CS61B Lecture #23

es (*Data Structures* §6.4, §6.5)

s (§6.2)

s: `SortedSet`, `Map`, etc.

Hashing (*Data Structures* Chapter 7).

Priority Queues, Heaps

e: defined by operations "add," "find largest," "remove

cheduling long streams of actions to occur at various

or sorting (keep removing largest).

ementation is the *heap*, a kind of tree.

this same term is used to described the pool of storage operator uses. Sorry about that.)

Heaps

is a binary tree that enforces the

perty: Both labels in *both* children of each node are node's label.

op has largest label.

binary search property, which allows us to keep tree

always valid to put the smallest nodes anywhere at the e tree.

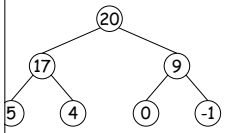
can be made *nearly complete:* all but possibly the last many keys as possible.

insertion of new value and deletion of largest value al- ne proportional to $\lg N$ in worst case.

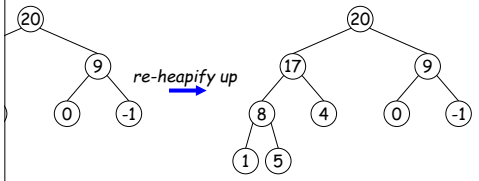
s basically the same, but with the minimum value at the dren having larger values than their parents.

mple: Inserting into a simple heap

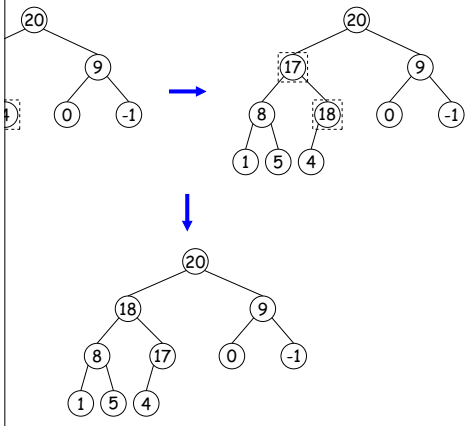
-1 20



boxes show where heap property violated

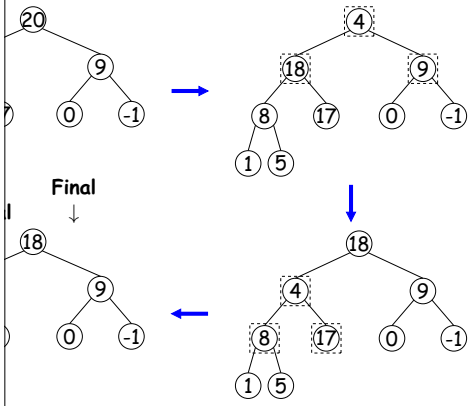


Heap insertion continued



Removing Largest from Heap

gest: Move bottommost, rightmost node to top, then as needed (swap offending node with larger child) to p property.



Ranges

looked for specific items

s, need an ordering anyway, and can also support looking
f values.

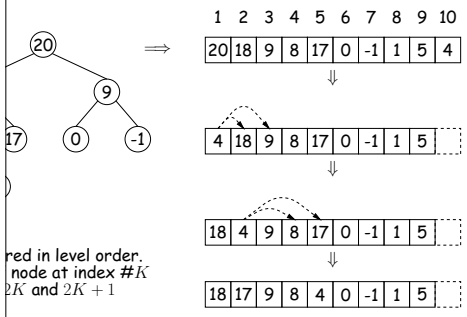
rform some action on all values in a BST that are within
in natural order):

```
TTODO to all labels in T that are
< U, in ascending natural order. */
isitRange (BST T, Comparable<Key> L, Comparable<Key> U,
           Action whatToDo)
ll) {
eft = L.compareTo (T.label ()),
ight = U.compareTo (T.label ());
eft < 0)   /* L < label */
nge (T.left (), L, U, whatToDo);
eft <= 0 && compRight > 0) /* L <= label < U */
o.action (T);
ight > 0)   /* label < U */
nge (T.right (), L, U, whatToDo);
```


Heaps in Arrays

are nearly complete (missing items only at bottom level),
ys for compact representation.

emoval from last slide (dashed arrows show children):



red in level order.
node at index #K
$2K$ and $2K+1$

red Sets and Range Queries in Java

Set supports range queries with *views* of set:

t(U): subset of S that is $<$ U.

t(L): subset that is $\geq$ L.

(L,U): subset that is $\geq$ L, $<$ U.

iews modify S.

, e.g., add to a headSet beyond U are disallowed.

hrough a view to process a range:

```
String> fauna = new TreeSet<String>
ys.asList ("axolotl", "elk", "dog", "hartebeest", "duck"));
g item : fauna.subSet ("bison", "gnu"))
.out.printf ("%s, ", item);
```

dog, duck, elk,"

type TreeSet<T> requires either that T be Comparable,
rovide a Comparator:

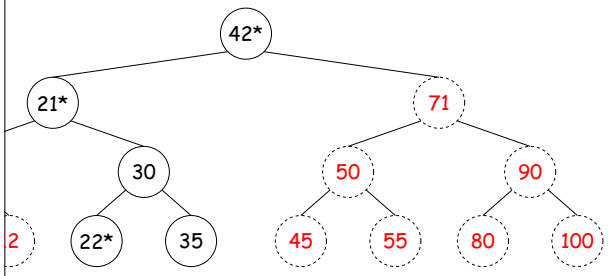
```
String> rev_fauna = new TreeSet<String> (Collections.reverseOrder());
```


Time for Range Queries

ge query $\in O(h+M)$, where h is height of tree, and M
data items that turn out to be in the range.

rching the tree below for all values, x, such that $25 \leq$

ple, the h comes from the starred nodes; the M comes
ion-dashed nodes. Dashed nodes are never looked at.



ample of Representation: BSTSet

entation for
d subsets.

BST, plus
y).

xpensive!

```
SortedSet<String>
    fauna = new BSTSet<String>(stuff);
    subset1 = fauna.subSet("bison","gnu");
    subset2 = subset1.subSet("axolotl","dog");
```