

**直接插入：**T=1+2+……+n=n(n+1)/2=O(n^2)

**快速排序:**

void quicksort(int left,int right)

{

int i,j,t,temp;

if(left>right) return;//递归结束标志

temp=a[left];//temp存基准数

i=left;j=right;//左右指针

while(i!=j)

{ while(a[j]>=temp && i<j) j--;

while(a[i]<=temp && i<j) i++;

if(i<j) {t=a[j];a[j]=a[i];a[i]=t;}

}

a[left]=a[i];a[i]=temp;//基准数归位

quicksort(left,i-1);

quicksort(i+1,right);

}

**归并排序:**

void merge(int low, int mid, int top)

{

int i=low, j=mid+1, k=low;

while(i<=mid && j<=top)

if(A[i]>A[j]) B[k++]=A[j++];

else B[k++]=A[i++];

while(i<=mid) B[k++]=A[i++];

while(j<=top) B[k++]=A[j++];

for(i=low;i<=top;i++) A[i]=B[i];

}

void mergesort(int low,int top)

{

if(low<top)

{ int mid=(low+top)/2;

mergesort(low,mid);

mergesort(mid+1,top);

merge(low,mid,top);

}

}

1.仅基于比较的算法能得到的最好的“最坏时间复杂度”是O(NlogN)。(Y)

2.对N个不同的数据采用冒泡排序进行从大到小的排序，当元素基本有序时交换元素次数肯定最多。(X)

3.要从50个键值中找出最大的3个值，选择排序比堆排序快。(Y)

4.对N个不同的数据采用冒泡排序进行从大到小的排序，当元素基本有序时交换元素次数肯定最多。(X)当元素从小到大排时(基本有序指从大到小排好)

5.对N个记录进行归并排序，归并排序的趟数是logN（递归的层次上），时间复杂度是N\*logN。

6.输入10510^510​5​​个只有一位数字的整数，可以用O(N)复杂度将其排序的算法是：（桶排序）

7.对10TB的数据文件进行排序，应使用的方法是：(堆排序)

8.在快速排序的一趟划分过程中，当遇到与基准数相等的元素时，如果左指针停止移动，而右指针在同样情况下却不停止移动，那么当所有元素都相等时，算法的时间复杂度是多少？(O(N^2))(T=n+……+2+1)

**满二叉树：**除叶子结点外的所有结点均有两个子结点，所有叶子结点必须在最底下一层。高H的满二叉树有2^(H+1)-1个节点。

**完全二叉树：**也称近似满二叉树，所有叶结点在最下面两层，最底下一层节点集中在左边。（书P113）

**哈夫曼树：**A、每个节点要么没有子节点，要么有两个子节点，树中一定没有度为1的结点。

B、树中两个权值最小的结点一定是兄弟结点 。

C、树中任一非叶结点的权值一定不小于下一层任一结点的权值。

结点的带权路径长度：结点到树根之间的路径长度与该结点上权的乘积.

树的带权路径长度：所有叶子结点的带权路径长度之和，记为WPL。[

# 线索二叉树：n个结点的二叉树中含有n+1个空指针域。(每个节点有2个空指针,n\*2-(n-1))利用二叉链表中的空指针域，存放指向[结点](http://baike.baidu.com/view/549491.htm" \t "_blank)在某种遍历次序下的前趋和后继结点的指针（这种附加的指针称为"线索"）。

# 如下，后序遍历dbca，后序线索二叉树是：

# 

# 森林：对任意二叉树有N个2度节点，则该树有且仅有N+1个叶节点。(2度节点会兵分两路,1度节点只有笔直一条路,0度结点是路的终点)

# 二叉树组成的森林F中，N个叶结点，M个度为2的结点，则树的个数为N-M。

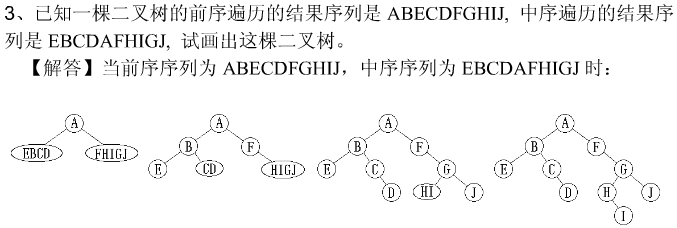
# 森林F有15条边、25个结点，则树的个数是：10

# 森林转换为二叉树的方法：第一棵二叉树不动,从第二棵二叉树开始,依次把后一棵二叉树的根结点作为前一棵二叉树根结点的右孩子,当所有二叉树连在一起后,所得到的二叉树就是由森林转换得到的二叉树.

# 

1.对一棵平衡二叉树，所有非叶结点的平衡因子都是0，当且仅当该树是完全二叉树。(X)

2.一棵有124个结点的完全二叉树，其叶结点个数是确定的。**(Y)**



4.(1)二叉树的前序序列与中序序列相同：空树或缺左子树的单支树；

(2)二叉树的中序序列与后序序列相同：空树或缺右子树的单支树；

(3)二叉树的前序序列与后序序列相同：空树或只有根结点的二叉树。

(4)二叉树的前序序列和后序序列相反: 缺左子树的单支树或缺右子树的单支树，即高度等于其结点数

5.二叉搜索树的查找和折半查找的时间复杂度相同.(X) 二叉排序树不一定是平衡树,只有是一棵平衡的二叉排序树时，其查找时间性能才和折半查找类似。

6.若二叉搜索树是有NNN个结点的完全二叉树，则正确的说法是：ABD

A所有结点的平均查找效率是O(logN)

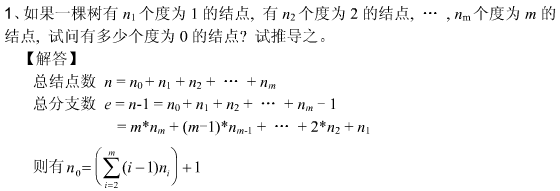
B最小值一定在叶结点上

C最大值一定在叶结点上

D中位值结点在根结点或根的左子树上

7.如果AVL树的深度为5（空树的深度定义为0），则此树最少有多少个结点 12 。12个结点的AVL树的最大深度是 5 。

8.由分别带权为9、2、5、7的四个叶子结点构成一棵哈夫曼树，该树的带权路径长度为： 44=2\*3+5\*3+7\*2+9\*1 。



# 数构中的度是指儿子个数。度为0的结点指树的叶结点。

5.有一个四叉树，度2的结点数为2，度3的结点数为3，度4的结点数为4。问该树的叶结点个数是 **21** (分支数m= n-1= 2+3+4+x-1= 2\*2+3\*3+4\*4+x\*0)

6.如果一棵非空k（k≥2）叉树T中每个非叶子结点都有k个孩子，则称T为正则k叉树。若T的高度为h（单结点的树h=1），则T的结点数最少为： k(h−1)+1

7.树最适合于用来表示元素之间无联系的数据

8.已知一棵二叉树的先序遍历结果是ABC，则CAB序列是不可能的中序遍历结果。

9.任何一棵二叉树的叶子结点在先序、中序和后序遍历序列中的相对次序（A）

A．不发生改变 B．发生改变

C．不能确定 D．以上都不对

10. 具有1102个结点的完全二叉树一定有\_\_个叶子结点 551

11.已知一棵完全二叉树的第9层（设根为第1层）有100个叶结点，则该完全二叉树的结点个数最多是： 823

满二叉树：第一层结点数2^0=1，总结点数2^1-1=1；

第二层结点数2^1=2，总结点数2^2-1=3；

第三层结点数2^2=4，总结点数2^3-1=7；

……

第十层结点数2^9=512个，总结点数2^10-1=1023；

第十一层2^10=1024，总结点数2^11-1=2047；

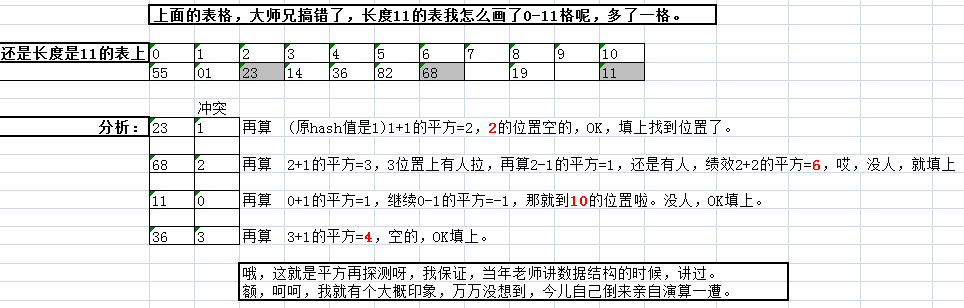
第十二层2^11=2048，总结点数2^3-1=7；

12设森林F中有三棵树，第一、第二、第三棵树的结点个数分别为M1​​，M2​​和M3。则与森林F对应的二叉树根结点的右子树上的结点个数是：M2+M3

**线性探测和平方探测的区别在冲突时找自己的位置的差别，**

**一个是挨个查找**hi(k)=(H(k)+1)%11**；**

**一个是高级点，**±**n^2 (+1,-1,+4,-4,+9,-9……)的查找**hi(k)=(H(k)±i^2)%11**。**



**装载因子**是指所有关键子填充哈希表后饱和的程度，它等于 关键字总数/哈希表的长度。

**计算等概率情况下查找成功和查找不成功的平均查找长度:**

****

1.对包含N个元素的散列表进行查找，平均查找长度为：不确定。由于散列表的一个重要特征是平均检索长度不直接依赖于元素个数n。平均检索长度不随表中元素增加而增加，而是随负载因子增大而增加。

2.给定输入序列 {4371, 1323, 6173, 4199, 4344, 9679, 1989} 以及散列函数 h(X)=X%10。如果用大小为10的散列表，并且用平方探测解决冲突，则输入各项经散列后在表中的下标为：（-1表示相应的插入无法成功）(C)

A 1, 3, 3, 9, 4, 9, 9

B 1, 3, 4, 9, 7, 5, -1

C 1, 3, 4, 9, 5, 0, 8

D 1, 3, 4, 9, 5, 0, 2

1.已知不相交集合用数组表示为{ 4, 6, 5, 2, -3, -4, 3 }。若集合元素从1到7编号，则调用Union(Find(7),Find(1))（按规模求并，并且带路径压缩）后的结果数组为：{ 6, 6, 5, 6, 6, -7, 5 }

**拓扑排序：**要找到入度为0的点，将其存入线性序列中，再将其从图中删除（与它相关的边都删除，相邻的顶点的入度均减1），再重复上面的操作，直至所有的顶点都被找到为止。

　　　　　对一个[有向无环图](http://baike.baidu.com/view/6204968.htm" \t "_blank)进行拓扑排序，是将G中所有顶点排成一个线性序列，使得图中任意一对顶点u和v，若边(u,v)∈E(G)，则u在线性序列中出现在v之前。通常，这样的线性序列称为拓扑序列。

　　　　　只要当前顶点还存在边指向其它任何顶点，它就会递归调用dfs方法，而不会退出。因此，退出dfs方法，意味着当前顶点没有指向其它顶点的边了，即当前顶点是一条路径上的最后一个顶点。最先退出DFS函数的顶点是出度为零的顶点,为拓 扑排序序列中最后一个顶点。 因此,按退出DFS函数的先后记录下来的顶点序列 即为逆向的拓扑排序序列。

１.无向连通图中，至少有一个顶点的度为1。（Ｘ）

顶点的度定义：与定点v相关联的边数(每个环计算两次)。度为零的顶点称为孤立顶点，度为奇数的顶点称为奇点，度为偶数的顶点称为偶点。

２．下面关于图的存储的叙述中，正确的是（Ａ)。

A．用邻接矩阵法存储图，占用的存储空间数只与图中结点个数有关，而与边数无关

B．用邻接矩阵法存储图，占用的存储空间数只与图中边数有关，而与结点个数无关

C．用邻接表法存储图，占用的存储空间数只与图中结点个数有关，而与边数无关

D．用邻接表法存储图，占用的存储空间数只与图中边数有关，而与结点个数无关

邻接矩阵的空间复杂度为O(n2)，与边的个数无关。邻接表的空间复杂度为O(n+e)，与图中的结点个数和边的个数都有关。

10线性表、堆栈、队列的主要区别是什么？堆栈和队列都是插入、删除受到约束的线性表    
11.在拓扑排序算法中用堆栈和用队列产生的结果会不同吗？ **C**   
 A、是的肯定不同  B、肯定是相同的   
 C、有可能会不同  D、以上全不对

对一个[有向无环图](http://baike.baidu.com/view/6204968.htm" \t "_blank)(Directed Acyclic Graph简称DAG)G进行拓扑排序，是将G中所有顶点排成一个线性序列，使得图中任意一对顶点u和v，若边(u,v)∈E(G)，则u在线性序列中出现在v之前。通常，这样的线性序列称为满足拓扑次序(Topological Order)的序列，简称拓扑序列。简单的说，由某个集合上的一个[偏序](http://baike.baidu.com/view/843128.htm)得到该集合上的一个[全序](http://baike.baidu.com/view/1366789.htm)，这个操作称之为拓扑排序。

在整个[工程](http://baike.baidu.com/view/48840.htm)中，有些子工程(活动)必须在其它有关子工程完成之后才能开始，也就是说，一个子工程的开始是以它的所有前序子工程的结束为先决条件的，但有些子工程没有先决条件，可以安排在任何时间开始。为了形象地反映出整个工程中各个子工程(活动)之间的先后关系可以用AOV网表示。一个AOV网应该是一个[有向无环图](http://baike.baidu.com/view/6204968.htm)，即不应该带有回路，因为若带有回路，则回路上的所有活动都无法进行。

12.在利用堆栈将下列哪个中缀表达式转换为后缀表达式过程中，堆栈的操作系列为: push(′∗′)、push(′(′)、push(′+′)、pop()、pop()、pop()、push(′∗′)、pop()、push(′+′)、pop()   **A**  
 A、2\*(3+4)\*5+6  B、2\*(3+4\*5)+6   
 C、2\*(3+4\*5+6)  D、以上都是

程序填空题

3-1 将小顶堆H中指定位置P上的元素的整数键值减小D个单位，然后继续将H调整为小顶堆。

void DecreaseKey( int P, int D, PriorityQueue H )

{

int i, key;

key = H->Elements[P] - D;

for ( i = P; H->Elements[i/2] > key; i/=2 )

H->Elements[i]=H->Elements[i/2];

H->Elements[i] = key;

}

3-2 将一列元素{ r[1] … r[n] }按非递减顺序排序。普通选择排序是每次仅将一个待排序列的最小元放到正确的位置上，而这个另类的选择排序是每次从待排序列中同时找到最小元和最大元，把它们放到最终的正确位置上。

void sort( list r[], int n )

{

int i, j, mini, maxi;

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

mini = maxi = i;

for( j=i+1; j<=n-i+1; ++j ){

if( r[j]->key < r[mini]->key ) mini = j;

else if(r[j]->key > r[maxi]->key) maxi = j;

}

if( mini!=i ) swap(&r[mini], &r[i]);

if( maxi != n-i+1 ){

if( maxi==i ) swap(&r[mini], &r[n-i+1]);

else swap(&r[maxi], &r[n-i+1]);

}

}

}

编程题

本题要求根据给定的一棵二叉树的后序遍历和中序遍历结果，输出该树的先序遍历结果。   
输入格式:

第一行给出正整数NNN(≤30\le 30≤30)，是树中结点的个数。随后两行，每行给出NNN个整数，分别对应后序遍历和中序遍历结果，数字间以空格分隔。题目保证输入正确对应一棵二叉树。   
输出格式:

在一行中输出Preorder:以及该树的先序遍历结果。数字间有1个空格，行末不得有多余空格。   
输入样例:

7   
2 3 1 5 7 6 4   
1 2 3 4 5 6 7

输出样例:

Preorder: 4 1 3 2 6 5 7

参考代码

#include <iostream>

#include <cstring>

using namespace std;

void getpre(int \*post, int \*in, int n) {

if(n <= 0) return;

int root = post[n - 1];

int i;

for(i = 0; i < n; i++) {

if(in[i] == root) {

break;

}

}

cout << ' ' << root;

getpre(post, in, i);

getpre(post + i, in + i + 1, n - i - 1);

}

int main() {

int post[40], in[40];

int n;

cin >> n;

int i, j;

for(i = 0; i < n; i++) {

scanf("%d", &post[i]);

}

for(i = 0; i < n; i++) {

scanf("%d", &in[i]);

}

printf("Preorder:");

getpre(post, in, n);

printf("\n");

return 0;

}