1. Generate Parentheses

Given *n* pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given *n* = 3, a solution set is:

[  
 "((()))",  
 "(()())",  
 "(())()",  
 "()(())",  
 "()()()"  
]

**解法1**

暴力求解。先根据生成一个序列，求出的全排列，逐个判断是否合法

*效率太低*

时间复杂度为，空间复杂度为

Note：

全排列求解见“十大算法精讲ch2字符串”，包含重复符号的全排列求解方法，注意到交换的条件：

**第个数与第个数交换时，要求中没有与第个数相等的数。**

class Solution {  
public:  
 //函数主体  
 vector<string> generateParenthesis(int n) {  
 string s;  
 for(int i = 0; i < n; ++i)s += '(';  
 for(int i = n; i < 2 \* n; ++i)s += ')';  
 vector<string>ans;  
 dfs(0, s, ans);  
 return ans;  
 }  
 //全排列求解  
 void dfs(int start, string &s, vector<string>&ans){  
 if(start == s.size()){  
 if(valid(s) == true)ans.push\_back(s);  
 return;  
 }  
 for(int i = start; i < s.size(); ++i){  
 if(isswap(start, i, s) == false)continue;  
 swap(s[start], s[i]);  
 dfs(start + 1, s, ans);  
 swap(s[start], s[i]);  
 }  
 }  
 //序列合法判定  
 bool valid(const string &s){  
 stack<char>v;  
 bool flag = true;  
 for(int i = 0; i < s.size(); ++i){  
 if(s[i] == '(')v.push(s[i]);  
 else{  
 if(v.size() > 0 && v.top() == '(')v.pop();  
 else{  
 flag = false;  
 break;  
 }  
 }  
 }  
 if(v.size() != 0)flag = false;  
 return flag;  
 }  
 //可交换判定  
 bool isswap(int x, int y, string &s){  
 bool flag = true;  
 for(int i = x; i < y; ++i){  
 if(s[i] == s[y]){  
 flag = false;  
 break;  
 }  
 }  
 return flag;  
 }  
};

**解法2**

后向搜索。思路同暴力基本相似，但是在生成排列的每一步中，都保证最终能够得到一个合法的序列。此种策略对解法1进行了剪枝

*效率大幅度提升*

时间复杂度：

空间复杂度：

class Solution {  
public:  
 vector<string> generateParenthesis(int n) {  
 vector<string>ans;  
 string s = "";  
 backTrack(ans, s, 0, 0, n);  
 return ans;  
 }  
 void backTrack(vector<string>&ans, string s, int open, int close, int max){  
 if(s.size() == 2 \* max){  
 ans.push\_back(s);  
 return;  
 }  
 if(open < max)backTrack(ans, s +"(", open + 1, close, max);  
 if(close < open)backTrack(ans, s + ")", open, close + 1, max);  
 }  
};