Qno.1)

class TreeNode:

def \_\_init\_\_(self, value=0, left=None, right=None):

self.value = value

self.left = left

self.right = right

def triple\_leaves(root):

if root is None:

return

if root.left is None and root.right is None:

root.value \*= 3

return

triple\_leaves(root.left)

triple\_leaves(root.right)

def inorder\_traversal(root):

if root is None:

return

inorder\_traversal(root.left)

print(root.value, end=" ")

inorder\_traversal(root.right)

if \_\_name\_\_ == "\_\_main\_\_":

# Example binary tree

# 1

# / \

# 2 3

# / \ / \

# 4 5 6 7

root = TreeNode(1, TreeNode(2, TreeNode(4), TreeNode(5)), TreeNode(3, TreeNode(6), TreeNode(7)))

print("Inorder traversal before tripling leaves:")

inorder\_traversal(root)

print()

triple\_leaves(root)

print("Inorder traversal after tripling leaves:")

inorder\_traversal(root)

print()

Qno.2)

class TreeNode:

def \_\_init\_\_(self, value=0, left=None, right=None):

self.value = value

self.left = left

self.right = right

def fib\_tree(n):

if n <= 1:

return TreeNode(n)

left\_child = fib\_tree(n - 1)

right\_child = fib\_tree(n - 2)

value = left\_child.value + right\_child.value

return TreeNode(value, left\_child, right\_child)

def inorder\_traversal(root):

if root is None:

return

inorder\_traversal(root.left)

print(root.value, end=" ")

inorder\_traversal(root.right)

if \_\_name\_\_ == "\_\_main\_\_":

n1 = 6

root1 = fib\_tree(n1)

print(f"Inorder traversal for Fib\_tree({n1}):")

inorder\_traversal(root1)

print()

n2 = 7

root2 = fib\_tree(n2)

print(f"Inorder traversal for Fib\_tree({n2}):")

inorder\_traversal(root2)

print()

Qno.3)

class TreeNode:

def \_\_init\_\_(self, value=0, left=None, right=None):

self.value = value

self.left = left

self.right = right

def prune\_tree(root, k):

if root is None or k == 0:

return None

left\_child = prune\_tree(root.left, k - 1)

right\_child = prune\_tree(root.right, k - 1)

return TreeNode(root.value, left\_child, right\_child)

def inorder\_traversal(root):

if root is None:

return

inorder\_traversal(root.left)

print(root.value, end=" ")

inorder\_traversal(root.right)

if \_\_name\_\_ == "\_\_main\_\_":

# Example binary tree

# 1

# / \

# 2 3

# / \ / \

# 4 5 6 7

root = TreeNode(1, TreeNode(2, TreeNode(4), TreeNode(5)), TreeNode(3, TreeNode(6), TreeNode(7)))

print("Inorder traversal of original tree:")

inorder\_traversal(root)

print()

k = 3

pruned\_tree = prune\_tree(root, k)

print(f"Inorder traversal of pruned tree (depth {k}):")

inorder\_traversal(pruned\_tree)

print()

Qno.4)

class TreeNode:

def \_\_init\_\_(self, value=0, left=None, right=None):

self.value = value

self.left = left

self.right = right

def eval\_tree(root):

if root is None:

return 0

if isinstance(root.value, int) or isinstance(root.value, float):

return root.value

left\_value = eval\_tree(root.left)

right\_value = eval\_tree(root.right)

if root.value == '+':

return left\_value + right\_value

elif root.value == '-':

return left\_value - right\_value

elif root.value == '\*':

return left\_value \* right\_value

elif root.value == '/':

return left\_value / right\_value

elif root.value == '//':

return left\_value // right\_value

elif root.value == '%':

return left\_value % right\_value

elif root.value == '\*\*':

return left\_value \*\* right\_value

if \_\_name\_\_ == "\_\_main\_\_":

# Example arithmetic expression binary tree

# +

# / \

# 3 \*

# / \

# + 2

# / \

# 5 9

expr\_tree = TreeNode('+', TreeNode(3), TreeNode('\*', TreeNode('+', TreeNode(5), TreeNode(9)), TreeNode(2)))

result = eval\_tree(expr\_tree)

print("The value of the arithmetic expression represented by the binary tree is:", result)