree(T_1, T_2) // both nonempty

O(1) { if T_1.key $< T_2$.key
swap T_1 with T_2
insert T_1 at the front of T_2.left

Delete(H, x) – delete x from H

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

End of Binomial Heaps

Fibonacci Heaps

Amortized performance is similar to Binomial Heaps with one useful exception:
DecreaseKey take O(1) amortized time