**LAB 09:**

**Question 01:**

public class Main{

static class node{

double d;

node l,r;

node(double d){

this.d=d;

}

}

public static node insert(node root,double val){

if (root==null){

root=new node(val);

return root;

}

if(root.d>val){

root.l=insert(root.l,val);

}

else{

root.r=insert(root.r,val);

}

return root;

}

public static void inorder(node root){

if (root==null){

return ;

}

inorder(root.l);

System.out.print(root.d+" , ");

inorder(root.r);

}

public static node delete(node root,double val){

if(root.d>val){

root.l=delete(root.l,val);

}

else if(root.d<val){

root.r=delete(root.r,val);

}

else{

if(root.l==null&&root.r==null){

return null;

}

if(root.l==null){

return root.r;

}

else if(root.r==null){

return root.l;

}

node IS=inordersuccessor(root.r);

root.d=IS.d;

root.r= delete(root.r, IS.d);

}

return root;

}

public static node inordersuccessor(node root){

while (root.l!=null){

root=root.l;

}

return root;

}

public static void main(String[] args) {

double values[]={5,1,3,3.5,4,4.5,2,7,8,9,10,11,12,13,14};

node root=null;

for (int i=0;i< values.length;i++){

root=insert(root,values[i]);

}

System.out.println("Insertion :");

inorder(root);

System.out.println("\n");

System.out.println("Deletion of 3 :");

delete(root,3);

inorder(root);

System.out.println("\n");

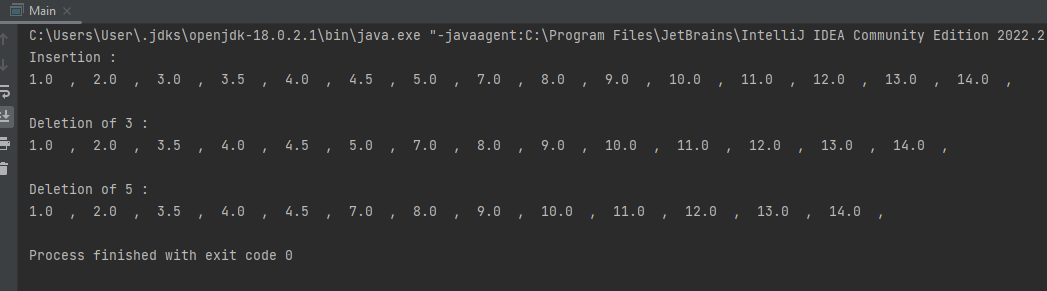
System.out.println("Deletion of 5 :");

delete(root,5);

inorder(root);

System.out.println();

} }



**Question 02:**

public class Main{

static class node{

node l,r;

int d;

node(int d){

this.d=d;

this.l=null;

this.r=null;

}

}

public node Insert(node root, int val){

if(root==null){

root=new node(val);

return root; }

else if(val<root.d){

root.l=Insert(root.l,val); }

else if(val>root.d){

root.r=Insert(root.r,val); }

return root;

}

public boolean bst\_check(node root){

if(root==null){

return true; }

if(root.l!=null&&root.l.d>root.d){

return false; }

if(root.r!=null&&root.r.d<root.d){

return false; }

return true;

}

public void Inorder(node root){

if(root==null){

return; }

Inorder(root.l);

System.out.print(root.d+" ");

Inorder(root.r);

}

public static void main(String[] args) {

Main m = new Main();

node root=new node(10);

m.Insert(root,10);

m.Insert(root,5);

m.Insert(root,15);

m.Insert(root,3);

m.Insert(root,7);

m.Insert(root,18);

m.Inorder(root);

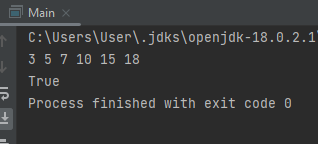
if (m.bst\_check(root)){

System.out.print("\nTrue"); }

else {

System.out.print("\nFalse"); }

} }



**Question 03:**

import java.util.Scanner;

public class q3\_21k3868 {

class node {

node left;

node right;

int data;

public node(int data) {

this.left = null;

this.right = null;

this.data = data;

}

}

node insertBinaryTree(node root) {

Scanner sc = new Scanner(System.in);

int data = sc.nextInt();

if (data == -1) {

return null;

}

root = new node(data);

System.out.print("Enter the left data of " + data + " :");

root.left = insertBinaryTree(root.left);

System.out.print("Enter the right data of " + data + " :");

root.right = insertBinaryTree(root.right);

return root;

}

node check(node root) {

if (root == null) {

return null;

}

if (root.data % 5 == 0) {

System.out.print(root.data + "\t");

}

root.left = check(root.left);

root.right = check(root.right);

return root;

}

public static void main(String[] args) {

node root = null;

lab09.q3\_21k3868 q = new lab09.q3\_21k3868();

System.out.println("Enter root data");

root = q.insertBinaryTree(root);

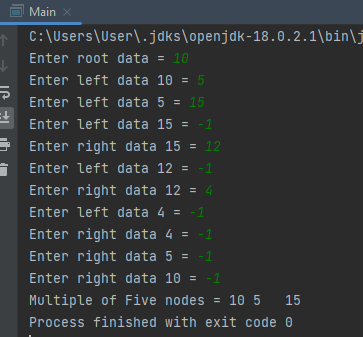
System.out.println("\n------------------------------------");

System.out.println("Nodes Having the multiple of Five : ");

root = q.check(root);

}

}



**Question 04:**

class Main{

class node{

node l,r;

float key;

public node(float d) {

key=d;

this.l=null;

this.r=null;

}

}

node root;

void BinarySearch(){

root=null;

}

void BinarySearch(float val){

root=new node(val);

}

void insert(float key){

root=insertRec(root,key);

}

node insertRec(node root,float key){

if(root==null){

root=new node(key);

return root; }

else if(key<root.key)

root.l=insertRec(root.l,key);

else if(key>root.key)

root.r=insertRec(root.r,key);

return root;

}

void insert(int key){

root = insertRec(root, key);

}

node insertRec(node root,int key){

if(root==null){

root=new node(key);

return root; }

if(key<root.key)

root.l=insertRec(root.l,key);

else if(key>root.key)

root.r=insertRec(root.r, key);

return root;

}

void inorder(){

Inorder(root);

}

void Inorder(node root){

if(root!=null){

Inorder(root.l);

System.out.println(root.key);

Inorder(root.r);

}

}

static int c=0;

public void count(node root,int d){

if(root==null){

return;

}

if(d>=root.key){

c++;

}

count(root.l,d);

count(root.r,d);

}

public static void main(String[] args) {

Main m=new Main();

m.insert(7);

m.insert(3);

m.insert(11);

m.insert(1);

m.insert(5);

m.insert(4);

m.insert(6);

m.insert(8);

m.insert(9);

m.insert(13);

m.insert(12);

m.insert(14);

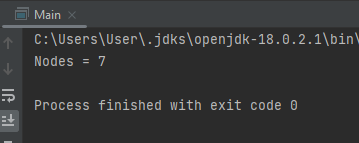
m.insert(3.5f);

m.insert(4.5f);

m.count(m.root,6);

System.out.println("Nodes = "+c);

} }



**Question 05:**

import java.util.\*;

public class Main {

static int array[] = new int[20];

public int factorial(int num) {

int fact = 1;

if(num == 0)

return 1;

else {

while(num > 1) {

fact = fact \* num;

num--;

}

return fact;

}

}

public int numOfBST(int key) {

int ank = factorial(2 \* key)/(factorial(key + 1) \* factorial(key));

return ank;

}

public static void main(String[] args) {

Main m = new Main();

Scanner s = new Scanner (System.in);

System.out.println("Enter a unique Number to find how many maximum numbers can be formed");

int number = s.nextInt();

m.numOfBST(number);

System.out.println("Maimum numbers that can form: "+m.numOfBST(number));

}

}

