1. Consider the parse tree for \((9 + (5 \times 3)) / (8 - 4)\):

Indentify the following items in the above tree:

a. node containing “\(*\)”

b. edge from node containing “-” to node containing “8”

c. root node

d. children of the node containing “+”

e. parent of the node containing “3”

f. siblings of the node containing “\(*\)”

g. leaf nodes of the tree

h. subtree who’s root is node contains “+”

i. path from node containing “+” to node containing “5”

j. branch from root node to “3”

k. mark the levels of the tree (level is the number of edges on the path from the root)

l. What is the height (max. level) of the tree?

2. A traversal iterates through all the nodes in the tree. The four common binary tree traversals are:

<table>
<thead>
<tr>
<th>Preorder</th>
<th>Inorder</th>
<th>Postorder</th>
<th>Level order</th>
</tr>
</thead>
<tbody>
<tr>
<td>visit the root node</td>
<td>Inorder traverse left subtree</td>
<td>Postorder traverse left subtree</td>
<td>visit nodes at each level from left-to-right starting with level 0</td>
</tr>
<tr>
<td>Preorder traverse left subtree</td>
<td>visit the root node</td>
<td>Postorder traverse right subtree</td>
<td></td>
</tr>
<tr>
<td>Preorder traverse right subtree</td>
<td>Inorder traverse right subtree</td>
<td>visit the root node</td>
<td></td>
</tr>
</tbody>
</table>

a. What is the preorder traversal of the above expression tree?

b. What is the inorder traversal of the above expression tree?

c. What is the postorder traversal of the above expression tree?

d. What is the level-order traversal of the above expression tree?
3. Consider the Binary Search Tree (BST). For each node, the values in the left-subtree are < the node and the values in the right-subtree are all > the node.

\[
\begin{center}
\begin{array}{c}
50 \\
30 \quad 60 \\
9 \\
34 \\
18 \\
32 \\
47 \\
58 \\
80 \\
\end{array}
\end{center}
\]

a. What would be the result of an inorder traversal?

b. Starting at the root, how would you find the node containing “32”?

c. Starting at the root, when would you discover that “70” is not in the BST?

d. Starting at the root, where would be the “easiest” place to add “70”?

4. a) If a BST contains n nodes and we start searching at the root, what would be the worst-case \( \Theta( ) \) notation for a successful search? (Draw the shape of the BST leading to the worst-case search)

b) We could store a BST in an array like we did for a binary heap, what would be the worst-case storage needed for a BST with n nodes?

5. a) If a BST contains n nodes, draw the shape of the BST leading to best, successful search in the worst case.

b) What is the worst-case \( \Theta( ) \) notation for a successful search in this “best” shape BST?
6. Consider the Binary Search Tree (BST):

   50
  /   \
30   60
 /   / \
9   34  58  80
   /   / \
 18  32  47  

   a. What would be the result of deleting 58 from the BST?

   b. What would be the result of deleting 9 from the BST?

   c. What would be the result of deleting 50 from the BST? (Hint: One technique when programming is to convert a hard problem into a simpler problem. Deleting a BST node that contains two children is a hard problem. Since we know how to delete a BST node with none or one child, how could we convert “deleting a node with two children” problem into a simpler problem?)
<table>
<thead>
<tr>
<th>BINARY TREE METHOD</th>
<th>WHAT IT DOES</th>
</tr>
</thead>
<tbody>
<tr>
<td>T = BinaryTree(item)</td>
<td>Creates a new binary tree with <strong>item</strong> as the root and empty left and right subtrees. This is essentially a leaf node.</td>
</tr>
<tr>
<td>T.<strong>str</strong>()</td>
<td>Same as <strong>str(T)</strong>. Returns a string representation of the tree that shows its structure.</td>
</tr>
<tr>
<td>T.isEmpty()</td>
<td>Returns <strong>True</strong> if T is empty, or <strong>False</strong> otherwise.</td>
</tr>
<tr>
<td>T.preorder(aList)</td>
<td>Performs a preorder traversal of T. <strong>Postcondition:</strong> the items visited are added to <strong>aList</strong>.</td>
</tr>
<tr>
<td>T.inorder(aList)</td>
<td>Performs an inorder traversal of T. <strong>Postcondition:</strong> the items visited are added to <strong>aList</strong>.</td>
</tr>
<tr>
<td>T.postorder(aList)</td>
<td>Performs a postorder traversal of T. <strong>Postcondition:</strong> the items visited are added to <strong>aList</strong>.</td>
</tr>
<tr>
<td>T.levelorder(aList)</td>
<td>Performs a level order traversal of T. <strong>Postcondition:</strong> the items visited are added to <strong>aList</strong>.</td>
</tr>
<tr>
<td>T.getRoot()</td>
<td>Returns the item at the root. <strong>Precondition:</strong> T is not an empty tree.</td>
</tr>
<tr>
<td>T.getLeft()</td>
<td>Returns the left subtree. <strong>Precondition:</strong> T is not an empty tree.</td>
</tr>
<tr>
<td>T.getRight()</td>
<td>Returns the right subtree. <strong>Precondition:</strong> T is not an empty tree.</td>
</tr>
<tr>
<td>T.setRoot(item)</td>
<td>Sets the root to <strong>item</strong>. <strong>Precondition:</strong> T is not an empty tree.</td>
</tr>
<tr>
<td>T.setLeft(tree)</td>
<td>Sets the left subtree to <strong>tree</strong>. <strong>Precondition:</strong> T is not an empty tree.</td>
</tr>
<tr>
<td>T.setRight(tree)</td>
<td>Sets the right subtree to <strong>tree</strong>. <strong>Precondition:</strong> T is not an empty tree.</td>
</tr>
<tr>
<td>T.removeLeft()</td>
<td>Removes and returns the left subtree. <strong>Precondition:</strong> T is not an empty tree. <strong>Postcondition:</strong> the left subtree is empty.</td>
</tr>
<tr>
<td>T.removeRight()</td>
<td>Removes and returns the right subtree. <strong>Precondition:</strong> T is not an empty tree. <strong>Postcondition:</strong> the left subtree is empty.</td>
</tr>
</tbody>
</table>

**TABLE 18.3** The operations on a binary tree ADT