## 根据完全二叉树计算根节点位置

## 例题：**1064** **Complete Binary Search Tree** (30分)

A Binary Search Tree (BST) is recursively defined as a binary tree which has the following properties:

* The left subtree of a node contains only nodes with keys less than the node's key.
* The right subtree of a node contains only nodes with keys greater than or equal to the node's key.
* Both the left and right subtrees must also be binary search trees.

A Complete Binary Tree (CBT) is a tree that is completely filled, with the possible exception of the bottom level, which is filled from left to right.

Now given a sequence of distinct non-negative integer keys, a unique BST can be constructed if it is required that the tree must also be a CBT. You are supposed to output the level order traversal sequence of this BST.

### Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer *N* (≤1000). Then *N* distinct non-negative integer keys are given in the next line. All the numbers in a line are separated by a space and are no greater than 2000.

### Output Specification:

For each test case, print in one line the level order traversal sequence of the corresponding complete binary search tree. All the numbers in a line must be separated by a space, and there must be no extra space at the end of the line.

### Sample Input:

10  
1 2 3 4 5 6 7 8 9 0

### Sample Output:

6 3 8 1 5 7 9 0 2 4

## 思路：

1. 首先，二叉查找树的中序历遍是一段有序序列。将题目给的序列进行排序可以得到二叉树的中序历遍结果。
2. 接下来要确定的就是根节点的位置。我们可以根据他是一颗完全二叉树来确定根节点
   * 找出小于等于节点个数的完美二叉树的层数
   * 计算左子树的结点数，应为

* 再计算完美二叉树之外多出的结点
* 如果length大于该层数最大结点的一半
* 令
* 反之不作操作
* 计算完美二叉树的左子树结点个数
* 可以得到root结点的下标

1. 计算出root后，计算数组的start和end，确定index，就可以得到数组层序序列
2. 递归地解决问题

**计算root的函数**

int computeRoot(int number)  
{  
 int n = log(number + 1) / log(2);  
 int length = number - pow(2, n) + 1;  
 if ((double)length > pow(2, n - 1))  
 length = pow(2, n - 1);  
 return pow(2, n - 1) - 1 + length;  
}

**建树函数**

void buildTree(int root,int index,int start,int end)  
{  
 if (start > end)  
 return;  
 tree[index] = seq[root];   
 buildTree(start + computeRoot(root - start), index \* 2 + 1, start, root - 1);  
 buildTree(root + 1 + computeRoot(end - root), index \* 2 + 2, root + 1, end);  
}

**完整代码**

#include<iostream>  
#include <vector>  
#include <cmath>  
#include <algorithm>  
using namespace std;  
vector<int> tree, seq;  
int number;  
void init()  
{  
 cin >> number;  
 tree.assign(number, 0);  
 seq.assign(number, 0);  
 for(int i=0;i<number;i++)  
 {  
 cin >> seq[i];  
 }  
 sort(seq.begin(), seq.end());  
}  
int computeRoot(int number)  
{  
 int n = log(number + 1) / log(2);  
 int length = number - pow(2, n) + 1;  
 if ((double)length > pow(2, n - 1))  
 length = pow(2, n - 1);  
 return pow(2, n - 1) - 1 + length;  
}  
void buildTree(int root,int index,int start,int end)  
{  
 if (start > end)  
 return;  
 tree[index] = seq[root];   
 buildTree(start + computeRoot(root - start), index \* 2 + 1, start, root - 1);  
 buildTree(root + 1 + computeRoot(end - root), index \* 2 + 2, root + 1, end);  
}  
int main()  
{  
 init();  
 buildTree(computeRoot(number), 0, 0, number - 1);  
 cout << tree[0];  
 for(int i=1;i<number;i++)  
 {  
 cout << " " << tree[i];  
 }  
 return 0;  
}