

Problem 1: Populate Next Right pointers of Tree class TreeNode:

```
def __init__(self, val=0, left=None, right=None, next=None):
```

```
    self.val = val
```

```
    self.left = left
```

```
    self.right = right
```

```
    self.next = next
```

```
def connect(root):
```

```
    if not root:
```

```
        return None
```

```
    level_start = root
```

```
    while level_start:
```

```
        current = level_start
```

```
        while current:
```

```
            if current.left:
```

```
                current.left.next = current.right
```

```
            if current.right and current.next:
```

```
                current.right.next = current.next.left
```

```
            current = current.next
```

```
        level_start = level_start.left
```

```
    return root
```

```
root = TreeNode(1)
```

```
root.left = TreeNode(2)
```

```
root.right = TreeNode(3)
```

```
root.left.left = TreeNode(4)
```

```
root.left.right = TreeNode(5)
```

```
root.right.left = TreeNode(6)
```

```
root.right.right = TreeNode(7)
```

```
connect(root)
```

```
print(root.val, "->", root.next)
```

```
print(root.left.val, "->", root.left.next.val)
```

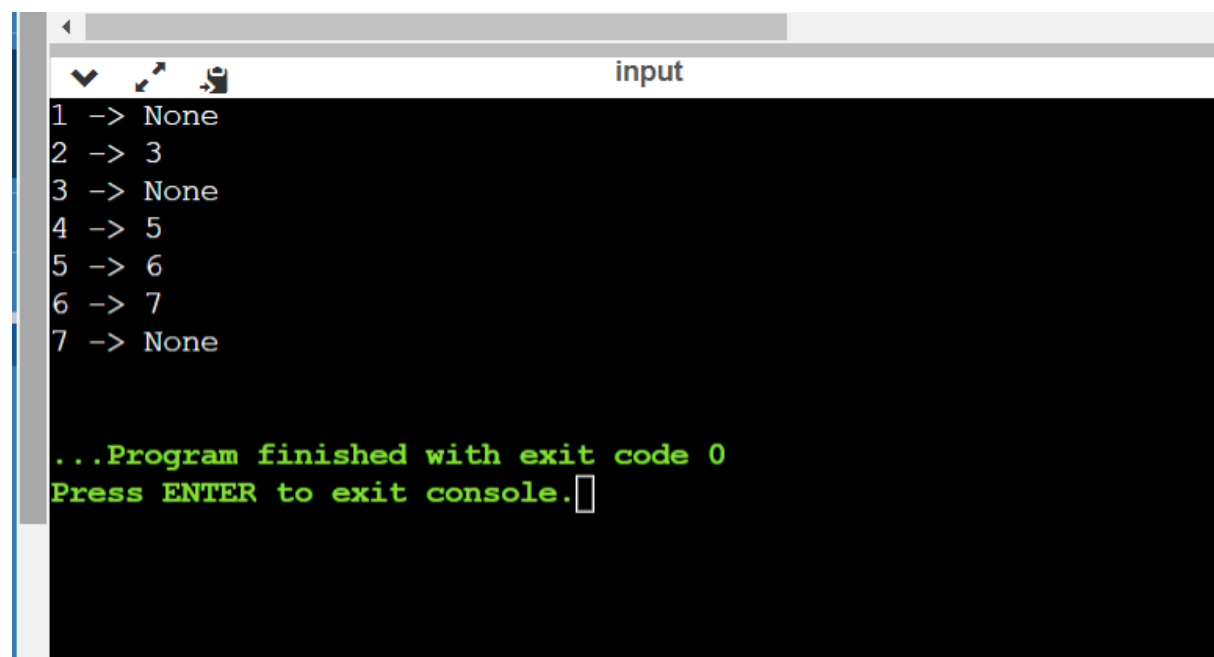
```
print(root.right.val, "->", root.right.next)
```

```
print(root.left.left.val, "->", root.left.left.next.val)
```

```
print(root.left.right.val, "->", root.left.right.next.val)
```

```
print(root.right.left.val, "->", root.right.left.next.val)
```

```
print(root.right.right.val, "->", root.right.right.next)
```



```
input
1 -> None
2 -> 3
3 -> None
4 -> 5
5 -> 6
6 -> 7
7 -> None

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 2: Search given Key in BST

class Node:

```
def __init__(self, key):  
    self.key = key  
    self.left = None  
    self.right = None
```

class BST:

```
def __init__(self):  
    self.root = None
```

```
def insert(self, key):  
    self.root = self._insert_recursive(self.root, key)
```

```
def _insert_recursive(self, root, key):  
    if root is None:  
        return Node(key)  
    if key < root.key:  
        root.left = self._insert_recursive(root.left, key)  
    elif key > root.key:  
        root.right = self._insert_recursive(root.right, key)  
    return root
```

```
def search(self, key):  
    return self._search_recursive(self.root, key)
```

```
def _search_recursive(self, root, key):  
    if root is None or root.key == key:  
        return root  
    if key < root.key:  
        return self._search_recursive(root.left, key)
```

```
return self._search_recursive(root.right, key)
```

```
bst = BST()
```

```
bst.insert(8)
```

```
bst.insert(3)
```

```
bst.insert(10)
```

```
bst.insert(1)
```

```
bst.insert(6)
```

```
bst.insert(14)
```

```
bst.insert(4)
```

```
bst.insert(7)
```

```
bst.insert(13)
```

```
key_to_search = 6
```

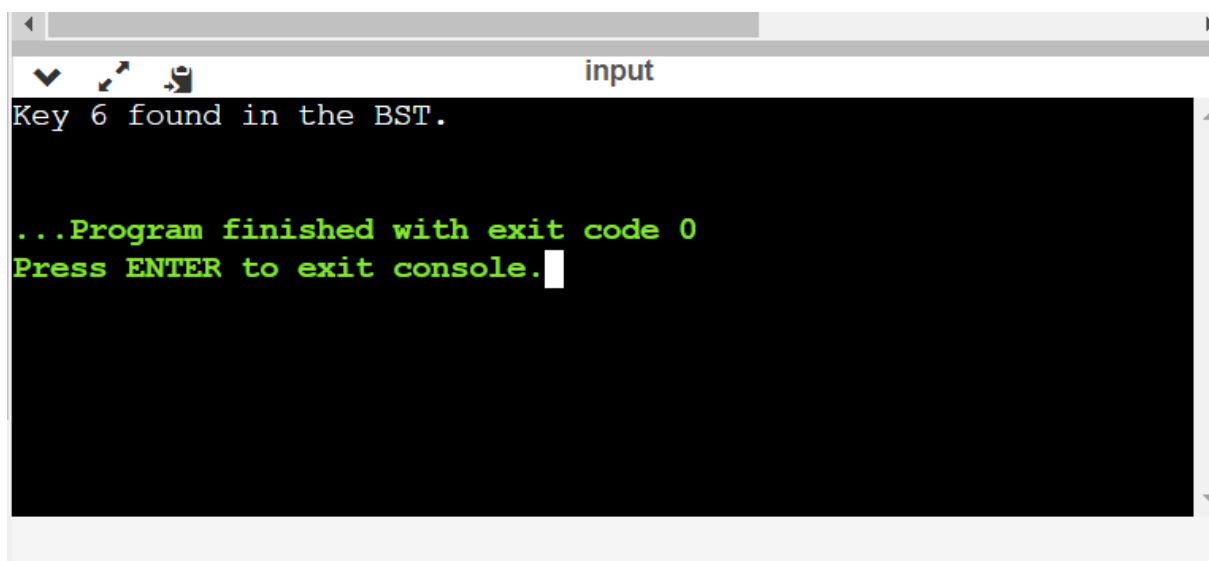
```
result = bst.search(key_to_search)
```

```
if result:
```

```
    print(f"Key {key_to_search} found in the BST.")
```

```
else:
```

```
    print(f"Key {key_to_search} not found in the BST.")
```



```
Key 6 found in the BST.  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Problem 3: Construct BST from given keys

class Node:

```
def __init__(self, key):
```

```
    self.key = key
```

```
    self.left = None
```

```
    self.right = None
```

```
def construct_bst(keys):
```

```
    if not keys:
```

```
        return None
```

```
    root = Node(keys[0])
```

```
    for key in keys[1:]:
```

```
        insert_node(root, key)
```

```
    return root
```

```
def insert_node(root, key):
```

```
    if key < root.key:
```

```
        if root.left:
```

```
            insert_node(root.left, key)
```

```
        else:
```

```
            root.left = Node(key)
```

```
    else:
```

```
        if root.right:
```

```
            insert_node(root.right, key)
```

```
        else:
```

```
            root.right = Node(key)
```

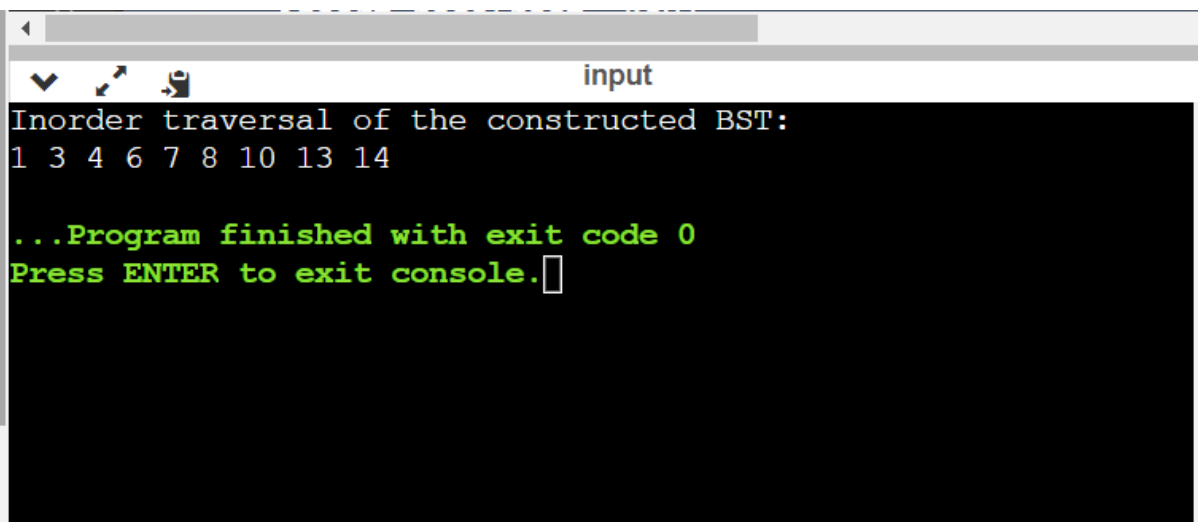
```
def inorder_traversal(node):  
    if node:  
        inorder_traversal(node.left)  
        print(node.key, end=" ")  
        inorder_traversal(node.right)
```

```
keys = [8, 3, 10, 1, 6, 14, 4, 7, 13]
```

```
bst_root = construct_bst(keys)
```

```
print("Inorder traversal of the constructed BST:")
```

```
inorder_traversal(bst_root)
```

A screenshot of a terminal window titled "input". The terminal has a black background with white and green text. The output shows the inorder traversal of a BST constructed from the keys [8, 3, 10, 1, 6, 14, 4, 7, 13]. The output is "Inorder traversal of the constructed BST:" followed by the numbers "1 3 4 6 7 8 10 13 14" on the next line. Below this, it says "...Program finished with exit code 0" and "Press ENTER to exit console." with a cursor at the end.

```
input  
Inorder traversal of the constructed BST:  
1 3 4 6 7 8 10 13 14  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Problem 4: Construct a BST from a preorder traversal

```
class TreeNode:
```

```
    def __init__(self, value):
```

```
        self.val = value
```

```
        self.left = None
```

```
        self.right = None
```

```
def construct_bst(preorder):
```

```
    if not preorder:
```

```
        return None
```

```
    root = TreeNode(preorder[0])
```

```
    stack = [root]
```

```
    for value in preorder[1:]:
```

```
        node = TreeNode(value)
```

```
        if value < stack[-1].val:
```

```
            stack[-1].left = node
```

```
        else:
```

```
            while stack and value > stack[-1].val:
```

```
                last = stack.pop()
```

```
            last.right = node
```

```
        stack.append(node)
```

```
    return root
```

```

def inorder_traversal(root):
    if root is None:
        return []

    result = []
    stack = []

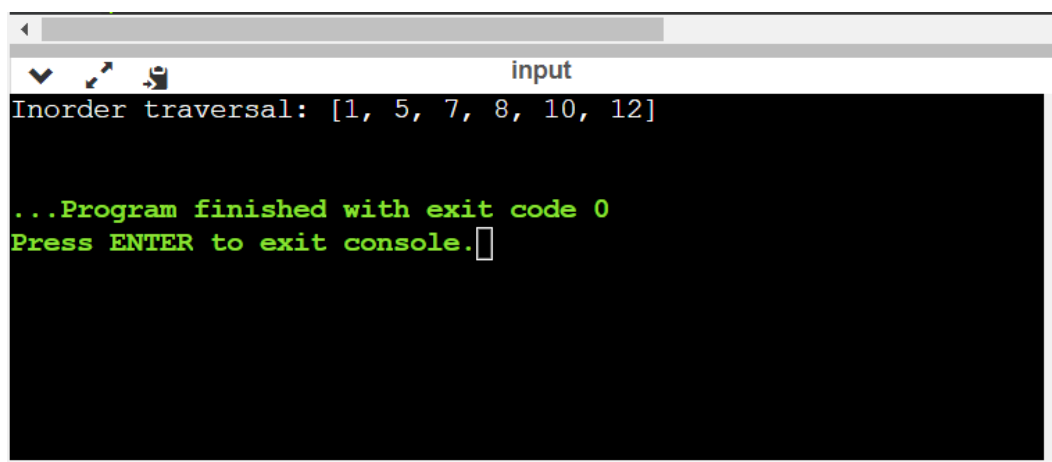
    while stack or root:
        if root:
            stack.append(root)
            root = root.left
        else:
            node = stack.pop()
            result.append(node.val)
            root = node.right

    return result

preorder = [8, 5, 1, 7, 10, 12]
bst = construct_bst(preorder)

inorder = inorder_traversal(bst)
print("Inorder traversal:", inorder)

```



The screenshot shows a terminal window titled "input". The output of the program is displayed in a monospaced font. The first line shows the inorder traversal result: "Inorder traversal: [1, 5, 7, 8, 10, 12]". The second line shows the program's exit status: "...Program finished with exit code 0". The third line shows a prompt to exit the console: "Press ENTER to exit console." followed by a cursor icon.

```

input
Inorder traversal: [1, 5, 7, 8, 10, 12]

...Program finished with exit code 0
Press ENTER to exit console.

```


Problem 5: Check is a BT is BST or not

```
class Node:
```

```
    def __init__(self, value):
```

```
        self.value = value
```

```
        self.left = None
```

```
        self.right = None
```

```
def is_bst(node, min_value=float('-inf'), max_value=float('inf')):
```

```
    if node is None:
```

```
        return True
```

```
    if node.value <= min_value or node.value >= max_value:
```

```
        return False
```

```
    return (
```

```
        is_bst(node.left, min_value, node.value) and
```

```
        is_bst(node.right, node.value, max_value)
```

```
    )
```

```
root = Node(4)
```

```
root.left = Node(2)
```

```
root.right = Node(6)
```

```
root.left.left = Node(1)
```

```
root.left.right = Node(3)
```

```
root.right.left = Node(5)
```

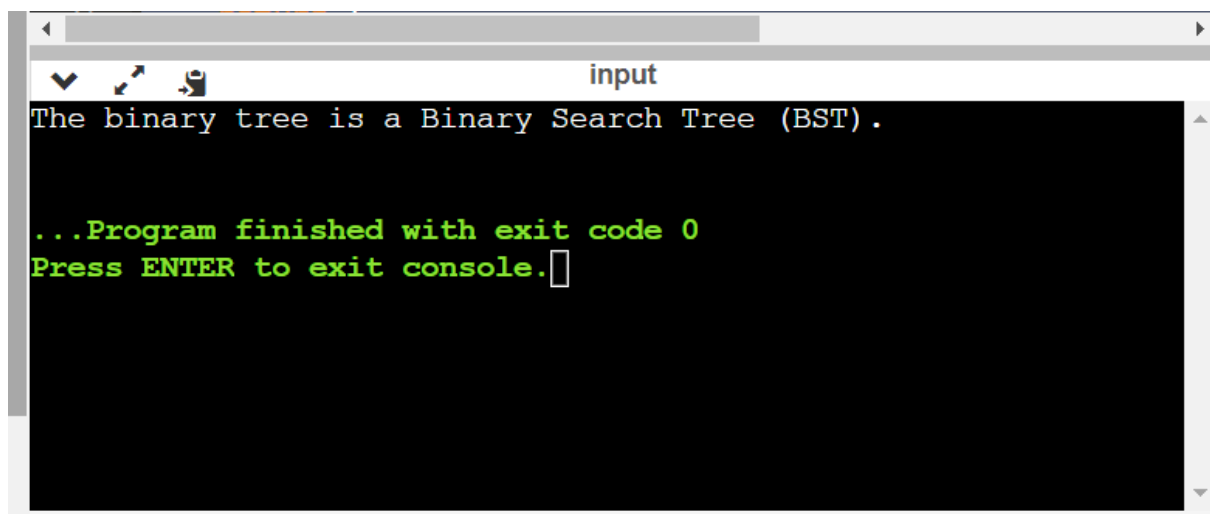
```
root.right.right = Node(7)
```

```
if is_bst(root):
```

```
    print("The binary tree is a Binary Search Tree (BST).")
```

```
else:
```

```
print("The binary tree is not a Binary Search Tree (BST).")
```



The screenshot shows a terminal window with a title bar that includes standard window controls and the title "input". The terminal has a black background with white text. The first line of output is "The binary tree is a Binary Search Tree (BST) .". The second line is "...Program finished with exit code 0". The third line is "Press ENTER to exit console." followed by a cursor. The terminal window is part of a larger application interface, as indicated by the window title bar and the presence of a scrollbar on the right.

```
input
The binary tree is a Binary Search Tree (BST) .
...Program finished with exit code 0
Press ENTER to exit console.█
```

Problem 6: Find LCA of two nodes in BST

```
class TreeNode:
```

```
    def __init__(self, val):
```

```
        self.val = val
```

```
        self.left = None
```

```
        self.right = None
```

```
def insert(root, val):
```

```
    if root is None:
```

```
        return TreeNode(val)
```

```
    if val < root.val:
```

```
        root.left = insert(root.left, val)
```

```
    else:
```

```
        root.right = insert(root.right, val)
```

```
    return root
```

```
def find_lca(root, p, q):
```

```
    if root is None:
```

```
        return None
```

```
    if p.val < root.val and q.val < root.val:
```

```
        return find_lca(root.left, p, q)
```

```
    if p.val > root.val and q.val > root.val:
```

```
        return find_lca(root.right, p, q)
```

```
    return root
```

```
root = None
```

```
root = insert(root, 6)
```

```
root = insert(root, 2)
```

```
root = insert(root, 8)
```

```
root = insert(root, 0)
```

```
root = insert(root, 4)
```

```
root = insert(root, 7)
```

```
root = insert(root, 9)
```

```
root = insert(root, 3)
```

```
root = insert(root, 5)
```

```
node_p = TreeNode(2)
```

```
node_q = TreeNode(8)
```

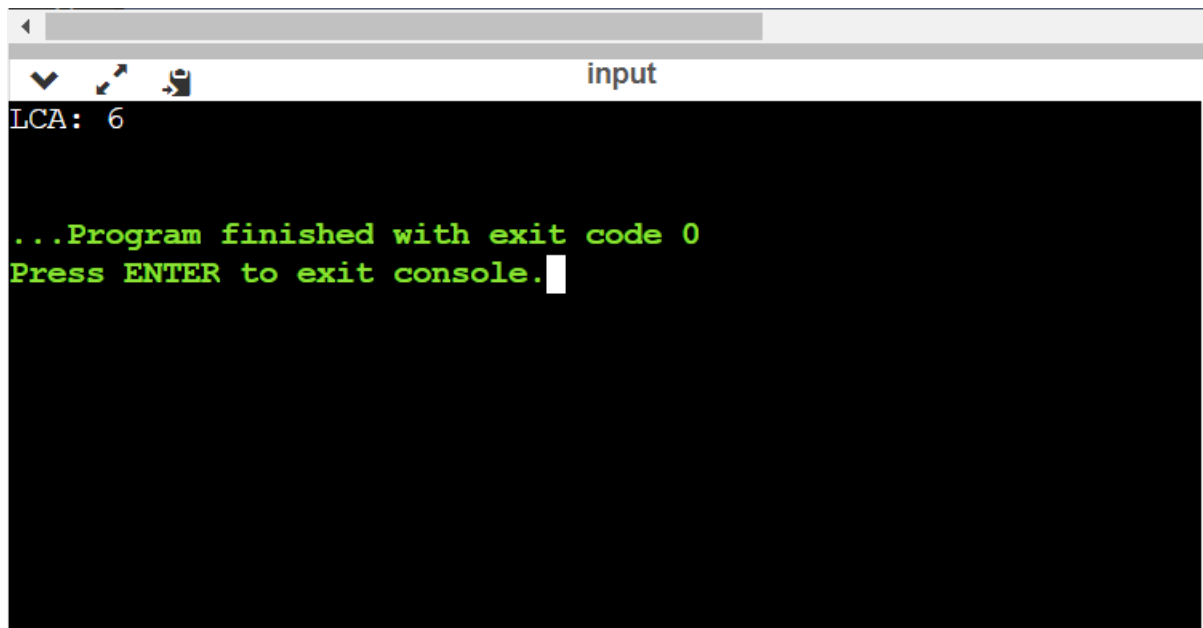
```
lca = find_lca(root, node_p, node_q)
```

```
if lca:
```

```
    print("LCA:", lca.val)
```

```
else:
```

```
print("LCA not found.")
```



A screenshot of a terminal window with a title bar that says "input". The terminal has a black background with green text. The first line of output is "LCA: 6". The second line is "...Program finished with exit code 0". The third line is "Press ENTER to exit console." followed by a white cursor block.

```
LCA: 6

...Program finished with exit code 0
Press ENTER to exit console.
```