Objective(s):

a. To understand the BST form characteristic which depends on the order of the input.

b. To understand the process of BST deletion

Task 1:

As mentioned during the class, BST shape depends on the order of the input.

Implement /* your code 9 */ so that your BST can return its height.

```
public static void demo1() {
  int [] data = {2,1,3,4,5,6,7,8,9};
  bst = new BST();
  for (int j = 0; j < data.length; j++)
     bst.insert(data[j]);

  bst.printInOrder();
  println("Tree height = " + bst.height());

  int[] dat = { 15, 20, 10, 18, 16, 12, 8, 25, 19, 30};
  bst = new BST();
  for (int j = 0; j < dat.length; j++)
          bst.insert(dat[j]);

  bst.printInOrder();
  println("Tree height = " + bst.height());
}
```

```
class BST {
…
  public int height() {
    return root == null ? 0 : height(root);
  }
  public int height(TreeNode node) {
    if (node == null)
        return 0;
    return 999 /* your code 9 */;
  }
}
```

Instruction: Capture your int height(TreeNode node) and demo1()'s output

```
public int height(TreeNode node) {
    if (node == null)
        return 0;

    /*
     * Code 9
     */

    return 1 + Math.max(height(node.left), height(node.right));
}
```

```
[Demo 1]
2 1 3 4 5 6 7 8 9 Tree height: 8
15 10 8 12 20 18 16 19 25 30 Tree height: 4
```

Task 2:

To delete a node on a BST, it must know the node with the maximum value to replace the deleted node content.

```
public static void demo2() {
  println("node with max value " +
                    bst.findMaxFrom(bst.getRoot()));
}
```

```
class BST {
…
  public TreeNode findMaxFrom(TreeNode findMaxFrom) {
        /* your code 10 */
        return current;
  }
}
```

Implement /* your code 10 */

Instruction: Capture your int `findMaxFrom` (TreeNode `findMaxFrom`) and demo2()'s output

```
public TreeNode findMaxFrom(TreeNode findMaxFrom) {
    /*
     * Code 10
     */

    TreeNode current = findMaxFrom;

    while (current.right != null) {
        current = current.right;
    }

    return current;
}
```

```
[Demo 2]
node with max value null ← 30 → null
```

Task 3:

Implement /* your code 10 */

```
public static void demo3() {
  bst.delete(12, bst.getRoot());
  println(bst.search(20)); // 18<-20->25
  println(bst.search(25)); // null<-25->30
  println(bst.search(16)); // null<-16->null
  println(bst.search(10));  // 8<-10->null
  println(bst.search(12)); // not found
}
```

```
class BST {
…
  public void delete(int d, TreeNode current) {
    if (current == null) return; //not found
    if (d < current.data)
        delete(d, current.left);
    else if (d > current.data)
        delete(d, current.right);
    else { //found ... time to delete
        if (current.left == null || current.right == null) { // 0 or 1 child
           TreeNode q = (current.left == null) ? current.right : current.left;
           if (current.parent.left == current)
               current.parent.left = q; //this node is left child
           else
               current.parent.right = q;
           if (q != null) q.parent = current.parent;
        else { // two children
           TreeNode q = findMaxFrom(current.left);
           /* your code 11 */
        } // two children
    } //found
  }
}
```

```
public static void demo4() {
  int[] dat = { 15, 20, 10, 18, 16, 12, 8, 30, 19, 25 };
  bst = new BST();
  for (int j = 0; j < dat.length; j++) {
    bst.insert(dat[j]);
  }

  bst.printInOrder();
  bst.delete(20);           // default TreeNode is root
  bst.printInOrder();
  bst.delete(15);           // root -> complete the delete(int, TreeNode)
  bst.printInOrder();
}
```

Instruction: Capture your int `delete(int d, TreeNode current` and demo3() and demo4()'s output

Submission: this pdf Due date: TBA

```java
public void delete(int d, TreeNode current) {
    if (current == null)
        return; // not found

    if (d < current.val)
        delete(d, current.left);
    else if (d > current.val)
        delete(d, current.right);
    else { // found ... time to delete
        if (current.left == null || current.right == null) { // 0 or 1 child
            TreeNode q = (current.left == null) ? current.right : current.left;
            if (current.parent.left == current)
                current.parent.left = q; // this node is left child
            else
                current.parent.right = q;

            if (q != null)
                q.parent = current.parent;
        } else { // two children
                TreeNode w = findMaxFrom(current.left);
                /*
                 * Code 11
                 */

                current.val = w.val;
                delete(w.val, current.left);
        } // two children
    } // found
}
```