TO RUN:

1. Download all files to the same location.
2. TO RUN THE DEMO:
    a. open a terminal with python
    b. type python bst_demo.py and hit enter. The demo input is [20, 10, 30, 5, 15, 25, 40]
3. TO RUN WITH YOUR OWN EXAMPLES:
    a. open a terminal with python
    b. type python
    c. type import binary_search_tree as bst
    d. type tree = bst.BinarySearchTree() and hit enter
    e. enter your list of numbers separated by spaces as below.
        i. 10 4 5 6 92 21
    f. type tree.buildBinarySearchTree() and hit enter.
    g. do any operations you wish after.

API Specifications

The comments '"comment"' give the functionality of each method.

```
TreeNode: node of tree containing left, right, parent, and value

BinarySearchTree(self,has_input=None): Contains methods to create and manipulate
trees

	if input is not False:
		self.num_lst = get_input()
		self.num_lst = parse_input()
	else:
		self.num_lst = num_lst

Algorithm

Create 2 Classes: TreeNode and BinarySearchTree

All methods are inside BinarySearchTree

def buildBinarySearchTree(self):
	"""creates the initial tree by inserting new nodes."""

	for num in self.num_lst:
		self.insert(num)

def insert(self,value):
	"""inserts a TreeNode into the tree."""
```

```
	if self.root is None:
		Treenode(value)
	else:
		current = self.root
		while True:
			if value < current.value and current.left is None:
				assign_left(value)
				assign_parent(current)
			move_left()
			else:
				if current.right is None:
					assign_right(value)
					assign_parent(current)
				move_right()

def insert_at(self,index,value):
	"""inserts a value into a specific index in the list.
	This can cause the tree to be unsorted. The all methods
	should still work on unsorted tree. """
	if index < 0:#Check non-negative
		raise ValueError
	if index == 0: #New root
		new_node =  TreeNode(value)
		new_node.left = self.root
		if self.root:
			self.root.parent = new_node
		self.root = new_node

	else: #Indices greater than 0
		inorder = inorderTraversal()
		if index > len(inorder):
			ValueError
		parent_value = inorder[index]
		find_parent_node()
		create_new_treeNode()
		if parent.left is None:
			assign_left()
		else:
			insert_node_between_parent_and_existing_left()
		new_node.parent = parent



def inorderTraversal(self,prints):
	'''traverses in order on the tree to print sorted list'''
	initialize_stack_and_list_and_output()
	while stack or current:
```

```
            while current:
                  append_to_stack()
                  move_left()
            out_str += str(current.value)
            append_to_result()
      if prints is True:
            print(result)
      return result

def find_node(self,current,value):
      '''gets to the location of the node with a given value on the tree'''
      if current is None:
            return None
      if current.value==value:
            return current
      if value < current.value:
            return self.find_node(current.left,value) #search_left()
      else:
            return self.find_node(current.right,value) #search_right()


def delete(self,x):
      '''deletes the node from the tree and reorganizes tree appropietly'''
      node_to_delete = self.find_node(self.root,x)
      if node_to_delete is None:
            print_error_string()
      if node has no children:
            self.delete_node(node_to_delete) #defined later
      elif node_to_delete has one child:
            if left_child_exists
                  child = node_to_delete.left
            else: #if right child exists
                  child = node_to_delete.right

            if node has no parent (is the root):
                  assign_child()
            else:#if node to delete is not the root
                  update_childs_parents()
                  update_parent_reference_to_child()

      else: #If node_to_delete has 2 children
                  successor = find_min(node_to_delete.right) #Defined later but
basically find the min value in right tree
                  replace_node_to_delete_value_with_successor()
                  delete_node(successor) #Recursively delete the successor node

      return self.inorderTraversal() #per project guidelines
```

```
def find_min(self,node):
	'''finds the minimum value in the tree'''
	while node.left:
		move_left()
	return node

def delete_node(self,node):
	'''actually deletes the node from the tree'''
	if node has no parent:
		self.root = None
	elif node.parent.left == node:
		node.parent.left = None
	else:
		node.parent.right = None

def firstCommonAncestor(self,x,y,root=False):
	'''finds the first common ancestor of two nodes'''
	if root is False:#Use trees root if not provided. Start
		root = self.root
	if root is None:
		return None
	if current_root = x or y:
		return current_root
	recursively_search_left()
	recursively_search_right()
	if left and right in subtrees:
		return current_root
	if only one ancestor found:
		return ancestor

##############################
# I added functionality to print the tree to aide with
# the development of this project and to check my answers.

    def printTree(self, node=False, depth=0):
        if node is False:#First Case
            node = self.root
        if node is None: #End of tree/subtree
            return
        if depth == 0:
            print("Binary Search Tree:")
        if node is not None: #Recursively print right and left
            self.printTree(node.right, depth + 1)
            print("    |" * depth + "-- " + str(node.value))
            self.printTree(node.left, depth + 1)
```

Example 1 [20, 10, 30, 5, 15, 25, 40]:

In [90]:

```
bst = BinarySearchTree(True)
bst.buildBinarySearchTree()
print("Inorder Traversal of Binary Search Tree:")
bst.inorderTraversal()
print('Inserting 210 at index 2')
bst.insert_at(2,210)
bst.inorderTraversal()
print('First common ancestor 210 and 30')
print(bst.firstCommonAncestor(210, 30))
print('Deleting 20')
bst.delete(20)
bst.inorderTraversal()
print('Deleting 10')
bst.delete(10)
bst.inorderTraversal()
print('Inserting 10 at index 4')
bst.insert_at(4,10)
bst.inorderTraversal()
print('First Common 15,30')
bst.firstCommonAncestor(15,30)
bst.printTree()
```

```
Inorder Traversal of Binary Search Tree:
5; 10; 15; 20; 25; 30; 40;
Inserting 210 at index 2
5; 10; 210; 15; 20; 25; 30; 40;
First common ancestor 210 and 30
20
Deleting 20
5; 10; 210; 15; 25; 30; 40;
Deleting 10
5; 210; 15; 25; 30; 40;
Inserting 10 at index 4
5; 210; 15; 25; 10; 30; 40;
First Common 15,30
    |    |-- 40
    |-- 30
    |    |-- 10
-- 25
    |    |-- 15
    |-- 210
    |    |-- 5
```

EXAMPLE 2 User Input 10 3 4 99 24 6 32 8 7 2 9 10 26 28 37 84 100 46 483 72

In [10]:

```
bst = BinarySearchTree()
bst.buildBinarySearchTree()
print("Inorder Traversal of Binary Search Tree:")
bst.inorderTraversal()
print('Inserting 210 at index 2')
bst.insert_at(2,210)
bst.inorderTraversal()
print('First common ancestor 210 and 30')
print(bst.firstCommonAncestor(210, 30))
print('Deleting 20')
bst.delete(20)
bst.inorderTraversal()
print('Deleting 10')
bst.delete(10)
bst.inorderTraversal()
print('Inserting 10 at index 4')
bst.insert_at(4,10)
bst.inorderTraversal()
print('First Common 72,100')
print(bst.firstCommonAncestor(72,100))
bst.printTree()
```

```
Please input a list of numbers separated by spaces: 10 3 4 99 24 6 32 8 7 2 9 10 26 28 37 84 100 46 483 72
Inorder Traversal of Binary Search Tree:
2; 3; 4; 6; 7; 8; 9; 10; 10; 24; 26; 28; 32; 37; 46; 72; 84; 99; 100; 483;
Inserting 210 at index 2
2; 3; 210; 4; 6; 7; 8; 9; 10; 10; 24; 26; 28; 32; 37; 46; 72; 84; 99; 100; 483;
First common ancestor 210 and 30
210
Deleting 20
20 is not in the tree
2; 3; 210; 4; 6; 7; 8; 9; 10; 10; 24; 26; 28; 32; 37; 46; 72; 84; 99; 100; 483;
2; 3; 210; 4; 6; 7; 8; 9; 10; 10; 24; 26; 28; 32; 37; 46; 72; 84; 99; 100; 483;
Deleting 10
2; 3; 210; 4; 6; 7; 8; 9; 10; 24; 26; 28; 32; 37; 46; 72; 84; 99; 100; 483;
Inserting 10 at index 4
2; 3; 210; 4; 10; 6; 7; 8; 9; 10; 24; 26; 28; 32; 37; 46; 72; 84; 99; 100; 483;
First Common 72,100
99
Binary Search Tree:
    |    |    |-- 483
    |    |-- 100
    |-- 99
    |    |    |    |    |-- 84
    |    |    |    |    |    |    |-- 72
    |    |    |    |    |    |-- 46
    |    |    |    |-- 37
    |    |    |-- 32
    |    |    |    |    |-- 28
    |    |    |    |-- 26
    |    |-- 24
-- 10
    |    |    |    |    |-- 9
    |    |    |    |-- 8
    |    |    |    |    |-- 7
    |    |    |-- 6
    |    |    |    |-- 10
    |    |-- 4
    |    |    |-- 210
    |-- 3
    |    |-- 2
```