* Worst-case of Remove-Largest? $O(h)$ with $h$ being height of $T$
so can be $O(n)$ for $n$ nodes

- how about requiring a complete binary tree?
  $O(h) = O(\log n)$

called max-heap

but need to check how to maintain.

* Heap-Remove-Largest ($T$) {
  compare LastNode $\rightarrow$ key with $T$ $\rightarrow$ left $\rightarrow$ key and $T$ $\rightarrow$ right $\rightarrow$ key
  if (LastNode $\rightarrow$ key $\geq$ largest) {
    replace $T$ $\rightarrow$ key, $T$ $\rightarrow$ data w/ LastNode $\rightarrow$ key / data
  }
  otherwise {
    replace $T$ $\rightarrow$ key, $T$ $\rightarrow$ data w/ Child $\rightarrow$ key / data
    (Heap-Remove-Largest (Child, Candidate))
  }
}

move back to the root, and trickle down

* Heap-Insert ($T$, Current) {
  while (Current $\rightarrow$ key $>$ Current $\rightarrow$ parent $\rightarrow$ key) {
    Swap Current and Current $\rightarrow$ parent;
    Current $=$ Current $\rightarrow$ parent;
  }
}

put in the back, and bubble up

* note: complete binary tree can be packed in an array

  max-heap is essentially a special array
  (not completely ordered, but follow some rules)
* Case 2: \[ ?? = k \]

- keys are words
- data are their explanations
- dictionary

Binary search revisited

\[
\text{Bin-Search}(k, \text{RangeL, RangeR}) \}\]

\[
middle = \ldots
\]

- if \( k < \text{middle} \)
  \[
  \text{Bin-Search}(k, \text{RangeL, RangeL-1})
  \]
- else if \( k > \text{middle} \)
  \[
  \text{Bin-Search}(k, \text{RangeR+1, RangeR})
  \]
- else
  \[
  \text{return location of middle}
  \]

- need
  \[
  \text{left subtree} < \text{root} < \text{right subtree}
  \]

- called binary search tree

- worst-case search time: \( O(h) \) w/ \( h \) being height of tree

- insert (also \( O(h) \))
  \[
  \text{if } (k < \text{middle})
  \]
  - insert to left-subtree
  \[
  \text{else if } (k > \text{middle})
  \]
  - insert to right-subtree

- delete (also \( O(h) \))
  - leaf: simple
  - one child: simple
  - two children: take right-most descendent of left-subtree as root

- join, split: READING ASSIGNMENT

- good binary search tree: balanced (\( h = O(\log n) \))
  \[
  \text{in practice: not always sure}
  \]

- randomly insert: yes in average

- challenge: binary search tree w/ still efficient insert/delete

Double A
heap: specially arranged complete binary tree
w/ application in simple priority queue
BST: specially arranged binary tree
w/ application in search (dictionary)
selection: general complete binary tree to process "tournament" data
w/ application in merging ordered lists

* l1: 9 8 7 3 1
  l2: 10 6 5 2

how to merge two
1. 9 8 7 6 5 3 2 1

output \( \max(\text{head}(l1), \text{head}(l2)) \)
remove the max from the associated list

\( O(N) \) for \( N \) elements (if removal is \( O(1) \))

* l1, l2, l3, l4?

output \( \max(\text{head}(l1), \text{head}(l2), \text{head}(l3), \text{head}(l4)) \)
remove

\( O(N \cdot \text{time}(\max)) \)
for \( k \) lists, \( \text{time}(\max) \) is \( O(k) \)

\( \Rightarrow O(Nk) \)
for naive implementation

* l1 9
  l2 8
  l3 10 6
  l4 5

Naive:
\[
\begin{align*}
(7, 5) & \quad (8, 10), (10, 5) \quad \Rightarrow \quad 10 \\
(7, 8) & \quad (8, 6), (8, 5) \quad \Rightarrow \quad 8 \\
(7, 6) & \quad (7, 5) \quad \Rightarrow \quad 7
\end{align*}
\]

\text{repeatedly checked}
* save time w/ tournaments

only (at most) the path from the new element to the root needs to be updated

\[ O(h) = O(\log_2 k) \] to maintain and \[ O(1) \] to find max
\[ \Rightarrow O(N \log k) \] to merge \( k \) ordered lists w/ a total of \( N \) elements

called max-winner tree (textbook: min-winner)

note: a bottom-up tree (leaf \( \rightarrow \) root)

* the path from leaf-10 to root all stores leaf-10
to rematch, need to find "sibling" (e.g. 5, 8)
can simplify finding sibling by storing sibling in non-leaf nodes + overall winner

called (max-) loser tree
i.e. sibling

* Section 5.9: Forest (READING ASSIGNMENT)
balance game & trees
9 coins A B C D E F G H I
one of them heavier
two uses of balance

how to find it?

ABC ? DEF
\[
\begin{array}{c}
\text{A} \quad \text{B} \\
\text{C} \quad \text{D} \\
\text{E}
\end{array}
\]

A < B > C < D < E
A < C < B < D < E
A < C < B < D < E

a complete trinary tree!
leaf: outcomes
non-leaf: conditions

two uses of balance = two non-leaf levels
⇒ at most 9 possibilities

if 10 coins (outcomes), probably impossible
if 9 coins, need to physically check if reasonable (like above)

brainstorm:
12 coins, 1 of them heavier or lighter, 3 uses of balance
13
14 impossible, why?