题目描述：

给定长度为 n 的无序的数字数组，每个数字代表二叉树的叶子节点的权值，数字数组的值均大于等于 1。请完成一个函数，根据输入的数字数组，生成哈夫曼树，并将哈夫曼树按照中序遍历输出。

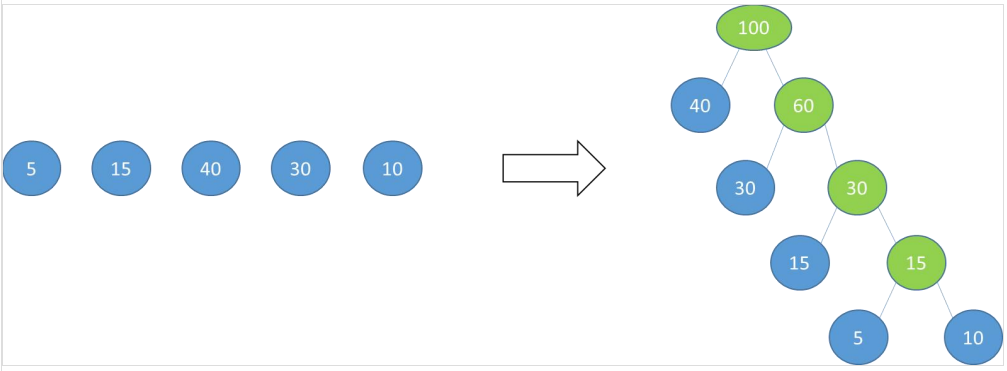
为了保证输出的二叉树中序遍历结果统一，增加以下限制：二叉树节点中，左节点权值小于等于右节点权值，根节点权值为左右节点权值之和。当左右节点权值相同时，左子树高度高度小于等于右子树。

注意：所有用例保证有效，并能生成哈夫曼树。

提醒：哈夫曼树又称最优二叉树，是一种带权路径长度最短的二叉树。所谓树的带权路径长度，就是树中所有的叶结点的权值乘上其到根结点的路径长度（若根结点为 0 层，叶结点到根结点的路径长度为叶结点的层数）。

例如：

由叶子节点 5 15 40 30 10 生成的最优二叉树如下图所示，该树的最短带权路径长度为 40*1+30*2+15*3+5*4+10*4=205。



输入描述：

第一行输入为数组长度，记为 N，1<=N<=1000，第二行输入无序数值数组，以空格分割，数值均大于等于 1，小于 100000

输出描述：

输出一个哈夫曼树的中序遍历的数组，数值间以空格分割

示例 1

输入：

```
5
5 15 40 30 10
```

输出：

```
40 100 30 60 15 30 5 15 10
```

说明：

根据输入，生成哈夫曼树，按照中序遍历返回。所有节点中，左节点权值小于等于右节点权值，根节点权值为左右节点权值之和。当左右节点权值相同时，左子树高度高度小于等于右子树。结果


```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <unordered_map>
#include <unordered_set>
#include <set>
#include <queue>
#include <map>
#include <deque>
#include <sstream>
#include <cstring>
#include <functional>

using namespace std;

struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};

vector<int> getInput(int n) {
    vector<int> v(n);
    for(int i = 0; i < n; i++) {
        cin >> v[i];
    }

    return v;
```

```
}

TreeNode* createHafmanTree(vector<int> v) {
    // 将为每个元素创建结点并推入小顶堆中

    deque<pair<int, TreeNode*>> pq;

    for(int i = 0; i < (int)v.size(); i++) {
        TreeNode* node = new TreeNode(v[i]);
        pq.push_back(make_pair(v[i], node));
    }
    sort(pq.begin(), pq.end(), [](pair<int, TreeNode*> a, pair<int, TreeNode*> b){
        return a.first > b.first;
    });

    // for(int i = 0; i < pq.size(); i++) {
    //         cout << pq[i].first << " ";
    // }
    // cout << endl;

    while(pq.size() > 1) {
        pair<int, TreeNode*> node1 = pq.back();
        pq.pop_back();
        pair<int, TreeNode*> node2 = pq.back();
        pq.pop_back();
        // cout << node1.first << " " << node2.first << endl;

        // node1.first <= node2.first
        // root 左结点小，右结点大
        TreeNode* root = new TreeNode(node1.first + node2.first, node1.second,
node2.second);
        // 如果两结点的值相同，左子树的深度要小于右子树

        // pq.push_back(make_pair(root->val, root));
        pq.insert(pq.begin(), make_pair(root->val, root));
        stable_sort(pq.begin(), pq.end(), [](pair<int, TreeNode*> a, pair<int, TreeNode*>
b){return a.first > b.first;});
    }

    return pq.back().second;
}

void inOrder(TreeNode* root) {
    if(root == NULL) return;
```

```
    inOrder(root->left);
    cout << root->val << " ";
    inOrder(root->right);
}

void preOrder(TreeNode* root) {
    if(root == NULL) return;

    cout << root->val << " ";
    preOrder(root->left);
    preOrder(root->right);
}

int main()
{
    int n;
    cin >> n;
    vector<int> v = getInput(n);

    // 将节点值存入小顶堆
    priority_queue<int, vector<int>, greater<int>> pq;
    for(int i = 0; i < n; i++) {
        pq.push(v[i]);
    }

    TreeNode* root = createHafmanTree(v);

    // cout << root->val << endl;

    inOrder(root);
    // cout << endl;
    // preOrder(root);

    return 0;
}

/*
5
5 15 40 30 10

*/
```