# 习题四

## 4.3

* 递归的插入排序：

typedef int data\_type;

typedef std::vector<data\_type> array\_t;

void insert\_sort(array\_t & A, const int n)//A[:n]已经排序

{

if (n == A.size() - 1) { return; }

data\_type tmp = A[n];

int i = n - 1;

while (i > 0 && A[i] > tmp)

{

A[i + 1] = A[i];

--i;

}

A[i] = tmp;

insert\_sort(A, n + 1);

}

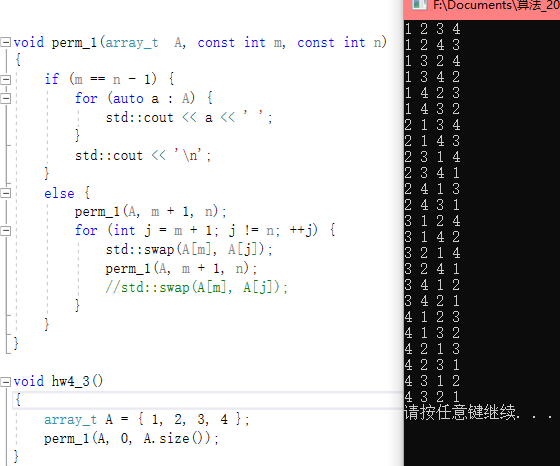
* 复杂度分析：

时间复杂度取决于已经排序的数组，递归深度为整个数组的大小，有递推公式：

所以复杂度依旧是

## 4.5

* 对程序的修改：



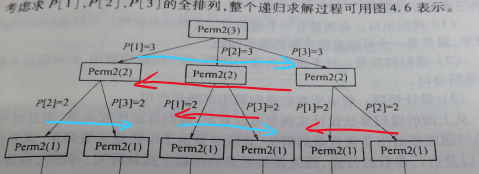
将原来循环中，下面的swap去掉，并且修改函数参数的引用传递为值传递，保证了：

1. 在一个函数调用中循环调用swap，形成一个有序(字典序)的轮换
2. 在函数递归调用栈间传递不相互影响，但是增加了存储开销

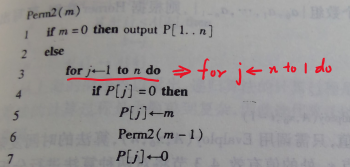
测试程序使用了数值代替字母进行测试

## 4.7

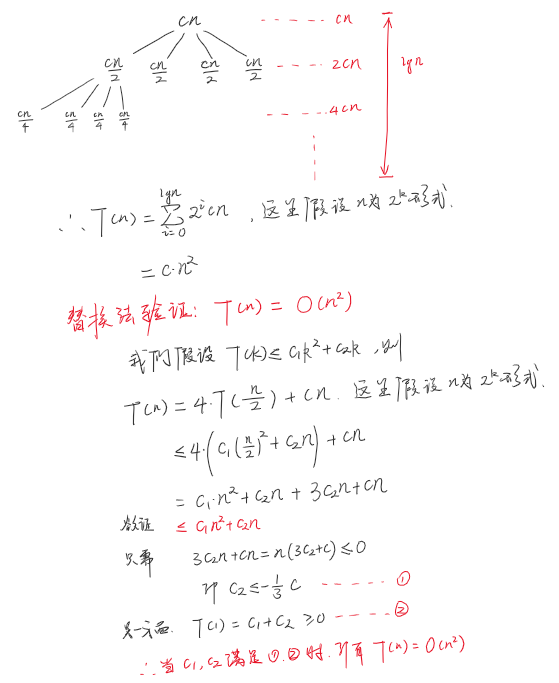
用递归树分析Perm2的调用：



* 蓝色箭头为Perm2的调用顺序，我们的目标则是红色箭头的顺序，因此只要修改所有的for循环为原来的逆序，即从大到小，即可实现和原来输出顺序相反



## 4.12 画出递归树并用替换法进行分析：



## 4.15

1. 解：

所以：

1. 解：

所以：

1. 解：

所以：