Subject : 2011

No. : ?-1

Date : ...../...../.....

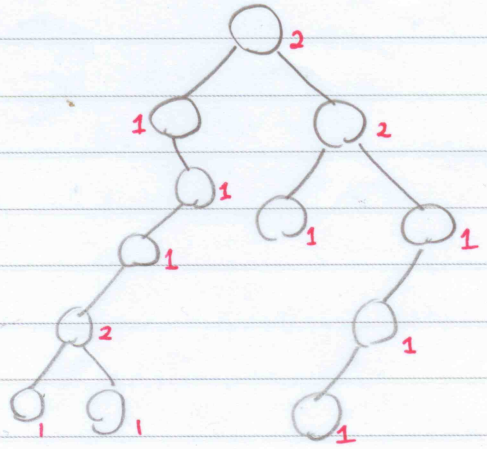
* more on priority queues

	max-heap	max-Ltree	usual max-tree
getMin	O(1)	O(1)	O(1)
insert	O(log n)	O(log n)	O(h)
delMin	O(log n)	O(log n)	O(h)
merge	O(n log n) w/ insert	O(log n)	O(1)
restriction	strict		loose
	complete bin. tree	leftist tree	any bin. tree

* leftist tree

"like to put nodes in the left"

let { shortest (node) = 1 + distance (closest decendant w/ < 2 children)
shortest (null) = 0



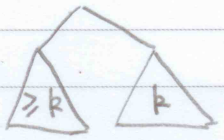
shortest (node)
= 1 + min (shortest (node → left) ,
shortest (node → right))

leftist : shortest (node → left) ≥ shortest (node → right)
for every node (subtree)

* why leftiest ? right-most path will be short

* for a leftist tree w/ shortest(root) = r ,
  # node $\geq 2^r - 1$

induction: when $r = 1$, # node ≥ 1
assume true when $r = k$
then when $r = k+1$

node in 

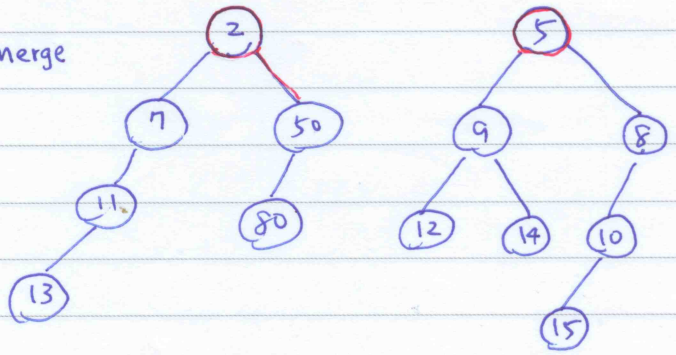
$\geq 1 + (2^k - 1) + (2^k - 1)$
left right
$\geq 2^{k+1} - 1$

* shortest(root) happens on the right-most path!

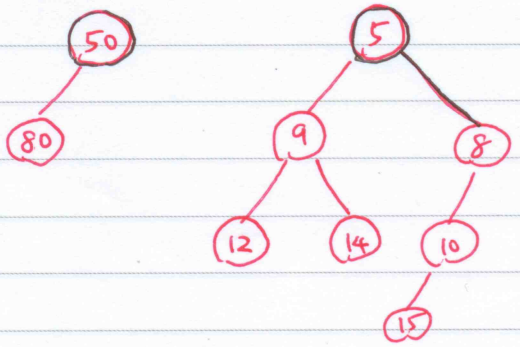
* merge: follows the right-most path

min-L Tree

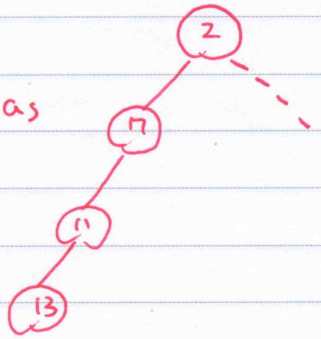
merge



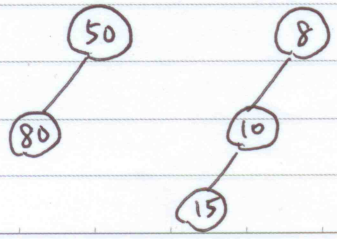
merge



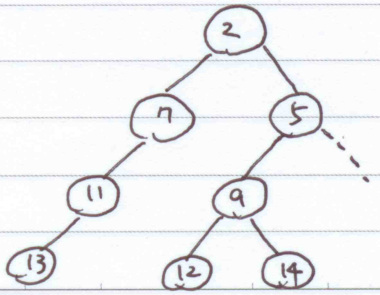
as

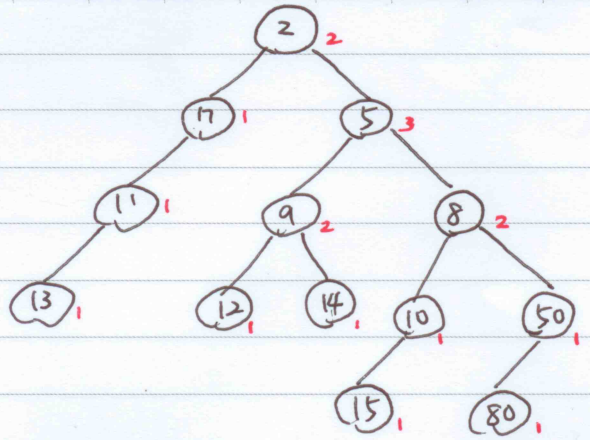


merge



as





swap (5) (17) to make leftist

* merge: { follow right-most path and continue merging
  go back and maintain leftist in every level
  $O(\log n)$ because right-most path $\leq \log_2(n+1)$
* insert : merge w/ single tree
* del Min: merge two sub-trees