import random

class Node:

def \_\_init\_\_(self, key):

self.key = key

self.left = None

self.right = None

self.size = 1

def update\_size(node):

if node:

node.size = 1 + (node.left.size if node.left else 0) + (node.right.size if node.right else 0)

# Split a BST into two trees: left < key and right > key

def split(root, key):

if not root:

return (None, None)

if key < root.key:

left, right = split(root.left, key)

root.left = right

update\_size(root)

return (left, root)

else:

left, right = split(root.right, key)

root.right = left

update\_size(root)

return (root, right)

# Join two BSTs where all keys in left < keys in right

def join(left, right):

if not left or not right:

return left or right

if random.random() < left.size / (left.size + right.size):

left.right = join(left.right, right)

update\_size(left)

return left

else:

right.left = join(left, right.left)

update\_size(right)

return right

# Insert a key while preserving randomness

def insert(root, key):

if not root:

return Node(key)

if random.randint(0, root.size) == root.size:

new\_node = Node(key)

left, right = split(root, key)

new\_node.left = left

new\_node.right = right

update\_size(new\_node)

return new\_node

if key < root.key:

root.left = insert(root.left, key)

else:

root.right = insert(root.right, key)

update\_size(root)

return root

# Delete a key while preserving randomness

def delete(root, key):

if not root:

return None

if key < root.key:

root.left = delete(root.left, key)

elif key > root.key:

root.right = delete(root.right, key)

else:

root = join(root.left, root.right)

update\_size(root)

return root

# Inorder traversal for testing

def inorder(node):

if not node: return []

return inorder(node.left) + [node.key] + inorder(node.right)

output –

# DemoExample

tree = None

for k in [5, 3, 8, 1, 4, 7, 9]:

tree = insert(tree, k)

print("Inorder after insertion:", inorder(tree))

tree = delete(tree, 5)

print("Inorder after deletion of 5:", inorder(tree))

example output –

Inorder after insertion: [1, 3, 4, 5, 7, 8, 9]

Inorder after deletion of 5: [1, 3, 4, 7, 8, 9]