* priority queue w/ $O(1)$ insert and $O(1)$ getMin ?

hard for each individual operation. (and $O(\log n)$ extractMin)

but possible "amortized"

cheap insert usually

expensive insert some time

* observation :

insert to (merge w/) small tree : cheap

large : expensive

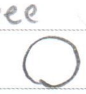
idea: instead of using one tree (heap or leftist)

use many trees, from small to large

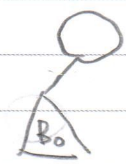
try merge w/ small trees first

* binomial tree

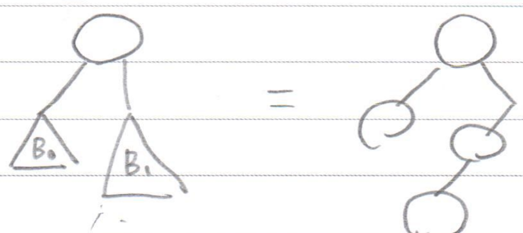
B_0 =



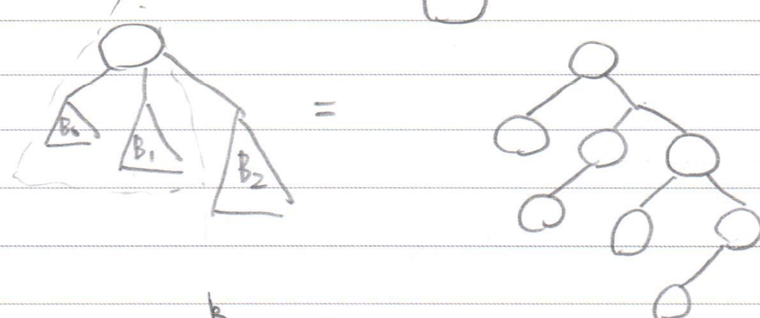
B_1 =



B_2 =



B_3 =



B_k contains 2^k nodes

Subject : 2011 No. : 9-2
Date : ..../......../........

* binomial forest
  { at most one B_k per each k }
  ⇒ can represent any n elements in node

$n = 10$: $\{B_3, B_1\}$ binary number representation
$n = 11$: $\{B_3, B_1, B_0\}$
⋮

* (min-) binomial heap
  binomial forest w/ min-trees

insert ○ to $\{B_0\}$ ⇒ $\boxed{\text{merge}}$ B_0 B_0 ⇒ B_1
insert ○ to $\{B_1\}$ ⇒ $\{B_0, B_1\}$
insert ○ to $\{B_0, B_1\}$ ⇒ $\boxed{\text{merge}}$ B_0, B_0, ⇒ $\boxed{\text{merge}}$ B_1, B_1 ⇒ B_2
insert ○ to $\{B_2\}$ ⇒ $\{B_0, B_2\}$
⋮

every one insertions : add B_0
two : merge B_0, B_0
four : merge B_1, B_1
eight : merge B_2, B_2

after n insertions
$\frac{n}{2^k} + \frac{n}{2^{k-1}} + \cdots + n = O(n)$ operations → amortized $O(1)$
(like incrementing binary numbers)

* get Min : search for $\{B_0, B_1, \cdots, B_k\}$ for min
  at most $O(\log n)$ trees for n nodes
  ⇒ $O(\log n)$

can be improved by keeping a pmin pointer to min-min-tree
* merge : $O(\log n)$, like adding to binary numbers
* delMin : remove and merge sub-trees w/ existing trees