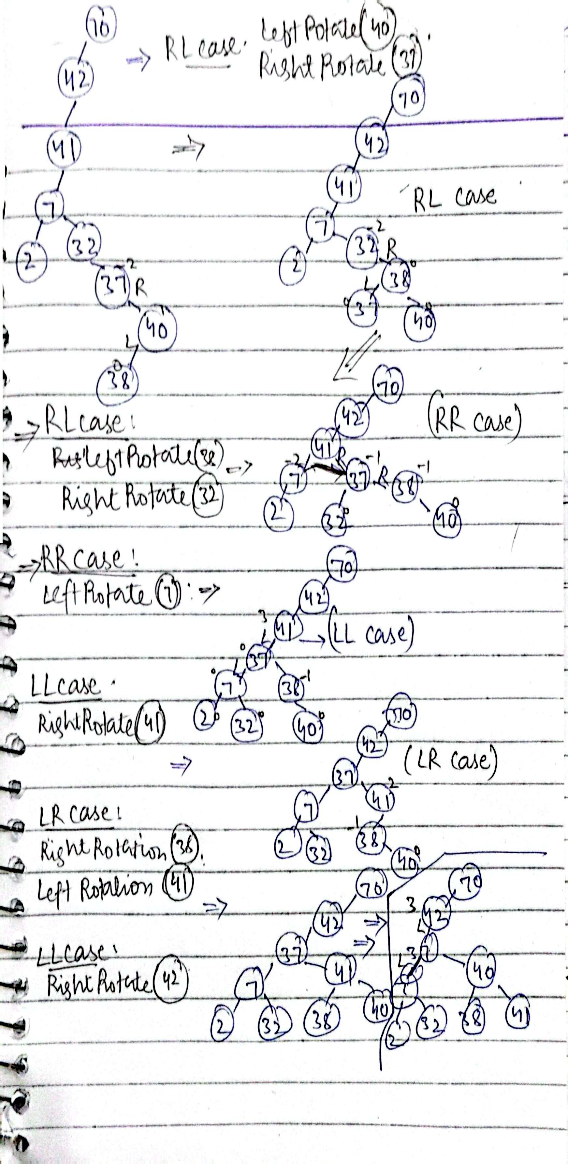
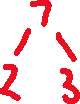
**Waleed Afzal  
23P-0566**

**3D  
DSA LAB 13**

**A notebook with lines and a drawing on it

Description automatically generatedQues1(a):**

**Ques2:**



A black and white text with black lines

Description automatically generated with medium confidence



**Ques3:**

**Ques4:**

bool isSubtreeAVL(Node\* root, int minVal, int maxVal) {

if (root == nullptr)

return true;

if (root->key < minVal || root->key > maxVal)

return false;

return isSubtreeAVL(root->left, minVal, maxVal) && isSubtreeAVL(root->right, minVal, maxVal);

}

bool checkAVLSubtree(Node\* root, int minVal, int maxVal) {

if (!isSubtreeAVL(root, minVal, maxVal))

return false;

return isAVL(root);

}

**Q5:**

#include <iostream>

using namespace std;

struct Node {

int key;

Node\* left;

Node\* right;

int height;

};

int height(Node\* N) {

if (N == nullptr) {

return 0;

}

return N->height;

}

int max(int a, int b) {

if (a > b) {

return a;

}

return b;

}

Node\* rightRotate(Node\* y) {

Node\* x = y->left;

Node\* T2 = x->right;

x->right = y;

y->left = T2;

y->height = max(height(y->left), height(y->right)) + 1;

x->height = max(height(x->left), height(x->right)) + 1;

return x;

}

Node\* leftRotate(Node\* x) {

Node\* y = x->right;

Node\* T2 = y->left;

y->left = x;

x->right = T2;

x->height = max(height(x->left), height(x->right)) + 1;

y->height = max(height(y->left), height(y->right)) + 1;

return y;

}

int getBalance(Node\* N) {

if (N == nullptr) {

return 0;

}

return height(N->left) - height(N->right);

}

Node\* insert(Node\* node, int key) {

if (node == nullptr) {

Node\* newNode = new Node();

newNode->key = key;

newNode->left = newNode->right = nullptr;

newNode->height = 1;

return newNode;

}

if (key < node->key) {

node->left = insert(node->left, key);

} else if (key > node->key) {

node->right = insert(node->right, key);

} else {

return node;

}

node->height = 1 + max(height(node->left), height(node->right));

int balance = getBalance(node);

if (balance > 1 && key < node->left->key) {

return rightRotate(node);

}

if (balance < -1 && key > node->right->key) {

return leftRotate(node);

}

if (balance > 1 && key > node->left->key) {

node->left = leftRotate(node->left);

return rightRotate(node);

}

if (balance < -1 && key < node->right->key) {

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

void checkKthLargest(Node\* root, int& k, int& result) {

if (root == nullptr || k <= 0) return;

checkKthLargest(root->right, k, result);

k--;

if (k == 0) {

result = root->key;

return;

}

checkKthLargest(root->left, k, result);

}

Node\* deleteNode(Node\* root, int key) {

if (root == nullptr) return root;

if (key < root->key) {

root->left = deleteNode(root->left, key);

} else if (key > root->key) {

root->right = deleteNode(root->right, key);

} else {

if ((root->left == nullptr) || (root->right == nullptr)) {

Node\* temp = root->left ? root->left : root->right;

if (temp == nullptr) {

temp = root;

root = nullptr;

} else {

\*root = \*temp;

}

delete temp;

} else {

Node\* temp = root->right;

while (temp && temp->left != nullptr) {

temp = temp->left;

}

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

}

if (root == nullptr) {

return root;

}

root->height = 1 + max(height(root->left), height(root->right));

int balance = getBalance(root);

if (balance > 1 && getBalance(root->left) >= 0) {

return rightRotate(root);

}

if (balance > 1 && getBalance(root->left) < 0) {

root->left = leftRotate(root->left);

return rightRotate(root);

}

if (balance < -1 && getBalance(root->right) <= 0) {

return leftRotate(root);

}

if (balance < -1 && getBalance(root->right) > 0) {

root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

void inorder(Node\* root) {

if (root != nullptr) {

inorder(root->left);

cout << root->key << " ";

inorder(root->right);

}

}

int main() {

Node\* root = nullptr;

int n, value, k;

cout << "Enter the number of elements to insert into the AVL tree: ";

cin >> n;

cout << "Enter " << n << " integers: ";

for (int i = 0; i < n; i++) {

cin >> value;

root = insert(root, value);

}

cout << "Inorder traversal of the constructed AVL tree: ";

inorder(root);

cout << endl;

cout << "Enter the value of k to delete the kth largest element: ";

cin >> k;

int kthLargest = 0;

checkKthLargest(root, k, kthLargest);

cout << "The " << k << "th largest element is: " << kthLargest << endl;

root = deleteNode(root, kthLargest);

cout << "Inorder traversal after deleting the " << k << "th largest element: ";

inorder(root);

cout << endl;

return 0;

}