#include<iostream>

#include<cmath>

using namespace std;

int static ceng = 0;//用于记录树打印到第几层了

bool static key = true;

//计算2的n次方(n>=0)

int power2(int x) {

int n = 1;

for (int i = 1; i <= x; i++) {

n \*= 2;

}

return n;

}

class Node {

public:

char c;

int data;

Node\* left;

Node\* right;

Node\* next;

Node() {

c = '#';

left = NULL;

right = NULL;

next = NULL;

}

Node(int x) {

data = x;

c = '#';

left = NULL;

right = NULL;

next = NULL;

}

};

class ListQueue {

public:

Node\* front1;

Node\* front2;//一直指向队首

Node\* rear;

ListQueue() {

front1 = front2 = rear = NULL;

}

//进队列

void push(Node\* tmp) {

if (front1 == NULL) {

front1 = front2 = tmp;

rear = tmp;

}

else {

rear->next = tmp;

tmp->next = NULL;

rear = tmp;

}

}

//出队列

void pop() {

if (front1 == NULL)return;

front1 = front1->next;

}

};

class tree {

public:

ListQueue Q;

Node\* root;

tree() {

root = NULL;

}

//看看这个节点的左右指针是否为空

Node\*& look(Node\* n) {

if (n->left == NULL)return n->left;

else {

if (n->right == NULL) return n->right;

else look(n->next);

}

}

//创建一个空的二叉树

void build() {

Node\* tmp = new Node();

if (root == NULL) {

root = tmp;

Q.push(tmp);

}

else {

look(Q.front1) = tmp;

Q.push(tmp);

if (Q.front1->right != NULL) {

Q.pop();

}

}

}

//插入函数

void Insert(Node\* root, int x) {

if (this->root->c == '#') {

this->root->c = '+';

this->root->data = x;

return;

}

if ((root->left->c == '#') && x < root->data) {

root->left->data = x;

root->left->c = '+';

return;

}

if ((root->left->c != '#') && x < root->data) {

Insert(root->left, x);

return;

}

if ((root->right->c == '#') && x > root->data) {

root->right->data = x;

root->right->c = '+';

return;

}

if ((root->right->c != '#') && x > root->data) {

Insert(root->right, x);

return;

}

}

//返回节点高度

int height(Node\* root) {

if (root == NULL || root->c == '#')return 0;

if (root != NULL) {

int a = height(root->left);

int b = height(root->right);

return 1 + ((a > b) ? a : b);

}

}

//判断是否为AVL

bool check(Node\* root) {

if (root == NULL) {

ceng = 0;

if (key) {

return true;

}

else {

key = true;

return false;

}

}

bool T = true;

for (int i = 1; i <= power2(ceng); i++) {

if (root->c == '#') {}

else {

T = false;

if (root->left->c == '#' && root->right->c == '#') {}

if (root->left->c == '#' && root->right->c != '#') {

if (height(root->right) >= 2) key = false;

}

if (root->left->c != '#' && root->right->c == '#') {

if (height(root->left) >= 2)key = false;

}

if (root->left->c != '#' && root->right->c != '#') {

if (abs(height(root->left) - height(root->right)) >= 2) {

key = false;

}

}

}

if (i == power2(ceng) && T) {

ceng = 0;

if (key) {

return true;

}

else {

key = true;

return false;

}

}

root = root->next;

}

ceng += 1;

check(root);

}

//按升序序列输出（中序遍历）

int\* sortprint(Node\* root, int n) {

//栈

int top1 = -1;

Node\*\* H = new Node \* [n];

int top2 = -1;

int\* p = new int[n];

//当栈为空且所指根节点为空，说明该二叉树遍历完成

while (root->c != '#' || top1 != -1) {

while (root->c != '#') {

H[++top1] = root;

root = root->left;

}

//退出循环，说明左子树遍历完毕，开始遍历右子树

if (top1 != -1) {

root = H[top1--];

cout << root->data << " ";

p[++top2] = root->data;

root = root->right;

}

}

return p;

}

//打印树

void print(Node\* root) {

if (root == NULL) {

ceng = 0;

return;

}

bool T = true;

for (int i = 1; i <= power2(ceng); i++) {

if (root->c == '#') {}

else {

T = false;

if (root->left->c == '#' && root->right->c == '#') {

cout << root->data << "(#,#) ";

}

if (root->left->c == '#' && root->right->c != '#') {

cout << root->data << "(#," << root->right->data << ") ";

}

if (root->left->c != '#' && root->right->c == '#') {

cout << root->data << "(" << root->left->data << ",#) ";

}

if (root->left->c != '#' && root->right->c != '#') {

cout << root->data << "(" << root->left->data << ", " << root->right->data << ") ";

}

}

if (i == power2(ceng) && T) {

ceng = 0;

return;

}

root = root->next;

}

cout << endl;

ceng += 1;

print(root);

}

};

//构造平衡二叉搜索树（二分的方法）

void AVL(tree t,int a[],int m,int n) {

int start = m;

int end = n;

if (start == end) {

t.Insert(t.root, a[(start + end) / 2]);

return;

}

else {

t.Insert(t.root, a[(start + end) / 2]);

if ((start + end) / 2 - 1 < start) {}

else {

AVL(t, a, start, (start + end) / 2 - 1);

}

if ((start + end) / 2 + 1>end) {

}

else {

AVL(t, a, (start + end) / 2 + 1, end);

}

}

}

int main() {

tree t;

int num;

int x;//关键字

int n;//数字数量

cout << "请输入数量：" << endl;

cin >> n;

num = power2(n + 1) - 1;//要构建的节点数量

for (int i = 1; i <= num; i++) {

t.build();

}

cout << "请输入序列" << endl;

for (int i = 1; i <= n; i++) {

cin >> x;

t.Insert(t.root, x);

}

cout << "该二叉搜索树为：" << endl;

t.print(t.root);

cout << endl;

cout << "升序输出元素：" << endl;

int \*p=t.sortprint(t.root, n);//升序数组

cout << endl;

cout << endl;

cout << "该二叉搜索树是否为AVL？" << endl;

if (t.check(t.root)) {

cout << "是" << endl;

}

else {

cout << "不是"<<endl;

cout << "下面将其转化为AVL：" << endl;

tree tt;

for (int i = 1; i <= num; i++) {

tt.build();

}

AVL(tt, p, 0, n - 1);

tt.print(tt.root);

}

system("pause");

return 0;

}