1. Consider the partial TreeNode class and partial BinarySearchTree class.

```python
class TreeNode:
    def __init__(self, key, val, left=None, right=None, parent=None):
        self.key = key
        self.payload = val
        self.leftChild = left
        self.rightChild = right
        self.parent = parent

    def hasLeftChild(self):
        return self.leftChild

    def hasRightChild(self):
        return self.rightChild

    def isLeftChild(self):
        return self.parent and \n        self.parent.leftChild == self

    def isRightChild(self):
        return self.parent and \n        self.parent.leftChild == self

    def isRoot(self):
        return not self.parent

    def isLeaf(self):
        return not (self.rightChild or self.leftChild)

    def hasAnyChildren(self):
        return self.rightChild or self.leftChild

    def hasBothChildren(self):
        return self.rightChild and self.leftChild

    def replaceNodeData(self, key, value, lc, rc):
        self.key = key
        self.payload = value
        self.leftChild = lc
        self.rightChild = rc

        if self.hasLeftChild():
            self.leftChild.parent = self

        if self.hasRightChild():
            self.rightChild.parent = self

def __iter__(self):
    if self:
        if self.hasLeftChild():
            for elem in self.leftChild:
                yield elem

        yield self.key

        if self.hasRightChild():
            for elem in self.rightChild:
                yield elem
```

```python
class BinarySearchTree:
    def __init__(self):
        self.root = None
        self.size = 0

    def length(self):
        return self.size

    def __len__(self):
        return self.size

    def iter(self):
        return self.root.__iter__()

    def __str__(self):
        """Returns a string representation of the tree
        rotated 90 degrees counter-clockwise""

        def strHelper(root, level):
            resultStr = ""
            if root:
                resultStr += strHelper(root.rightChild, level + 1)
                resultStr += "| " * level
                resultStr += str(root.key) + "\n"
                resultStr += strHelper(root.leftChild, level + 1)

            return resultStr

        return strHelper(self.root, 0)
```

a) How do the BinarySearchTree __iter__ and __str__ methods work?
More partial TreeNode class and partial BinarySearchTree class.

class BinarySearchTree:

    def __contains__(self, key):
        if self._get(key, self.root):
            return True
        else:
            return False

    def get(self, key):
        if self.root:
            res = self._get(key, self.root)
            if res:
                return res.payload
            else:
                return None
        else:
            return None

    def _get(self, key, currentNode):
        if not currentNode:
            return None
        elif currentNode.key == key:
            return currentNode
        elif key < currentNode.key:
            return self._get(key, currentNode.leftChild)
        else:
            return self._get(key, currentNode.rightChild)

    def __getitem__(self, key):
        return self.get(key)

    def __setitem__(self, key, val):
        self.put(key, val)

    def put(self, key, val):
        if self.root:
            self._put(key, val, self.root)
        else:
            self.root = TreeNode(key, val)
            self.size += 1

    def _put(self, key, val, currentNode):
        if key < currentNode.key:
            if currentNode.hasLeftChild():
                self._put(key, val, currentNode.leftChild)
            currentNode.leftChild = TreeNode(key, val, currentNode)
        elif key > currentNode.key:
            if currentNode.hasRightChild():
                self._put(key, val, currentNode.rightChild)
            currentNode.rightChild = TreeNode(key, val, currentNode)
        else:
            currentNode.payload = val
            self.size -= 1

b) The _get method is the "work horse" of BST search. It recursively walks currentNode down the tree until it finds key or becomes None.

In English, what are the base and recursive cases?

c) What is the put method doing?

d) Complete the recursive _put method.

e) Draw the "shape" of the BST after puts of:
   50, 60, 30, 70, 90, 40, 65

f) If "n" items are in the BST, what is put's: Best-case $O(\log n)$? Worst-case $O(n)$? Average-case $O(\log n)$?
2. More partial TreeNode class and partial BinarySearchTree class.

class BinarySearchTree:
    ...
    def delete(self, key):
        if self.size > 1:
            nodeToRemove = self._get(key, self.root)
            if nodeToRemove:
                self.remove(nodeToRemove)
                self.size = self.size-1
            else:
                raise KeyError('Error, key not in tree')
        elif self.size == 1 and self.root.key == key:
            self.root = None
            self.size = self.size - 1
        else:
            raise KeyError('Error, key not in tree')

    def __delitem__(self, key):
        self.delete(key)

    def remove(self, current_node):
        if current_node.isLeaf(): # leaf
            if current_node == current_node.parent.leftChild:
                current_node.parent.leftChild = None
            else:
                current_node.parent.rightChild = None
        elif current_node.hasBothChildren(): # interior
            succ = current_node.findSuccessor()
            succ.spliceOut()
            current_node.key = succ.key
            current_node.payload = succ.payload
        else: # this node has one child
            if current_node.hasLeftChild():
                if current_node.isLeftChild():
                    current_node.leftChild.parent = current_node.parent
                    current_node.parent.leftChild = current_node.leftChild
                elif current_node.isRightChild():
                    current_node.leftChild.parent = current_node.parent
                    current_node.parent.rightChild = current_node.leftChild
                else:
                    current_node.leftChild.replaceNodeData(current_node.leftChild.key,
                                                           current_node.leftChild.payload,
                                                           current_node.leftChild.leftChild,
                                                           current_node.leftChild.rightChild)
            else:
                if current_node.isLeftChild():
                    current_node.rightChild.parent = current_node.parent
                    current_node.parent.leftChild = current_node.rightChild
                elif current_node.isRightChild():
                    current_node.rightChild.parent = current_node.parent
                    current_node.parent.rightChild = current_node.rightChild
                else:
                    current_node.replaceNodeData(current_node.rightChild.key,
                                                   current_node.rightChild.payload,
                                                   current_node.rightChild.leftChild,
                                                   current_node.rightChild.rightChild)

a) Update picture where we delete a leaf.

b) Where in the code is each handled?

c) Draw all pictures deleting all nodes with one child.
1. Consider the Binary Search Tree (BST):

![BST Diagram]

a. What would need to be done to delete 32 from the BST? **set parents left or right child pointer to None**

b. What would need to be done to delete 9 from the BST? **disconnect by setting child's parent and parent's left or right to deleted node's child**

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, we can convert "deleting a node with two children" problem into a simpler problem by overwriting 50 with another node's value. Which nodes can be used to overwrite 50 and still maintain the BST ordering?**

d. Which node would the TreeNode's findSuccessor method return for succ if we are deleting 50 from the BST?

2. When the findSuccessor method is called how many children does the self node have?

3. How could we improve the findSuccessor method?

4. When the spliceOut method is called from remove how many children could the self node have?

5. How could we improve the spliceOut method?