波兰表达式

* + 11.0 12.0 + 24.0 35.0

1357.000000

可使用 atof(str) 把字符串转换为一个double类型的浮点数

题目提示得比较明显，由于输入是字符和数字混合输入，所以统一采用字符串输入比较方便

分析

先分析一下样例：

(2 + 3) * 4  (* (+ 2 3) 4)

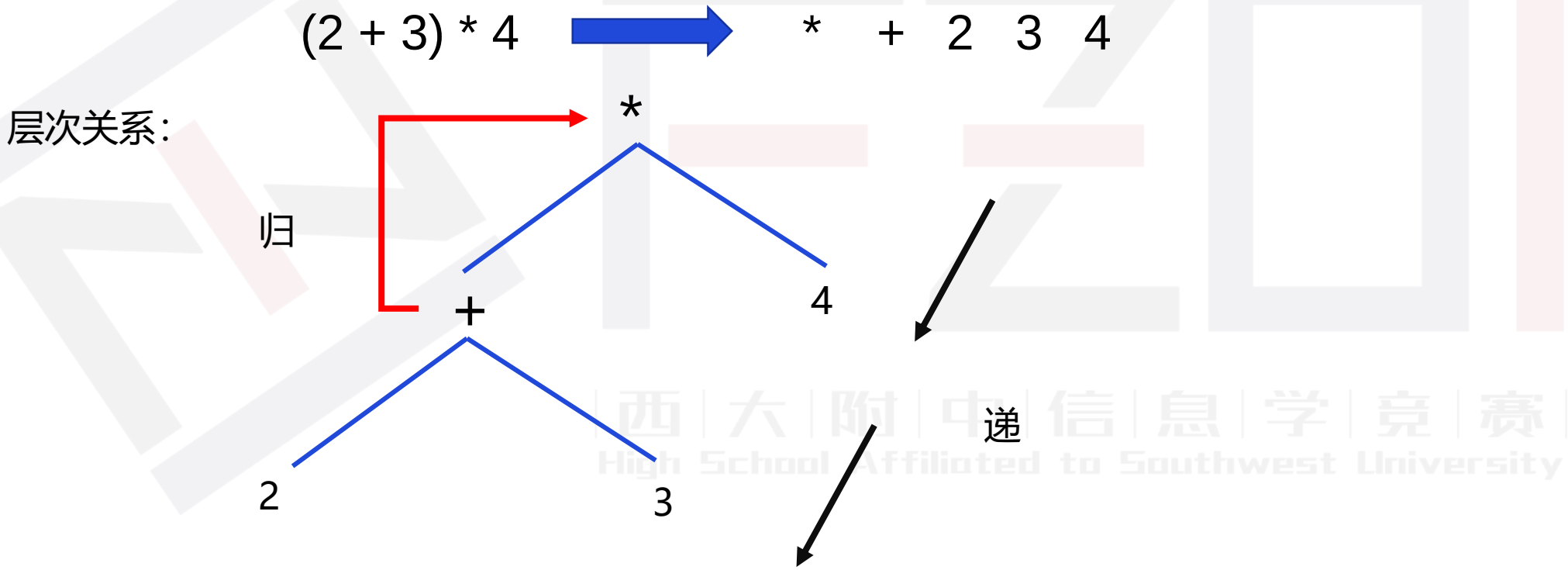
模拟计算过程：

1.读入一个*号，但是没有数字可计算

2.继续读入一个+号，没有数字可计算，继续读入2 3，计算2+3

3.读入4，计算（2+3）*4

读入时，读入的是运算符，就作为一层，先后顺序代表它们的层次



参考代码

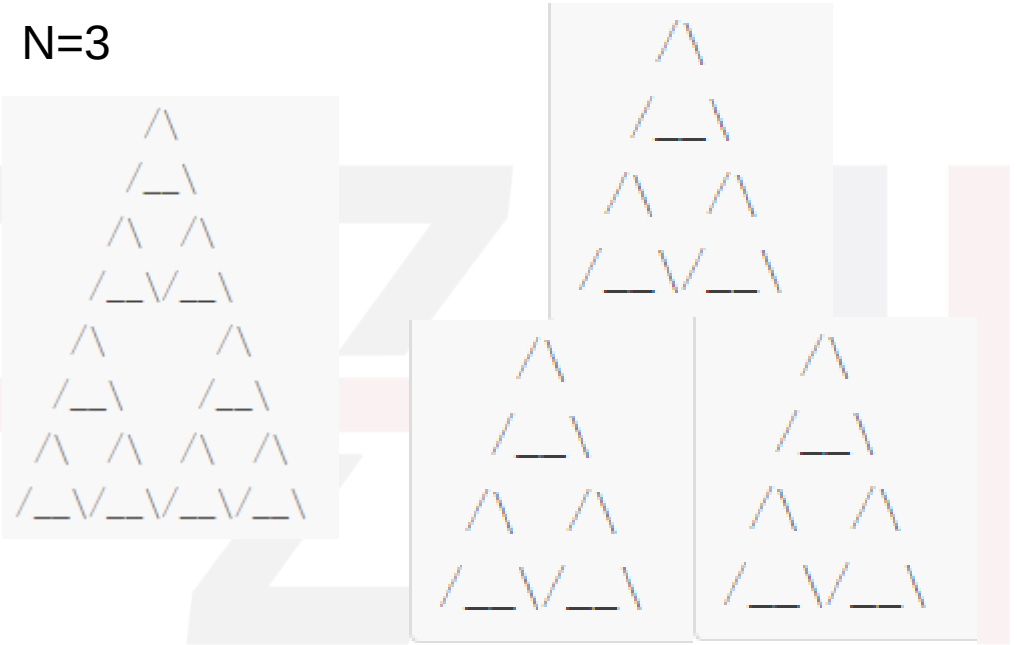

```
double f()
{
    char t[50];
    cin>>t;
    int len=strlen(t);
    if(t[0]=='+')
            return f()+f();
    else if(t[0]=='-'&&len==1) //去除是负数的情况
            return f()-f();
    else if(t[0]=='*')
            return f()*f();
    else if(t[0]=='/')
            return f()/f();
    else
      return atof(t);
}
```

南蛮图腾

西南大学附属中学
High School Affiliated to Southwest University

N=2

N=3



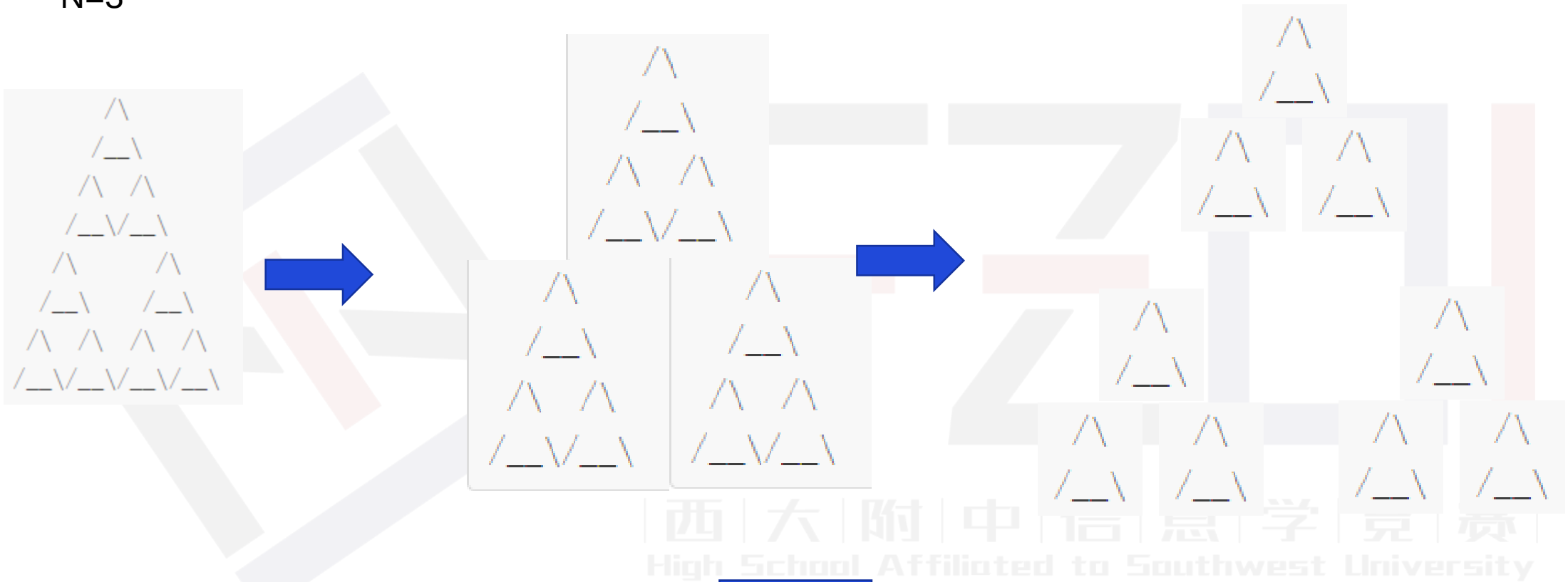
N=3的图案，是N=2的图案构成的
N=2的图案，是N=1的图案构成的

相似的问题：每次画顶端的三角形、左下、右下的三角形

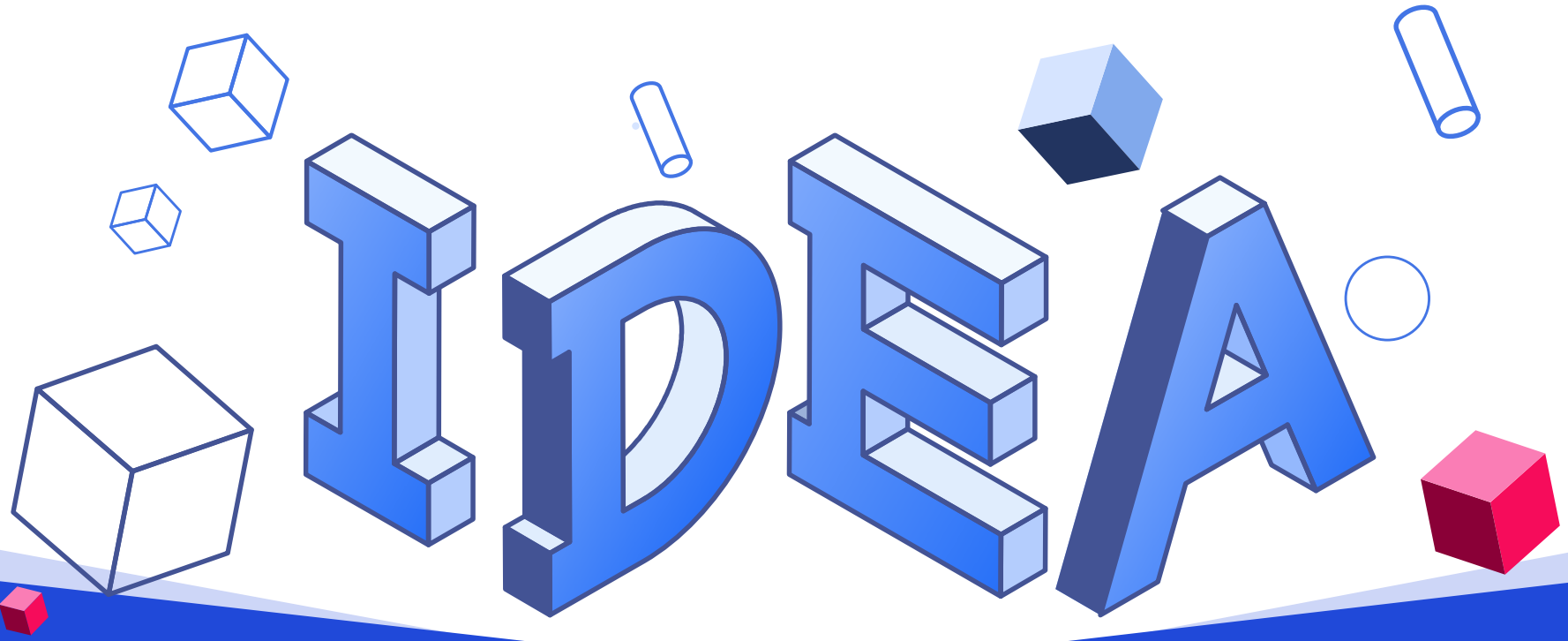
绘制问题分解为三个部分三角形的绘制，即可得到整体的图案

南蛮图腾

N=3



分形图



信息学

分治法(Divide and conquer)

西南大学附属中学校

信息奥赛教练组

解释：分而治之，Divide and conquer

求解思想

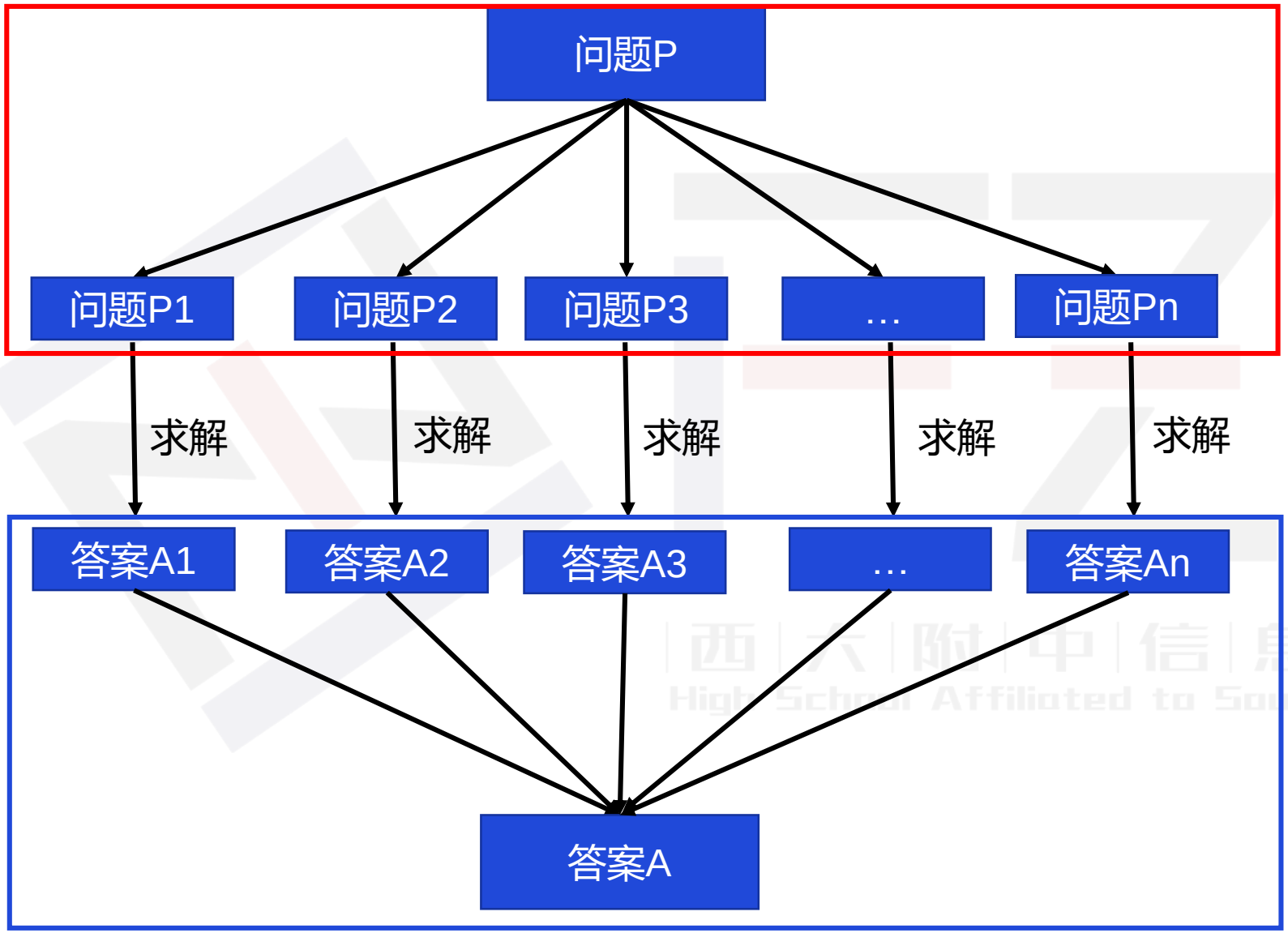
在求解一个规模为n，而n的取值又很大的问题时，直接求解往往非常困难。这时，把问题分割成一些规模较小的相同问题，以便各个击破，分而治之。

分治法是很多高效算法的基础：归并排序和快速排序，快速幂等

分治法**适用的情形**:

1. 问题的规模缩小到一定的规模就可以较容易地解决
2. 问题可以分解为若干个规模较小的模式相同的子问题
3. 合并问题分解出的子问题的解可以得到问题的解
4. 问题所分解出的各个子问题之间是独立的

分治求解过程图



求解过程

1.问题的**分解**
(一般使用递归)

2.子问题的**求解**

3.答案的**合并**

分治法的关键是如何合并子问题的答案

分治一般的程序结构

西南大学附属中学 High School Affiliated to Southwest University

```
void Divide(...){
        if(问题已经小到足以求解或无法再分解){
                //求解过程
        }
        else{
                对原问题进行分解；//分解
                递归对每一个分治的部分进行求解；//解决
                归并整个问题，得出全问题的解；    //合并
        }

}
```

以二分查找为例

二分查找的过程：

以mid为中心点，将区间分为左、右部分
如果k==a[mid] 查找成功
如果k>a[mid] 继续在右半部分查找
如果k<a[mid] 继续在左半部分查找
如果查找不成功，返回-1

```
int BinarySearch(int k,int left,int right){
  if (left>right)
    return -1;
  int mid=(left+right)/2;
  if(k==a[mid])
    return mid;
  else if(k>a[mid]) //查找右半部分
    return BinarySearch (k, mid+1, right);
  else if(k<a[mid]) //查找左半部分
    return BinarySearch (k, left, mid-1);
}
```

与大问题做法相似，只是规模、范围不同

例1：快速幂

求A^B的最后三位数表示的整数。
说明：A^B的含义是“A的B次方”

直接循环求解：
1.计算B次
2.很容易数据溢出

取余运算的几个法则：

(a + b) % p = (a % p + b % p) % p （1）
(a - b) % p = (a % p - b % p) % p （2）
(a * b) % p = (a % p * b % p) % p （3）

Q：如何保证B数值很大的时候也能快速求解呢？

快速幂算法的核心思想：

每一步都把指数分成两半，而相应的底数做平方运算。
这样不仅能把非常大的指数给不断变小，所需要执行的计算次数也变小。

快速幂模板


```
//递归快速幂
int qpow(int a, int n){
    if (n == 0)
        return 1;
    else if (n % 2 == 1) //奇数
        return qpow(a, n - 1) * a;
    else{  //偶数
        int temp = qpow(a, n / 2);
        return temp * temp;
    }
}
```

```
//递归快速幂（对某大素数取模）
const int MOD =1e9+7;
typedef long long ll;
ll qpow(ll a, ll n){
    if (n == 0)
        return 1;
    else if (n % 2 == 1)
        return qpow(a, n - 1) * a % MOD;
    else{
        ll temp = qpow(a, n / 2) % MOD;
        return temp * temp % MOD;
    }
}
```


递归写法也很有可能会爆栈，更好的写法：利用位运算来写快速幂(以后学习)

例2：南蛮图腾

西南大学附属中学 High School Affiliated to Southwest University

输入 #1 复制

```
2
```

输出 #1 复制

```
  /\
 /__\
/\  /\
/__\/__\
```

输入 #2 复制

```
3
```

输出 #2 复制

```
    /\
   /__\
  /\  /\
 /__\/__\
/\    /\
/__\  /__\
/\  /\  /\  /\
/__\/__\/__\/__\
```

参考代码


```
char M[3050][3050];   //存储答案
int n;
void draw(int x,int y,int size){        //x,y表示图形的第一个“/”的坐标
    if(size==1){                  //画出n=1的基本图形
        M[x][y]='/';
        M[x][y+1]='\\';
        M[x+1][y-1]='/';
        M[x+1][y]='_';
        M[x+1][y+1]='_';
        M[x+1][y+2]='\\';
        return;
    }
    draw(x,y,size-1);                          //递归分别画三个部分
    draw(x+pow(2,size-1),y-pow(2,size-1),size-1);
    draw(x+pow(2,size-1),y+pow(2,size-1),size-1);
}

int main(){
    cin>>n;
    for(int i=1;i<=pow(2,n);i++){          //初始化
        for(int j=1;j<=pow(2,n+1);j++)
                    M[i][j]=' ';
    }
    draw(1,pow(2,n),n);
    for(int i=1;i<=pow(2,n);i++){              //输出
        for(int j=1;j<=pow(2,n+1);j++)
        cout<<M[i][j];
        cout<<endl;
    }
}
```

例2：循环日程表

设有N个选手进行循环比赛，其中N=2M，要求每名选手要与其他N-1名选手都赛一次，每名选手每天比赛一次，循环赛共进行N-1天，要求每天没有选手轮空。

输入
M（M<=10)

输出
输出：表格形式的比赛安排表

数字与数字之间的用一个空格隔开

样例输入
3

样例输出
1 2 3 4 5 6 7 8
2 1 4 3 6 5 8 7
3 4 1 2 7 8 5 6
4 3 2 1 8 7 6 5
5 6 7 8 1 2 3 4
6 5 8 7 2 1 4 3
7 8 5 6 3 4 1 2
8 7 6 5 4 3 2 1

1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1

规律：

- 左上=右下
  右上=左下
- 右上-左上=n/2
  n是当前构造的矩形边长

参考代码


```
#include<bits/stdc++.h>
using namespace std;
int a[10000][2000],m;
int mian() {
cin>>m;
int k=1,mid=1; //mid表示当前构造的矩形长度n的一半
a[1][1]=1;
while(k<=m){
    for(int i=1;i<=mid;i++){    //构造上半部分
      for(int j=1;j<=mid;j++){
            a[i][j+mid]=a[i][j]+mid;
        }
    }
    for(int i=1;i<=mid;i++){    //构造下半部分
        for(int j=1;j<=mid;j++){
            a[i+mid][j]=a[i][j+mid];
            a[i+mid][j+mid]=a[i][j];
        }
    }
    mid=mid>>1;  //mid扩大一倍
    k++;
}
for(int i=1;i<=1<<m;i++){
    for(int j=1;j<=1<<m;j++){
        cout<<a[i][j]<<" ";
    }
    cout<<endl;
  }
 return 0;
}
```

分治也不一定完全要用递归来实现

南蛮图腾非递归写法


```
#include<bits/stdc++.h>
using namespace std;
char a[1024][2048];
int main()
{
	int n,length=4,k=1;//length表示当前图腾的宽，length/2是图腾的高
	cin>>n;
	for(int i=0;i<1024;i++)
    for(int j=0;j<2048;j++)
	a[i][j]=' '; //先全部置为空
	a[0][0]=a[1][1]='/',a[0][1]=a[0][2]='_',a[0][3]=a[1][2]='\\'; //存n=1时的基础图腾
	while(k<n) //不断复制
	{
		for(int i=0;i<length/2;i++)
		for(int j=0;j<length;j++)
		a[i+(length/2)][j+(length/2)]=a[i][j+length]=a[i][j];
		length*=2,k++;
	}
	for(int i=(length/2)-1;i>=0;i--)//倒序输出
	{
	    for(int j=0;j<length;j++)
	    cout<<a[i][j];
		cout<<endl;
	}
	return 0;
}
```

小结

- 不是所有的问题都可以采用分治，只有那些能将问题分成与原问题类似的子问题，并且归并后符合原问题的性质的问题，才能进行分治
- 分治的关键点在于“分”之后，如何“治”，即如何合并答案，这个过程没有固定的模板和套路，因题而异

习题

「模板」快速幂

「模版」二分查找

循环比赛日程表

取余运算 加强版

南蛮图腾

地毯填补

黑白棋子的移动

要求：结合ppt的代码理解日程表与南蛮图腾，可以模仿写一遍，但注重理解分治递归的解题思路，思考并尝试书写地毯填补与黑白棋子移动的代码。

目标：让自己的思路接受递归这种思想，并且能够渐渐理解递归，理解分治的过程。

Thanks

For Your Watching