CS61B Lecture #24

es (*Data Structures* §6.4, §6.5)

s (§6.2)

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

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: Labels of *both* children of each node are less '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.

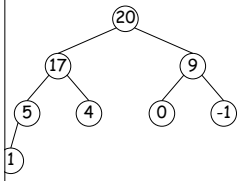
can be made *nearly complete:* all but possibly the last many keys as possible.

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

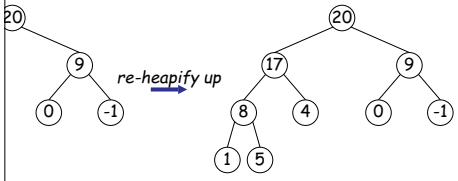
s basically the same, but with the minimum value at the dren having larger values than their parents.

mple: Inserting into a simple heap

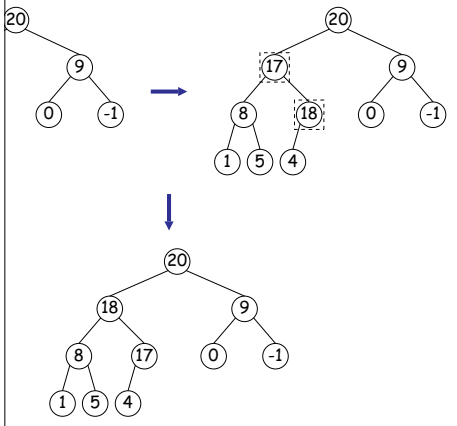
1 20



boxes show where heap property violated

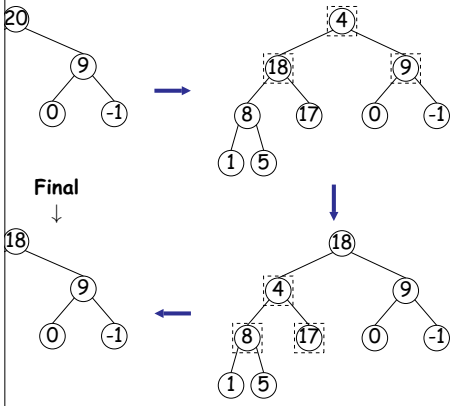


Heap insertion continued



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.

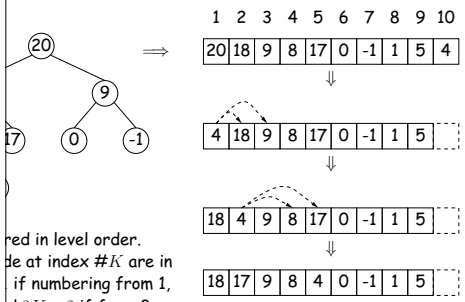
rform some action on all values in a BST that are within
in natural order):

```
WHATTODO to all labels in T that are >= L and < U,
ending natural order. */
d visitRange(BST<String> T, String L, String U,
             Consumer<BST<String>> whatToDo) {
 null) {
mpLeft = L.compareTo(T.label ()),
mpRight = U.compareTo(T.label ());
mpLeft < 0)                    /* L < label */
sitRange (T.left(), L, U, whatToDo);
mpLeft <= 0 && compRight > 0) /* L <= label < U */
atToDo.accept(T);
mpRight > 0)                   /* label < U */
sitRange (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.
de at index #K are in
if numbering from 1,
d $2K+2$ if from 0.

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);
```

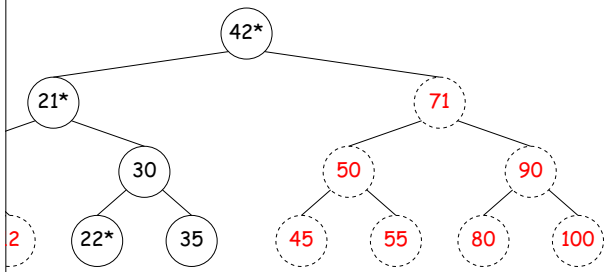
dog, duck, elk,"

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 $25 \leq x < 40$.

s are never looked at. Starred nodes are looked at but
The h comes from the starred nodes; the M comes from
n-dashed nodes.



TreeSet

type TreeSet<T> requires either that T be Comparable,
provide a Comparator, as in:

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

is a type of function object:

```
Comparator<T> {
turn <0 if LEFT<RIGHT, >0 if LEFT>RIGHT, else 0. */
mpare(T left, T right);
```

ith what Comparator<T extends Comparable<T>> is all

, the reverseOrder comparator is defined like this:

```
arator that gives the reverse of natural order. */
extends Comparable<T>> Comparator<T> reverseOrder() {
 figures out this lambda expression is a Comparator<T>.
(x, y) -> y.compareTo(x);
```

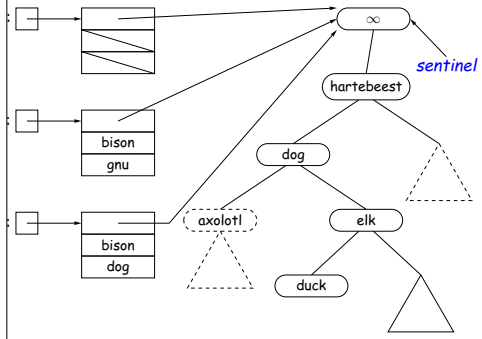

ample of Representation: BSTSet

entation for
d subsets.

BST, plus
y).

xpensive!

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



Comparing Search Structures

ms, k is #answers to query.

	Unordered List	Sorted Array	Bushy Search Tree	"Good" Hash Table	Heap
	$\Theta(N)$	$\Theta(\lg N)$	$\Theta(\lg N)$	$\Theta(1)$	$\Theta(N)$
ed)	$\Theta(1)$	$\Theta(N)$	$\Theta(\lg N)$	$\Theta(1)$	$\Theta(\lg N)$
	$\Theta(N)$	$\Theta(k + \lg N)$	$\Theta(k + \lg N)$	$\Theta(N)$	$\Theta(N)$
	$\Theta(N)$	$\Theta(1)$	$\Theta(\lg N)$	$\Theta(N)$	$\Theta(1)$
st	$\Theta(N)$	$\Theta(1)$	$\Theta(\lg N)$	$\Theta(N)$	$\Theta(\lg N)$