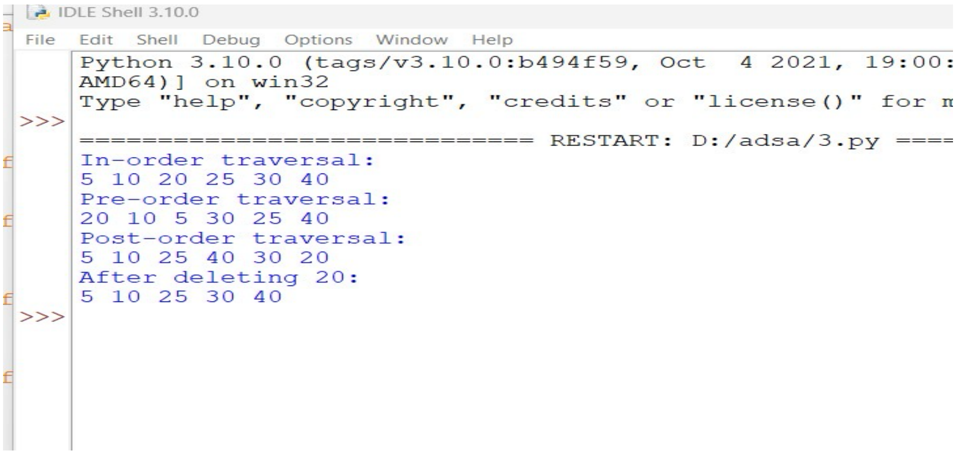
PRABHNOOR SINGH
102115059
3nc3
3(A)

```
class Node:
   def __init__(self, key):
      self.key = key
      self.height = 1
      self.left = None
      self.right = None

def height(node):
   return node.height if node else 0

def update_height(node):
   # Update the height of a node based on the maximum height of its children
   node.height = 1 + max(height(node.left), height(node.right))

def balance_factor(node):
   # Calculate the balance factor of a node
   return height(node.left) - height(node.right)

def right_rotate(y):
   # Right rotation to balance the tree
   x = y.left
   T2 = x.right

   x.right = y
   y.left = T2

   update_height(y)
   update_height(x)

   return x

def left_rotate(x):
   # Left rotation to balance the tree
   y = x.right
   T2 = y.left

   y.left = x
   x.right = T2

   update_height(x)
   update_height(y)

   return y

def insert(root, key):
    # Insert a new element
    if not root:
        return Node(key)

    if key < root.key:
        root.left = insert(root.left, key)
    elif key > root.key:
        root.right = insert(root.right, key)
    else:
        # Duplicates not allowed
        return root

    update_height(root)

    balance = balance_factor(root)

    if balance > 1:
        if key < root.left.key:
            return right_rotate(root)
        else:
            root.left = left_rotate(root.left)
            return right_rotate(root)

    if balance < -1:
        if key > root.right.key:
            return left_rotate(root)
        else:
            root.right = right_rotate(root.right)
            return left_rotate(root)

    return root

def delete(root, key):
    # Delete an existing element
    if not root:
        return root

    if key < root.key:
        root.left = delete(root.left, key)
    elif key > root.key:
        root.right = delete(root.right, key)
    else:
        if not root.left:
            return root.right
        elif not root.right:
            return root.left

        # Find the minimum node in the right subtree
        temp = find_min(root.right)
        root.key = temp.key
        root.right = delete(root.right, temp.key)
    update_height(root)

    balance = balance_factor(root)

    if balance > 1:
        if balance_factor(root.left) >= 0:
            return right_rotate(root)
        else:
            root.left = left_rotate(root.left)
            return right_rotate(root)

    if balance < -1:
        if balance_factor(root.right) <= 0:
            return left_rotate(root)
        else:
            root.right = right_rotate(root.right)
            return left_rotate(root)

    return root

def find_min(node):
    # Find the node with the minimum key value in a subtree
    while node.left:
        node = node.left
    return node

def inorder_traverse(root):
    # In-order traversal
    if root:
        inorder_traverse(root.left)
        print(root.key, end=" ")
        inorder_traverse(root.right)

def preorder_traverse(root):
    # Pre-order traversal
    if root:
        print(root.key, end=" ")
        preorder_traverse(root.left)
        preorder_traverse(root.right)
def postorder_traverse(root):
    # Post-order traversal of the AVL tree
    if root:
        postorder_traverse(root.left)
        postorder_traverse(root.right)
        print(root.key, end=" ")
root = None
keys = [30, 20, 40, 10, 25, 5]
for key in keys:
    root = insert(root, key)
```

```
print("In-order traversal:")
inorder_traverse(root)
print("\nPre-order traversal:")
preorder_traverse(root)
print("\nPost-order traversal:")
postorder_traverse(root)
root = delete(root, 20)
print("\nAfter deleting 20:")
inorder_traverse(root)
```



```
    Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for m
>>> 
    ============================= RESTART: D:/adsa/3.py ====
    In-order traversal:
    5 10 20 25 30 40
    Pre-order traversal:
    20 10 5 30 25 40
    Post-order traversal:
    5 10 25 40 30 20
    After deleting 20:
    5 10 25 30 40
>>> 
```