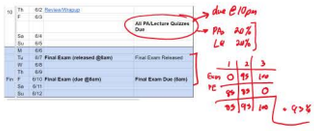
class CompInt implements Comparable<Integer> {
public int compare(Integer value1, Integer value2) {
if (value1 == value2) { return 0; }
else if (value1 < value2) { return -1; }
else { return 1; }
}

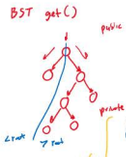
return value1 - value2;



In addition, you must average 65% on the programming assessments (exams) in order to pass the course.

compare()
Comparable<T>
Comparable<E>
compareTo()
0 if equal
-1 if less than
1 if greater than
class BST<K extends Comparable, V>

BST get()



public V get(K key) {
return get(key, root);
}

private V get(K key, Node n) {
if (n == null) { throw "Error the ...."; }
if (comparator.compare(N.getKey(), key) == 0) {
N.getKey().compareTo(key) == 0
return N.getValue();
} if
else if(comparator.compare(N.getKey(), key) > 0) {
node > key
return get(key, N.left);
}
else {
node < key
return get(key, N.right);
}
}

BST get

Circular Array List

int indexFor(int index) {
return (start + index) % array.length;
}

add(4)
value
indexFor(this.size)
0 + 0 = 0
array[0] = 4

add(10)
value
indexFor(this.size)
0 + 1 = 1
array[1] = 10

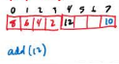
int remove(0)
index
save return = array[indexFor(0)]
0 + 0 = 0
4
return value
if (index == start) {
start += 1;
size -= 1;
start = start % array.length;
}
else { // middle, end
4 ← return temp

add(7)
indexFor(2)
1 + 1 = 2
array[2] = 7

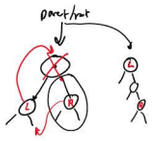
add(5)
add(1)
remove(1)
prepend(99)

size = 4
start = 1
copy 4

[prepend(2)
prepend(4)
start = 0 - 1 = -1 + capacity (5)
prepend(6)
prepend(8)
prepend(10)



add(12)



BST Remove

void remove(K key) {
if (root == null) {
// throw error
// return;
}
if (root.getKey().compareTo(key) == 0) {
// remove root
this.root = replace(this.root);
}
else if (root.getKey().compareTo(key) > 0) {
remove(K, root.left);
}
else { // go right root < key
remove(K, root.right);
}

void remove(K key, Node n) {
if (n == null) { return; // throw error }
int compare = N.getKey().compareTo(key);
if (compare > 0) {
if (n.left == null) { return; // throw error }
if (N.left.getKey().compareTo(key) == 0) {
N.left = replace(N.left);
}
else {
remove(key, N.left);
}
}
else {
if (N.right == null) { return; // throw error }
if (N.right.getKey().compareTo(key) == 0) {
N.right = replace(N.right);
}
else {
remove(key, N.right);
}
}
}

Node replace(Node n) {
if (left == null) { return n.right; }
Node right = n.right;
Node left = n.left;
add(left, right);
return left;
}

void add(Node top, Node bottom) {
if (bottom == null) { return; }
int compare = top.getKey().compareTo(bottom.getKey());
if (compare > 0) { // go left
if (top.left == null) {
top.left = bottom;
}
else {
add(top.left, bottom);
}
} else { // go right
if (top.right == null) {
top.right = bottom;
}
else {
add(top.right, bottom);
}
}
}