```
//HW4 Due: 11:59pm, Wednesday, Oct. 25
//Implement initializer_list as described below
#include <iostream>
#include <typeinfo>
#include <math.h> //log2
using namespace std;

class node {
public:
        int value;
        node* Lchild;
        node* Rchild;
        node(int i) : value(i), Lchild{ nullptr }, Rchild{ nullptr } {}
        node() { Lchild = Rchild = nullptr; }
};
class tree {//Full binary trees: 1, 3, 7, 15 ,   2^k-1 nodes
public:
        node* root;
        tree() { root = nullptr; }
        tree(int n, int m);//constructor
        tree(const initializer_list<int>& I);
        void helper(const initializer_list<int>& I, const int*& pI, node* p);
        //You can assume that constructor is avaialble to use.
        node* Const_help(int n, int m);
        node* MakeTree(int n, int m);//Make a full binary tree with n level and random
values in 0 ...m-1
        void InOrderT(node* p);//Inorder Traveral of a subtree whose root is pointed by p
        void PreOrderT(node* p);
        void PostOrderT(node* p);

        void maxHeap(node* p);//rearrange the subtree whose root pointed by p into a maxHeap
        pair<int, int> MaxMin(node* p);

};
pair<int, int> tree::MaxMin(node* p) {
        if (!p) return { -1, -1 };
        if (!p->Lchild) return { p->value, p->value };
        pair<int, int> P1, P2;
        P1 = MaxMin(p->Lchild);
        P2 = MaxMin(p->Rchild);
        int Max{ max(P1.first, P2.first) }, Min{ min(P1.second, P2.second) };
        if (P1.first - P1.second < P2.first - P2.second) swap(p->Lchild, p->Rchild);
        return { max(Max, p->value), min(Min, p->value) };
}




/*
Initializer_list will create a T such that when printed using InorderT, the output will have
the same sequence as
in the initializer_list.

MaxMin will return the maximum and minimum values in the tree as a pair of int.
In addition, at each node, restructure the tree, if needed, such that
the difference between max and min on the left child branch must be
greater than or equal to that of right child branch.  Recursion implementation is required.
To return a pair of i1 and i2:    return {i1, i2};
If (p== nullptr) return {-1, -1};
You can use functions max (a,b) and min(a, b).  max returns the larger of the two, and min
returns the smaller of the two.  Note that max and min only work on TWO numbers. */


node* tree::Const_help(int n, int m) {
```

```
        if (n == 0) return nullptr;//NULL is nullptr and zero; don't use it anymore
        node* p{ new node{ rand() % m } };
        p->Lchild = Const_help(n - 1, m);
        p->Rchild = Const_help(n - 1, m);
        return  p;
}


tree::tree(int n, int m) {

        root = Const_help(n, m);

}



void tree::PostOrderT(node* p) {
        if (!p) return;
        PostOrderT(p->Lchild);
        //cout << p->value << " ";
        PostOrderT(p->Rchild);
        cout << p->value << " ";
}

void tree::PreOrderT(node* p) {
        if (!p) return;
        cout << p->value << " ";
        PreOrderT(p->Lchild);
        //cout << p->value << " ";
        PreOrderT(p->Rchild);
}

void tree::InOrderT(node* p) {
        if (!p) return;
        InOrderT(p->Lchild);
        cout << p->value << " ";
        InOrderT(p->Rchild);
}

node* tree::MakeTree(int n, int m) {
        if (n == 0) return nullptr;
        node* p{ new node{rand() % m} };
        p->Lchild = MakeTree(n - 1, m);
        p->Rchild = MakeTree(n - 1, m);
        return p;
}


int main() {

        tree T20{ 0,1,2,3,4,5,6 };
        T20.InOrderT(T20.root);
        cout << endl;//0 1 2 3 4 5 6







        return 0;
}
```