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。

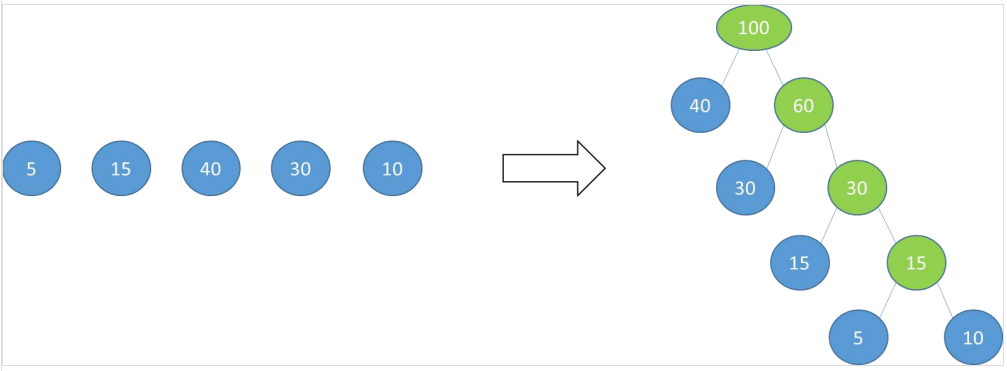
为了保证输出的二叉树中序遍历结果统一，增加以下限制：二叉树节点中，左节点权值小于等于右节点权值，根节点权值为左右节点权值之和。当左右节点权值相同时，左子树高度高度小于等于右子树。

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：

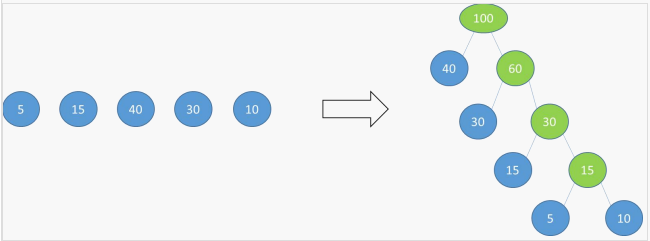
```
5
5 15 40 30 10
```

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




```
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int n = input.nextInt();
        int[] values = new int[n];
        for (int i = 0; i < n; i++) {
            values[i] = input.nextInt();
        }
        Node root = buildHuffman(values);
        int[] ints = inorderTraversal(root);
        for (int anInt : ints) {
            System.out.print(anInt + " ");
        }
    }
    public static Node buildHuffman(int[] values) {
        PriorityQueue<Node> pq = new PriorityQueue<Node>((a, b) -> a.value - b.value);
        for (int value : values) {
            pq.offer(new Node(value));
        }
        while (pq.size() > 1) {
            Node left = pq.poll();
            Node right = pq.poll();
            Node parent = new Node(left.value + right.value);
            parent.left = left;
            parent.right = right;
            pq.offer(parent);
```

```
            }
            return pq.poll();
        }

        public static int[] inorderTraversal(Node root) {
            ArrayList<Integer> result = new ArrayList<>();
            inorderHelper(root, result);
            int[] inorder = new int[result.size()];
            for (int i = 0; i < result.size(); i++) {
                inorder[i] = result.get(i);
            }
            return inorder;
        }

        public static void inorderHelper(Node node, List<Integer> result) {
            if (node == null) {
                return;
            }
            inorderHelper(node.left, result);
            result.add(node.value);
            inorderHelper(node.right, result);
        }
}
class Node {
        int value;
        Node left;
        Node right;
        public Node(int value) {
            this.value = value;
        }
}
```