*

data	
l	r

so far

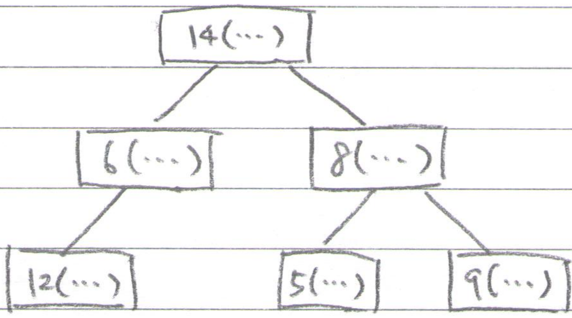
key	data
l	r

next

* task: find the node w/ largest

e.g. { key means priority
data is an entry to an item in your todo list

idea: put the node w/ largest key close to the root
— how about the root itself?

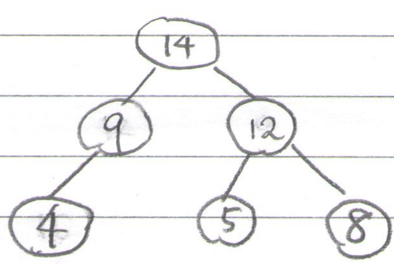


but after getting $\boxed{14(\cdots)}$, hard to get next (2nd-largest) node

* binary max-tree

① root key larger than (or =) key of other node
② every sub-tree is a bin. max-tree

$\boxed{\text{for each } v,\ \text{parent}(v) \to \text{key} \geq v \to \text{key}}$



GetLargest(T) { return T.root → data; }

RemoveLargest(T) { node ⇐ larger of two children of T.root;
replace T.root w/ node [in terms of content];
RemoveLargest(subtree at node);
}

* worst-case time of RemoveLargest ?

$O(h)$ and hence possibly $O(n)$

③ how about requiring a complete binary tree ?

$O(h) = O(\log n)$

called max-heap

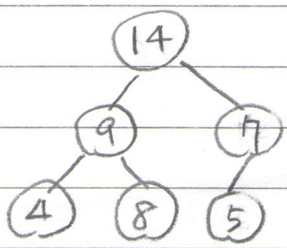
" but can we maintain it efficiently " ?

* removeMax

swap "last" to the "root", and roll down

* insert

put to "last", and roll up



removeMax ?

insert (10)

* complete binary tree → array (special)

max-heap → partially ordered array

$\boxed{\text{if there is a max-heap}}$ on an array

usual sel. sort

$O(n) \cdot \underbrace{O(n)}_{\text{selection}}$

heap sort

$O(n) \cdot O(\log n) = O(n \log n)$

* $\square$ from unsorted : $O(n \log n)$ by calling n insertion

or faster ! $O(n)$

reading assignment

* min-heap instead of max-heap in text book

key can be anything that is Comparable

ADT w/ insert & removeMax called priority-queue

- PQ w/ heap : $O(\log n)$ insertion $O(\log n)$ removal
- PQ w/ max-tree : $O(h)$ insertion $O(h)$ removal
- PQ w/ ordered linked list : $O(n)$ insertion $O(1)$ removal
- PQ w/ unordered linked list : $O(1)$ insertion $O(n)$ removal

STL : PQ w/ heap (on vector)

* heap sort

selection sort + max-heap

n iter $O(\log n)$ $\Rightarrow$ $O(n \log n)$ w/ only original array