(a) Start by constructing a BST with these values in order: **H, W, D, L, Z, E, A**

(b) What are the values in the BST when visited in PRE-ORDER traversal?

**public** **void** preorder() { preorder(root); }

**private** **void** preorder(Node n) {

**if** (n != **null**) {

StdOut.*println* (n.key);

preorder (n.left);

preorder (n.right);

}

}  
  
(c) Let’s review what happens on deletion.

**public** **void** delete(Key key) { root = delete(root, key); }

**private** Node delete(Node parent, Key key) {

**if** (parent == **null**) **return** **null**;

// recurse until you find parent with this key.

**int** cmp = key.compareTo(parent.key);

**if** (cmp < 0) parent.left = delete(parent.left, key);

**else** **if** (cmp > 0) parent.right = delete(parent.right, key);

**else** {

// handle easy cases first:

**if** (parent.right == **null**) **return** parent.left;

**if** (parent.left == **null**) **return** parent.right;

// has two children: Plan on returning min of our right child

Node old = parent;

parent = min(old.right); // will eventually be "new parent"

// Note this is a simpler case: Delete min from right subtree

// and DON'T FORGET to stitch back in the original left child

parent.right = deleteMin(old.right);

parent.left = old.left;

}

// as recursions unwind, pass back potential new parent

**return** parent;

}