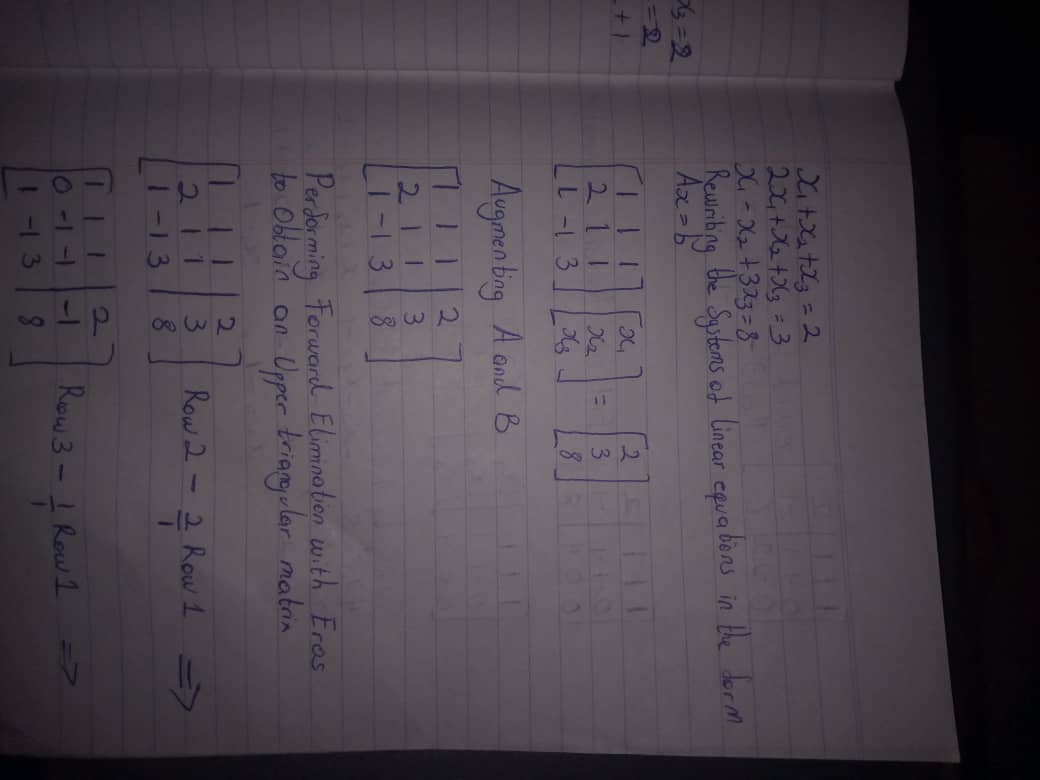
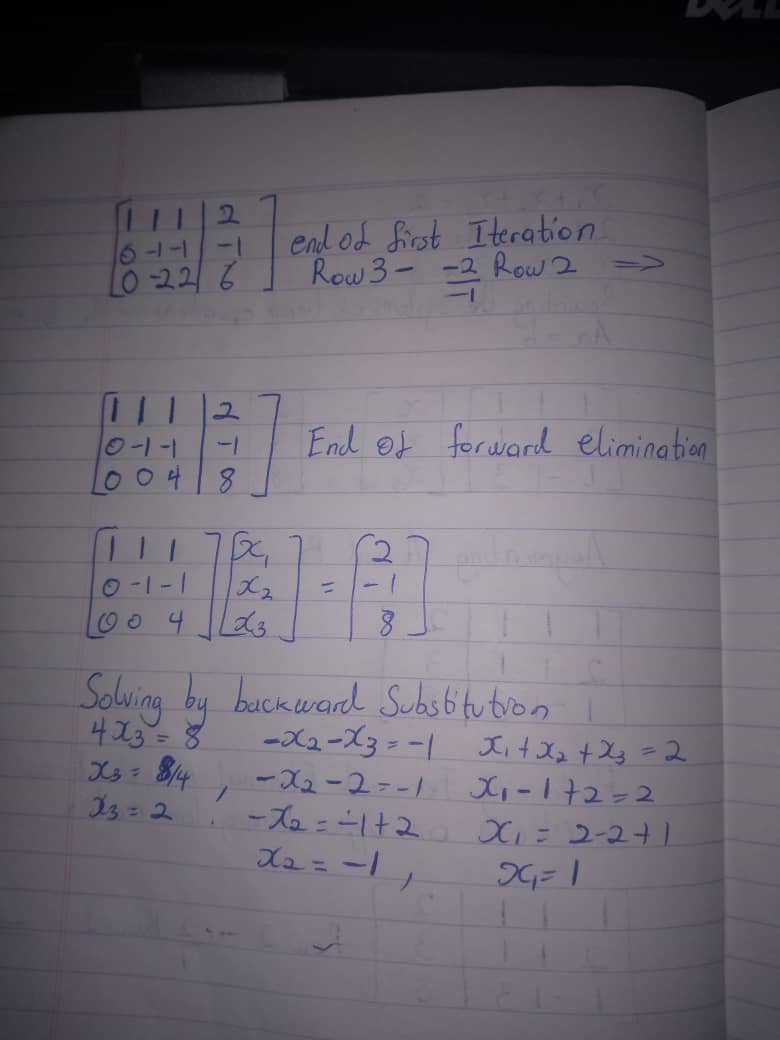
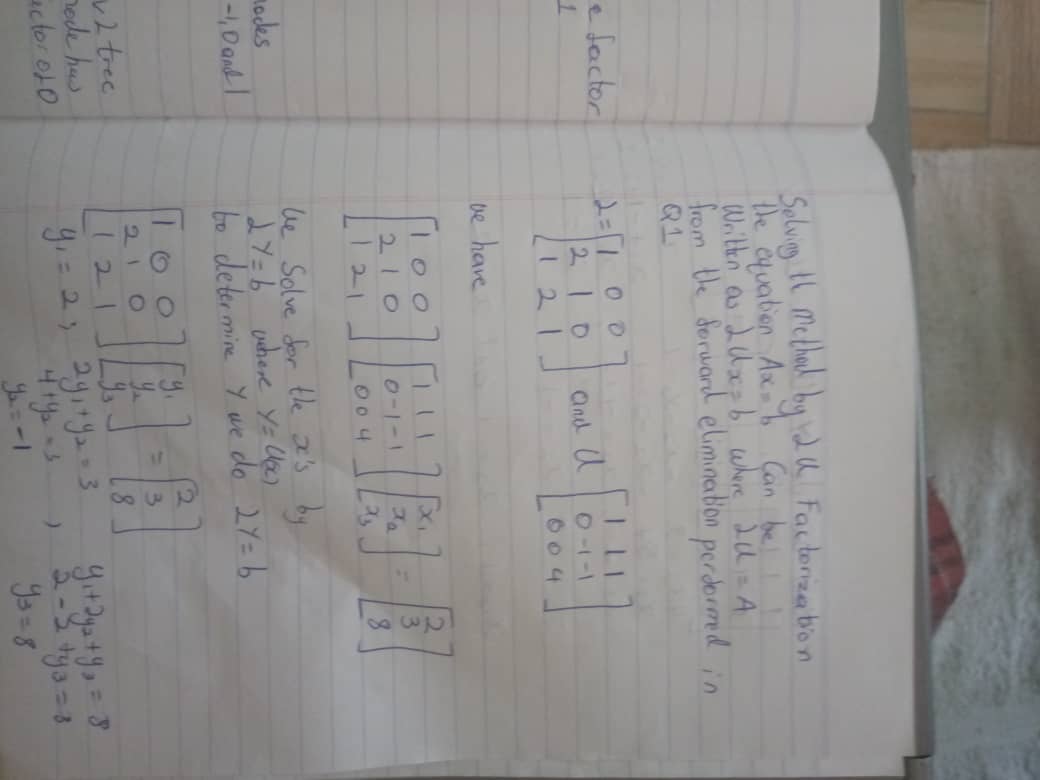
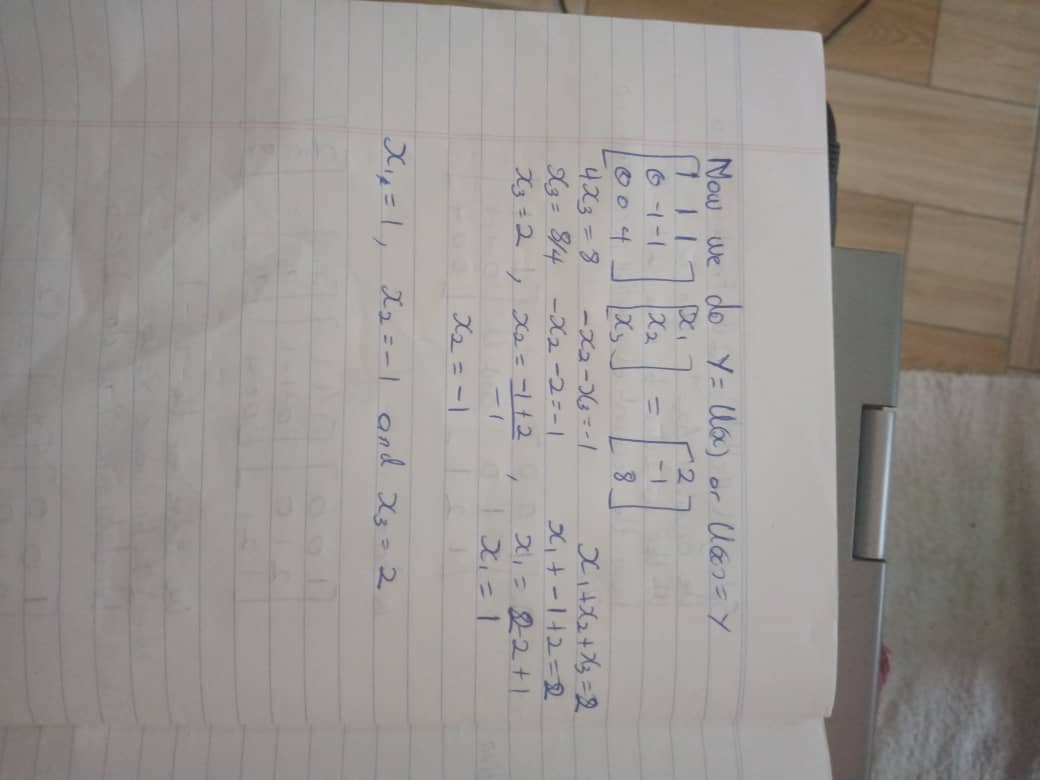
**EXERCISE 6.2:**

**Q1.**



**Q2.a**

****

****

**Q5. Gaussian Backward Substitution Pseudocode**

// Finds the solution of a n unknowns using back substitution on an upper triangular

// matrix and column vector of results

// Input: Matrix A [1…n, 1…n+1] with the first n column in the upper triangular form

//Output: Solution of the n linear system of equations in n unknowns

for i n to 1 do

initial = 0

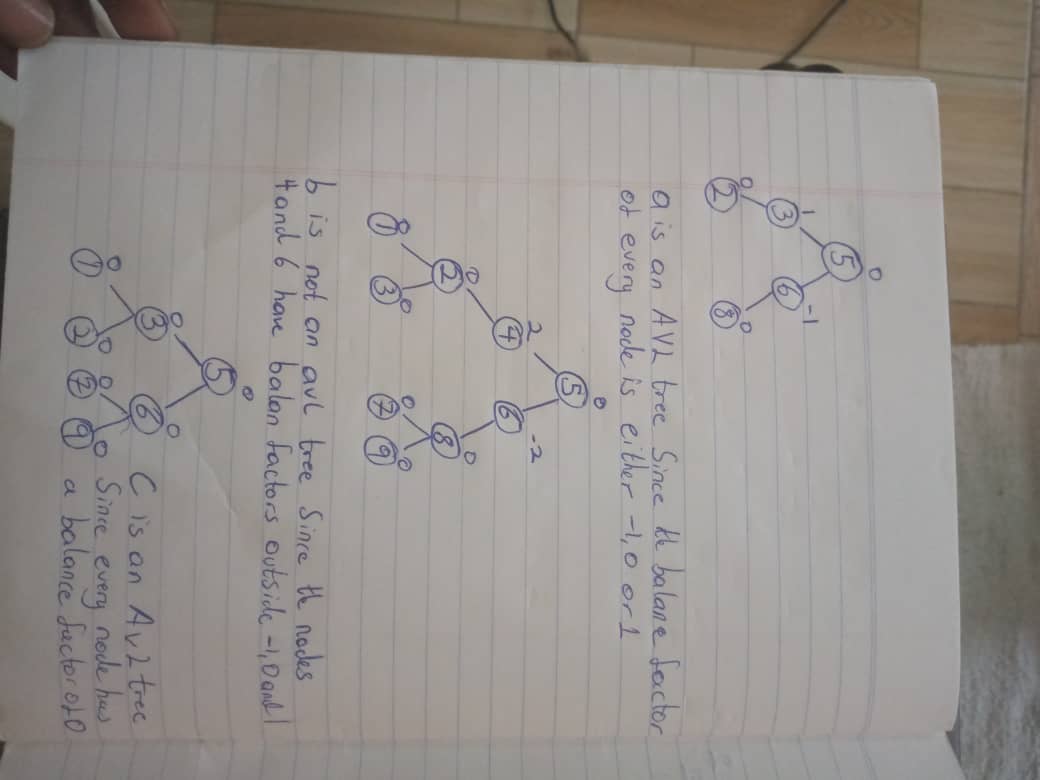
for j n to i +1 do

initial initial + A [i, j] \*x[j]

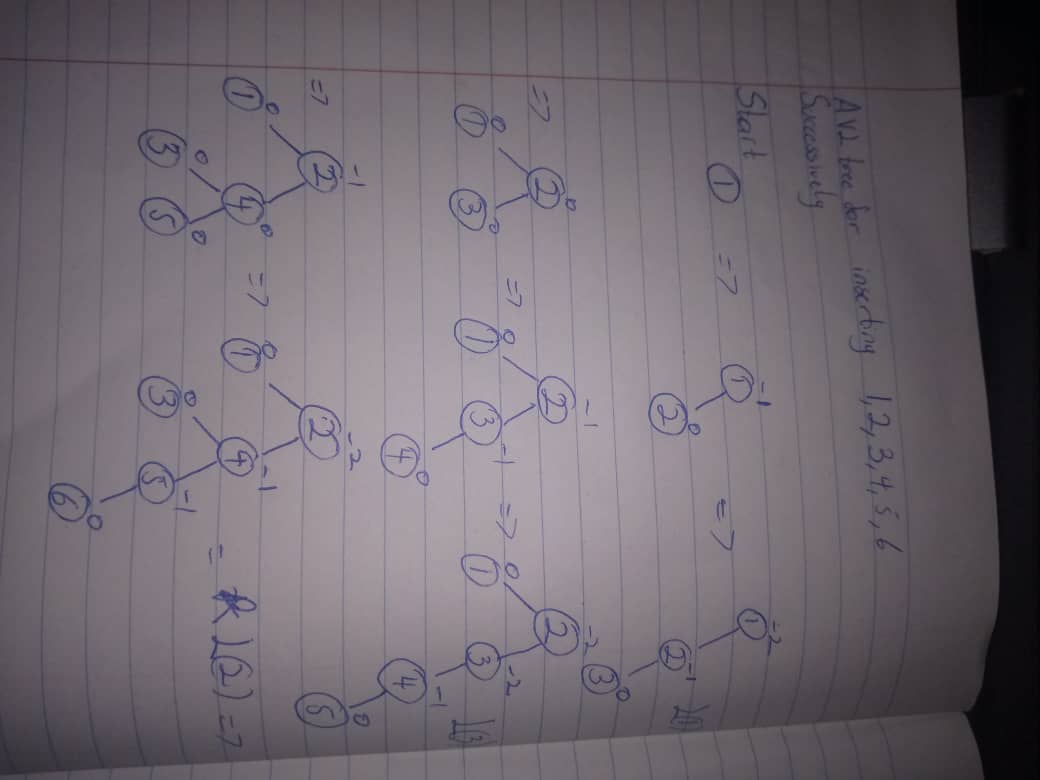
x[i] (A[i, n+1] – initial) / A[i,i]

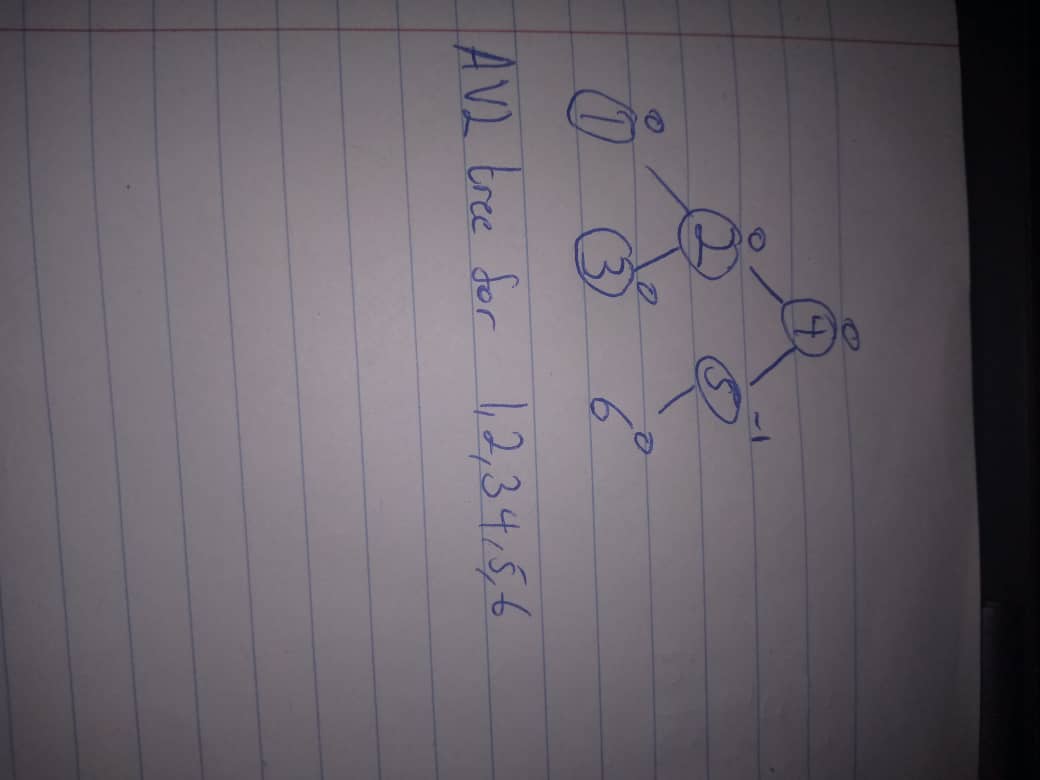
return x

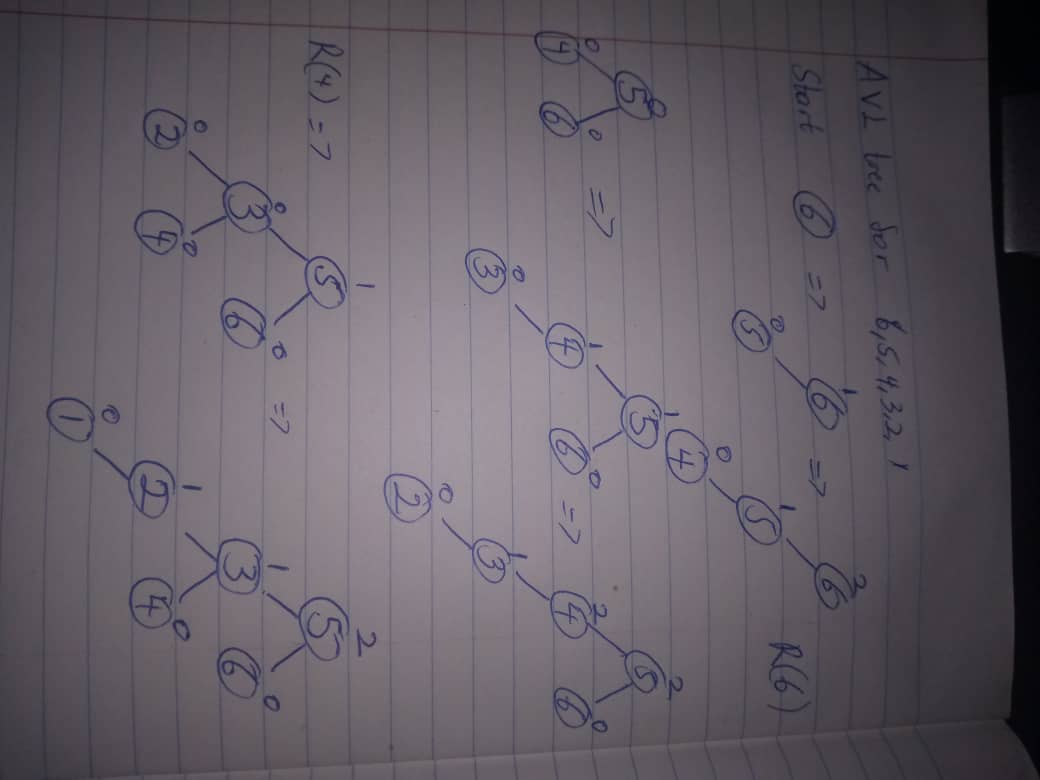
**EXERCISE 6.3:**

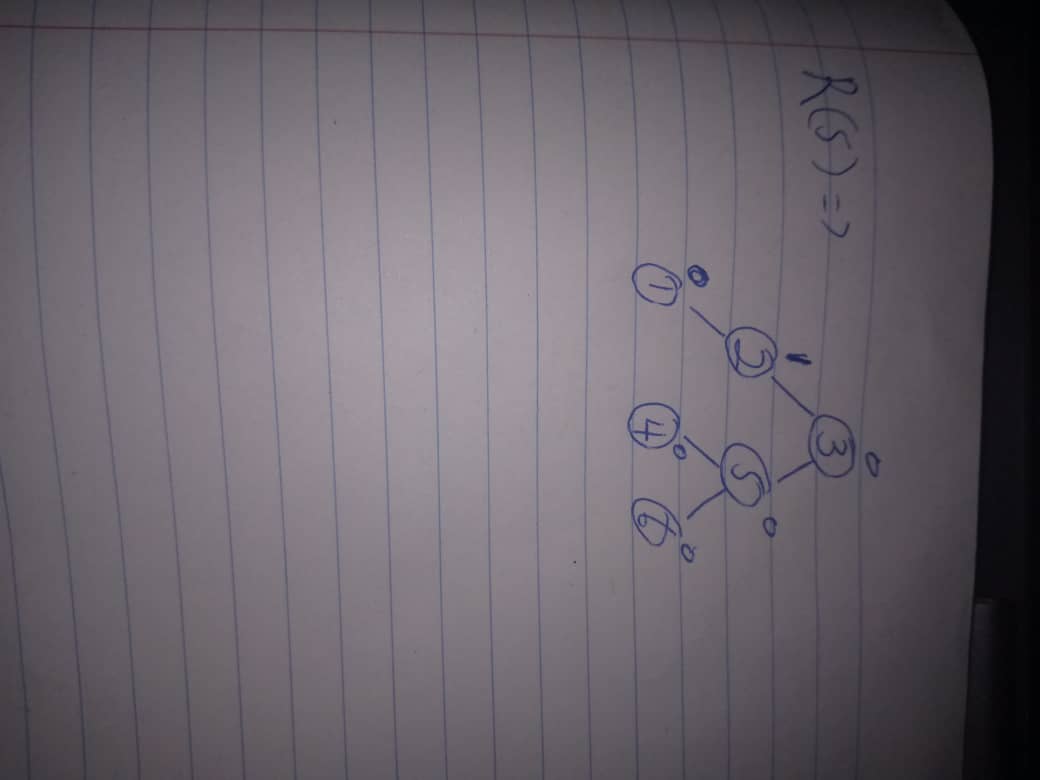
**Q1.**

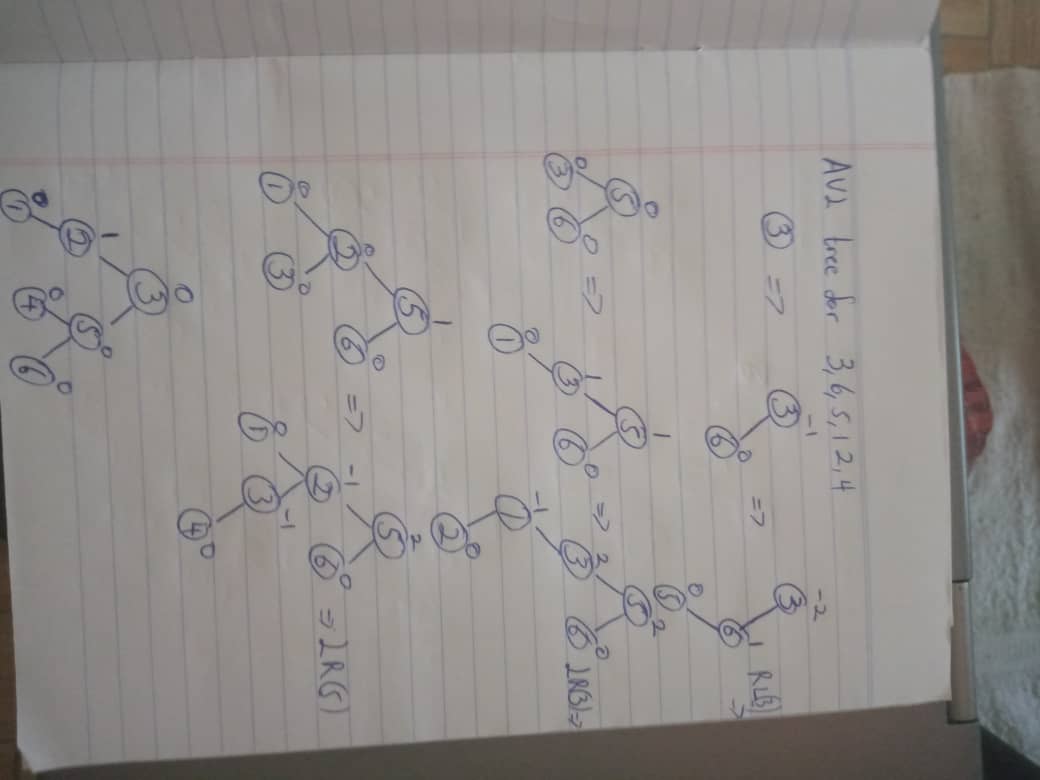
**Q4.**

****

****

****

****

****

**Q6. Construct a program for creating an avl tree with n distinct elements**

**#include<iostream>**

**#include<cstdio>**

**#include<sstream>**

**#include<algorithm>**

**#define pow2(n) (1 << (n))**

**using namespace std;**

**struct Node {**

**int key;**

**struct Node \*left;**

**struct Node \*r;**

**}\*r;**

**class avlTree {**

**public:**

**int height(Node \*);**

**int difference(Node \*);**

**Node \*rightr\_rotate(Node \*);**

**Node \*leftl\_rotate(Node \*);**

**Node \*leftright\_rotate(Node\*);**

**Node \*rightleft\_rotate(Node \*);**

**Node \* balance(Node \*);**

**Node \* insert(Node\*, int);**

**void show(Node\*, int);**

**avlTree() {**

**r = NULL;**

**}**

**};**

**int avlTree::height(Node \*t) {**

**int h = 0;**

**if (t != NULL) {**

**int leftHeight = height(t->left);**

**int rightHeight = height(t->r);**

**int maxHeight = max(leftHeight, rightHeight);**

**h = maxHeight + 1;**

**}**

**return h;**

**}**

**int avlTree::difference(Node \*t) {**

**int leftHeight = height(t->left);**

**int rightHeight = height(t->r);**

**int b\_factor = leftHeight - rightHeight;**

**return b\_factor;**

**}**

**Node \*avlTree::rightr\_rotate(Node \*parent) {**

**Node \*t;**

**t = parent->r;**

**parent->r = t->left;**

**t->left = parent;**

**return t;**

**}**

**Node \*avlTree::leftl\_rotate(Node \*parent) {**

**Node \*t;**

**t = parent->left;**

**parent->left = t->r;**

**t->r = parent;**

**return t;**

**}**

**Node \*avlTree::leftright\_rotate(Node \*parent) {**

**Node \*t;**

**t = parent->left;**

**parent->left = rightr\_rotate(t);**

**return leftl\_rotate(parent);**

**}**

**Node \*avlTree::rightleft\_rotate(Node \*parent) {**

**Node \*t;**

**t = parent->r;**

**parent->r = leftl\_rotate(t);**

**return rightr\_rotate(parent);**

**}**

**Node \*avlTree::balance(Node \*t) {**

**int balFactor = difference(t);**

**if (balFactor > 1) {**

**if (difference(t->left) > 0)**

**t = leftl\_rotate(t);**

**else**

**t = leftright\_rotate(t);**

**} else if (balFactor < -1) {**

**if (difference(t->r) > 0)**

**t = rightleft\_rotate(t);**

**else**

**t = rightr\_rotate(t);**

**}**

**return t;**

**}**

**Node \*avlTree::insert(Node \*r, int v) {**

**if (r == NULL) {**

**r = new Node;**

**r->key = v;**

**r->left = NULL;**

**r->r = NULL;**

**return r;**

**} else if (v< r->key) {**

**r->left = insert(r->left, v);**

**r = balance(r);**

**} else if (v >= r->key) {**

**r->r = insert(r->r, v);**

**r = balance(r);**

**} return r;**

**}**

**void avlTree::show(Node \*p, int left) {**

**int i;**

**if (p != NULL) {**

**show(p->r, left+ 1);**

**cout<<" ";**

**if (p == r)**

**cout << "Root -> ";**

**for (i = 0; i < left&& p != r; i++)**

**cout << " ";**

**cout << p->key;**

**show(p->left, left + 1);**

**}**

**}**

**int main() {**

**int size;**

**avlTree object;**

**cout<<"Enter the total element of the tree"<<endl;**

**cin>>size;**

**int value;**

**cout<<"------------------------"<<endl;**

**for (int i = 0; i < size; i++) {**

**cout<<"Enter value for location "<<i+1<<" and press ENTER to continue"<<endl;**

**cin>>value;**

**r = object.insert(r,value);**

**object.show(r,1);**

**cout<<endl;**

**}**

**return 0;**

**}**