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)

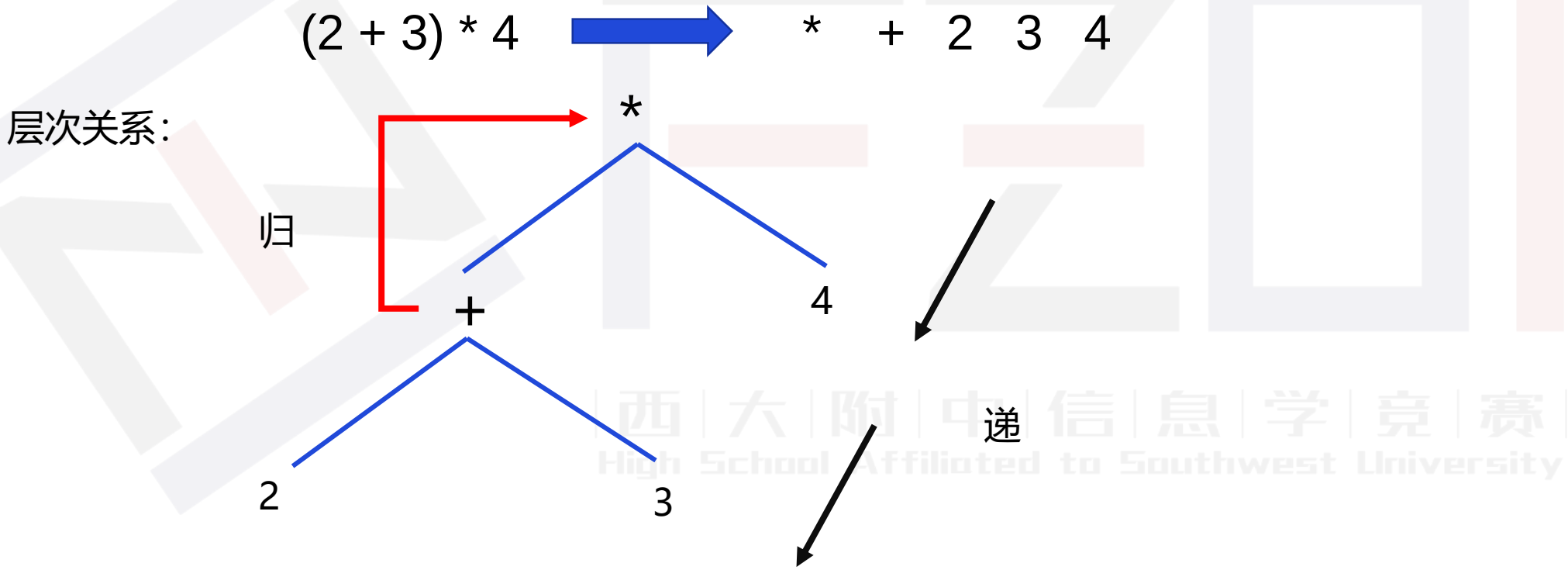
模拟计算过程：

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

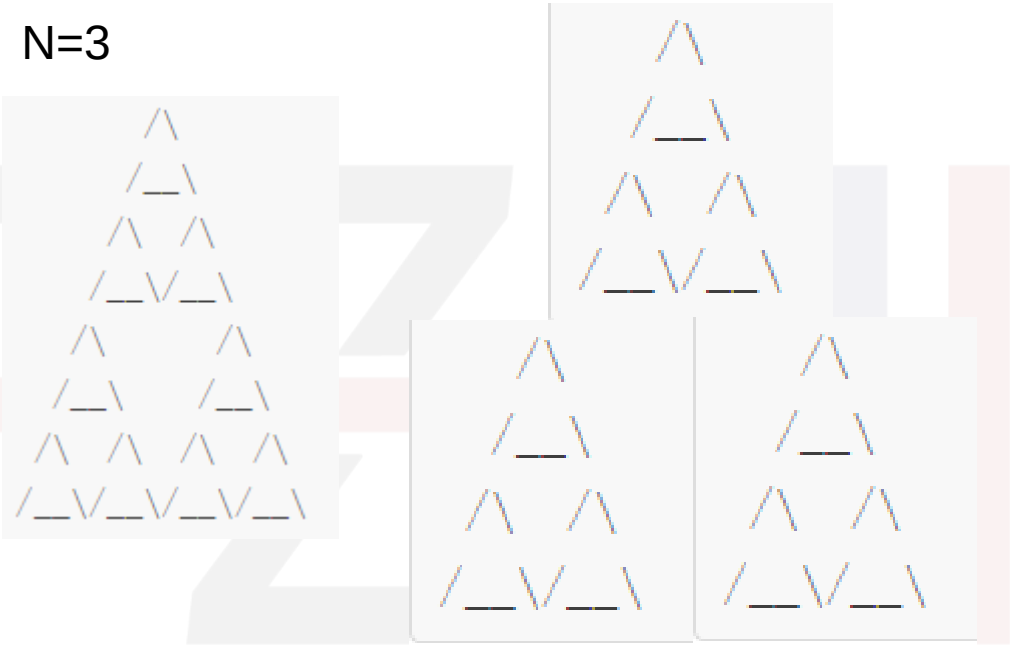

```
double f()
{
    char t[50];
    cin>>t;
    int len=strlen(t);
    if(t[0]=='+')
            return f()+f();
    else if(t[0]=='-'&&len==1) //去除是负数的情况
            return f()-f();
    else if(t[0]=='*')
            return f()*f();
    else if(t[0]=='/')
            return f()/f();
    else
     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



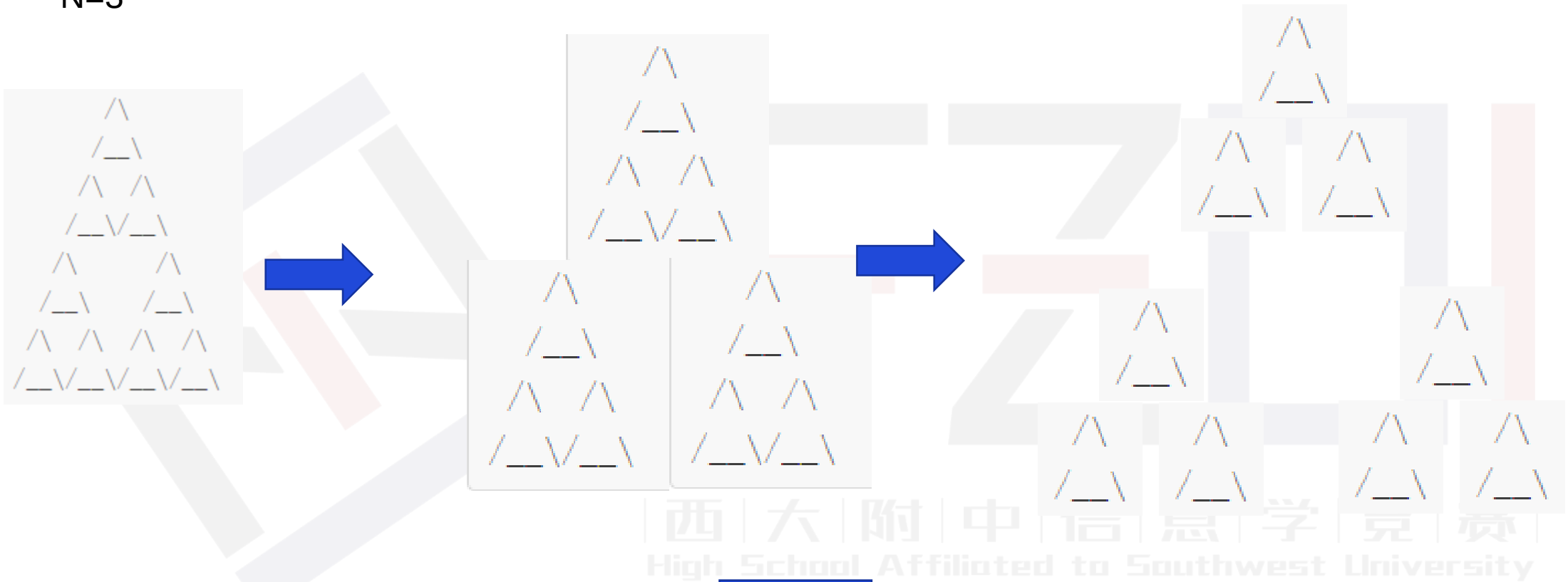
N=3的图案，是N=2的图案构成的
N=2的图案，是N=1的图案构成的

相似的问题：每次画顶端的三角形、左下、右下的三角形

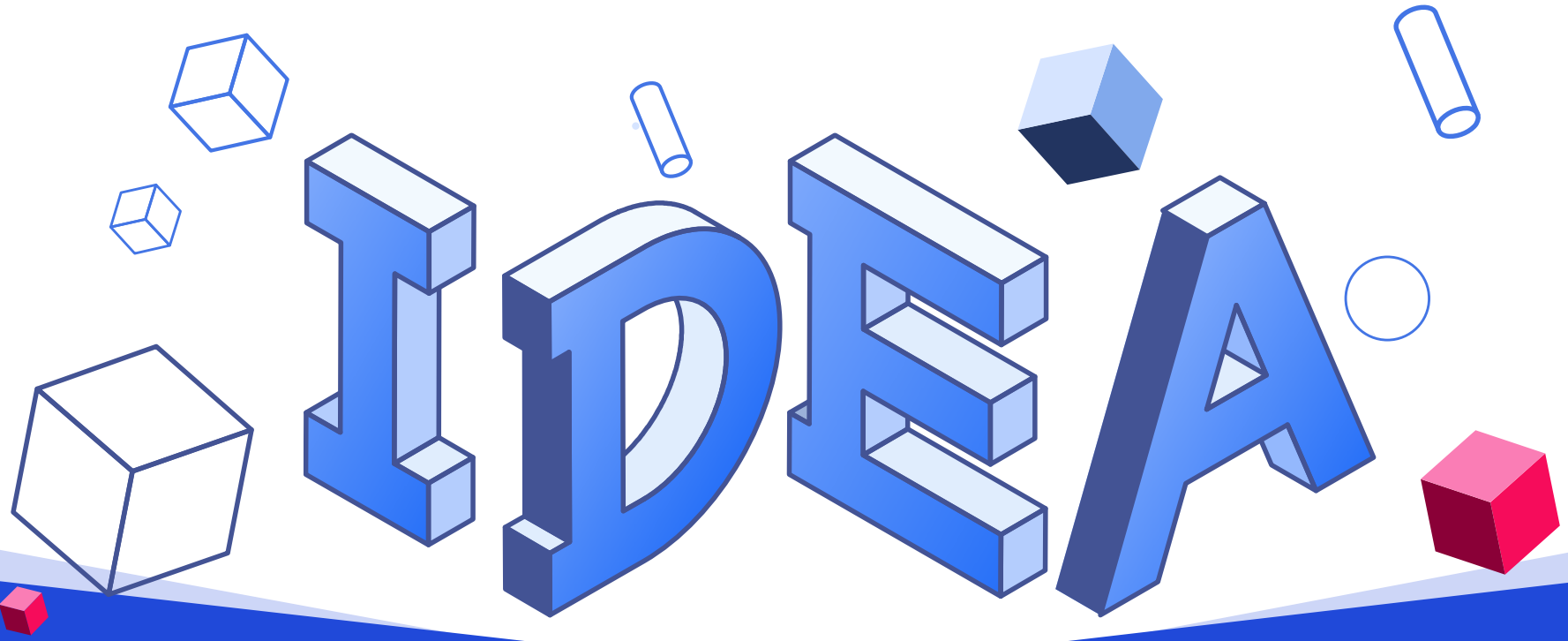
绘制问题分解为三个部分三角形的绘制，即可得到整体的图案

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

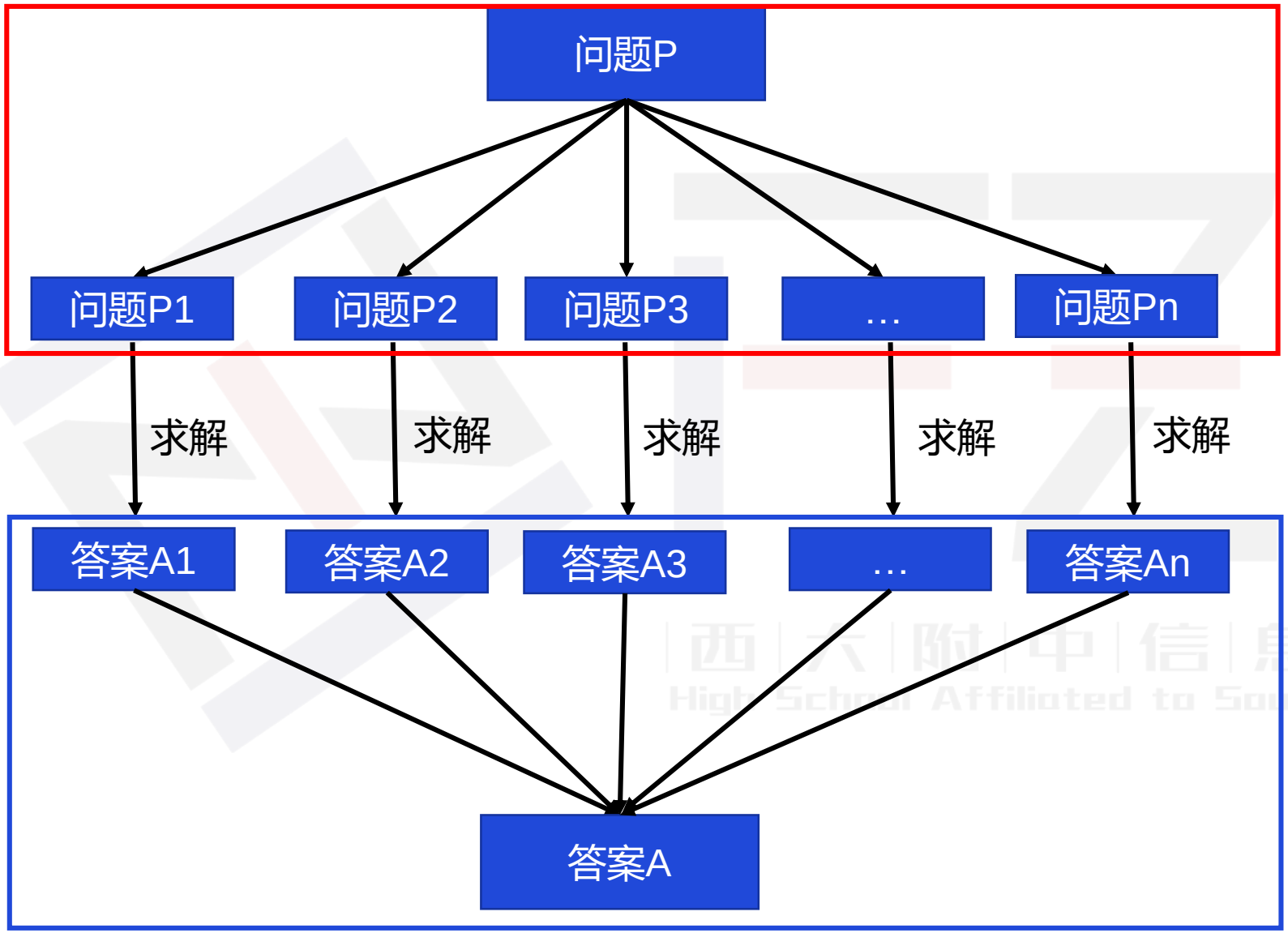
在求解一个规模为n，而n的取值又很大的问题时，直接求解往往非常困难。这时，把问题分割成一些规模较小的相同问题，以便各个击破，分而治之。

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。

两种基于分治的排序

排序算法

排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	排序方式	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	In-place	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	In-place	稳定
希尔排序	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(1)$	In-place	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Out-place	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	In-place	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	In-place	不稳定
计数排序	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	Out-place	稳定
桶排序	$O(n + k)$	$O(n + k)$	$O(n^2)$	$O(n + k)$	Out-place	稳定
基数排序	$O(n \times k)$	$O(n \times k)$	$O(n \times k)$	$O(n + k)$	Out-place	稳定

当数据n为10^5

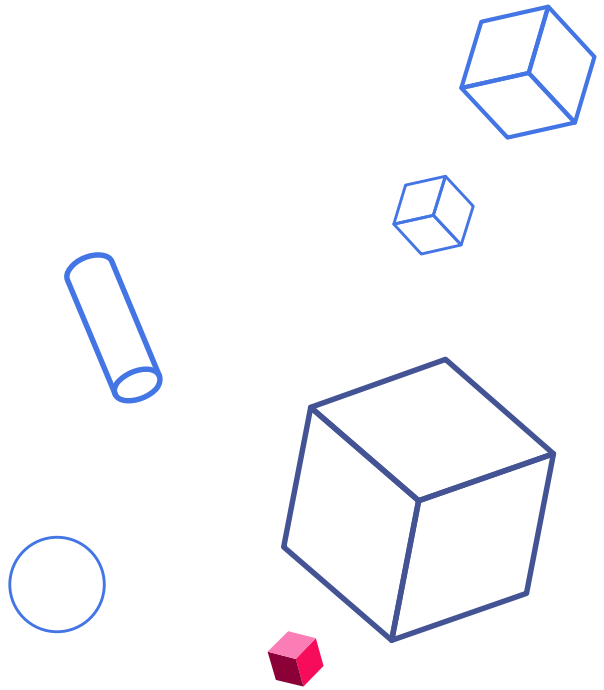
据不完全测试，$O(n^2)$耗时约是O(nlogn)的6000倍

如果O(nlogn)算法一组数据排序需要一天， $O(n^2)$则需要6000天，约16.5年

一个好的算法能高效地解决问题，这也是学习和设计算法的意义所在

FZOI
西大附中信息学竞赛
High School Affiliated to Southwest University

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



归并排序

归并排序

基本思想：建立在分治思想和归并操作上的一种排序方法

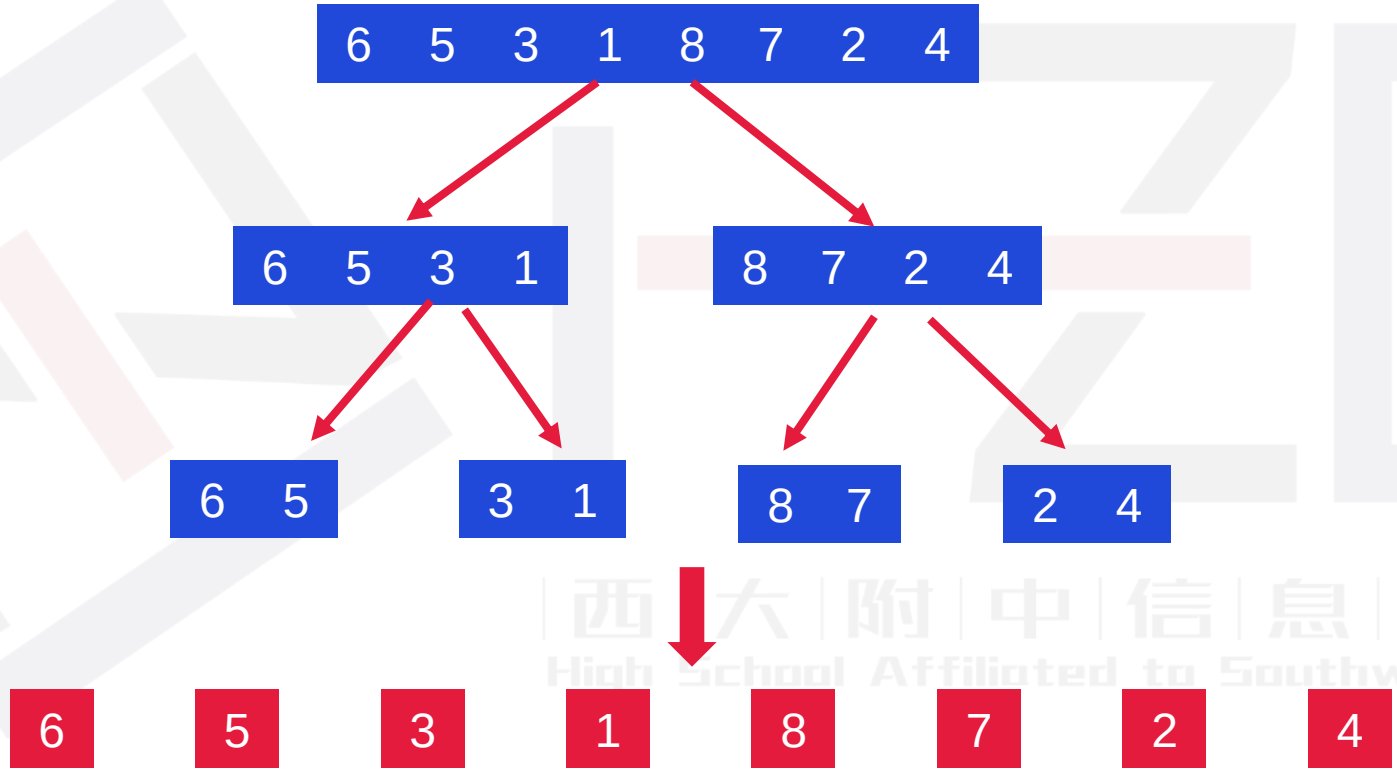
算法过程：

“分解”——将序列每次对半划分（递归实现）

“合并”——将划分后的序列段两两合并后排序

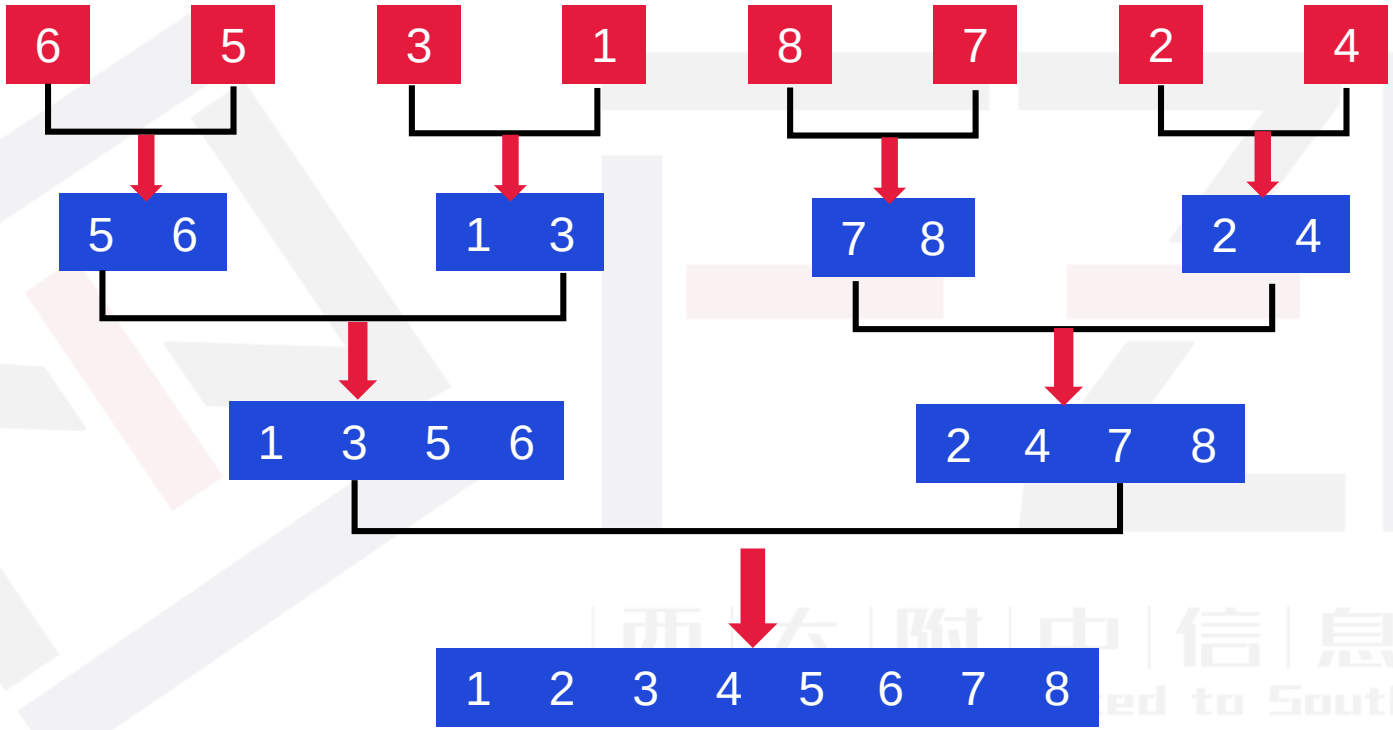
排序过程

“分解”——将序列每次折半划分（递归实现）



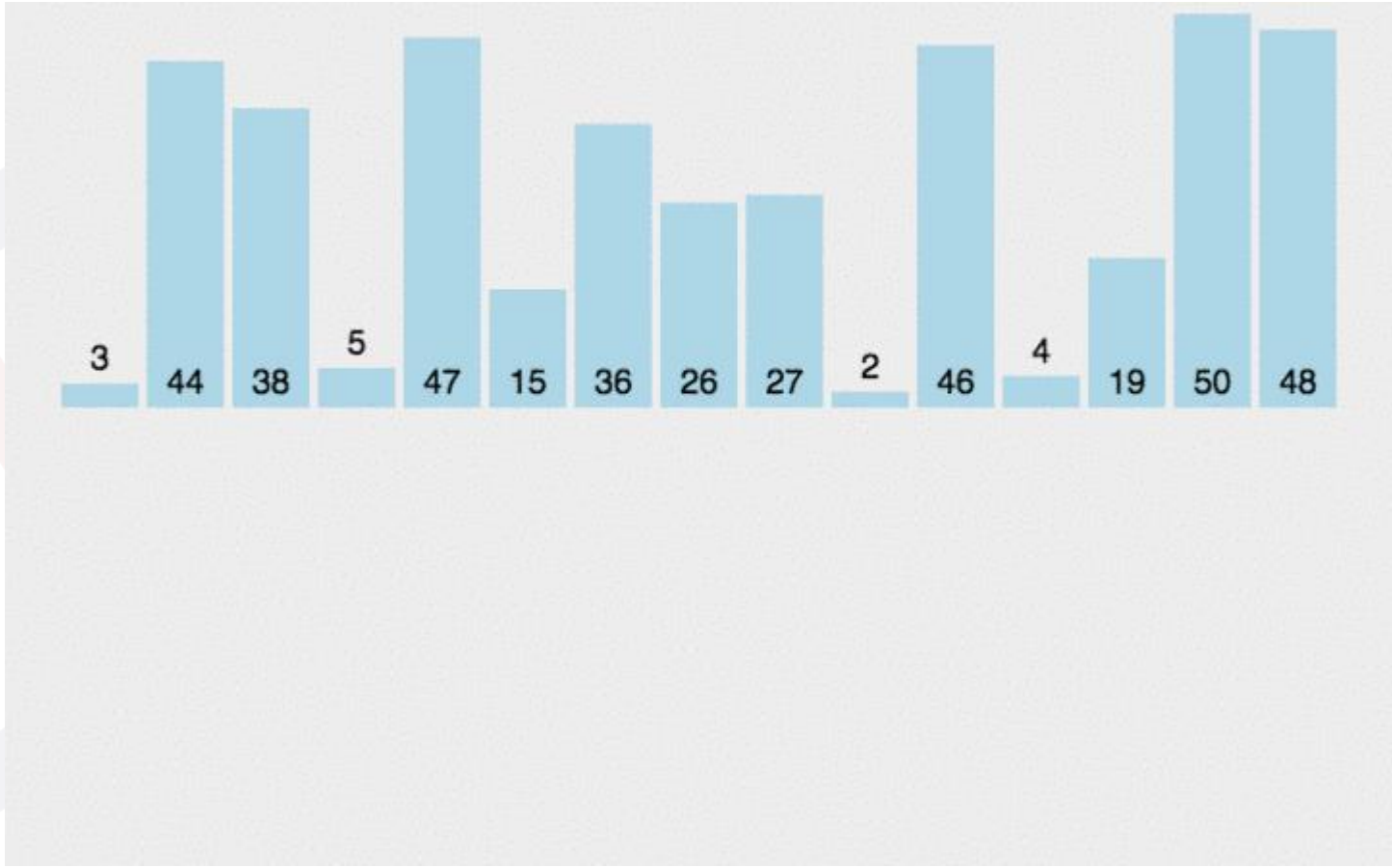
排序过程

“合并”——将划分后的序列段两两合并后排序



思考：合并操作能否在原数组的基础上进行？
不能，所以需要新开一个数组r[ ]

排序动画



归并排序参考代码

算法过程：

1.分解左右区间

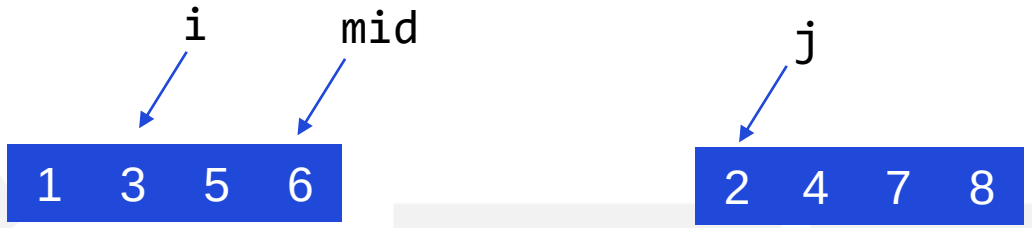
2. 借助r数组，对左右区间进行分别排序

3.合并排序后的结果

```
void Msort(int left,int right){
          int mid=(left+right)>>1;
          if(left==right) return ;
          Msort(left,mid);    //分解左右区间
          Msort(mid+1,right);
          int i=left,j=mid+1,k=left;   //i,j为左右区间的起始下标，k为r数组的下标
          while(i<=mid&&j<=right){
                    if(a[i]<=a[j]){
                      r[k]=a[i];
                      k++,i++;
                    }
                    else{
                      r[k]=a[j];
                      k++,j++;

                    }
          }
          while(i<=mid){   //复制左右剩余数字
                    r[k]=a[i];
                    k++,i++;
          }
          while(j<=right){
                    r[k]=a[j];
                    k++,j++;
          }
          for(int i=left;i<=right;i++) a[i]=r[i]; //r排序后的结果存放到a
}
```

归并排序应用：求逆序对




```
if(a[i] > a[j])
```

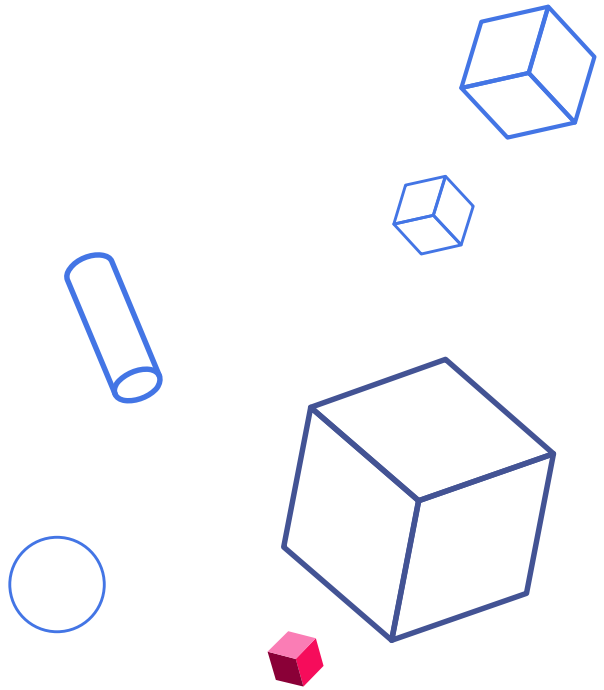
那么j和区间[i,mid]每一个点都形成逆序对

这一次合并，产生的逆序对的数目是?

累加每次的逆序对数目：`sum += mid - i + 1;`

FZOI
西大附中信息学竞赛
High School Affiliated to Southwest University

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



快速排序

快速排序

基本思想：快速排序可以看做是冒泡排序利用分治进行的优化

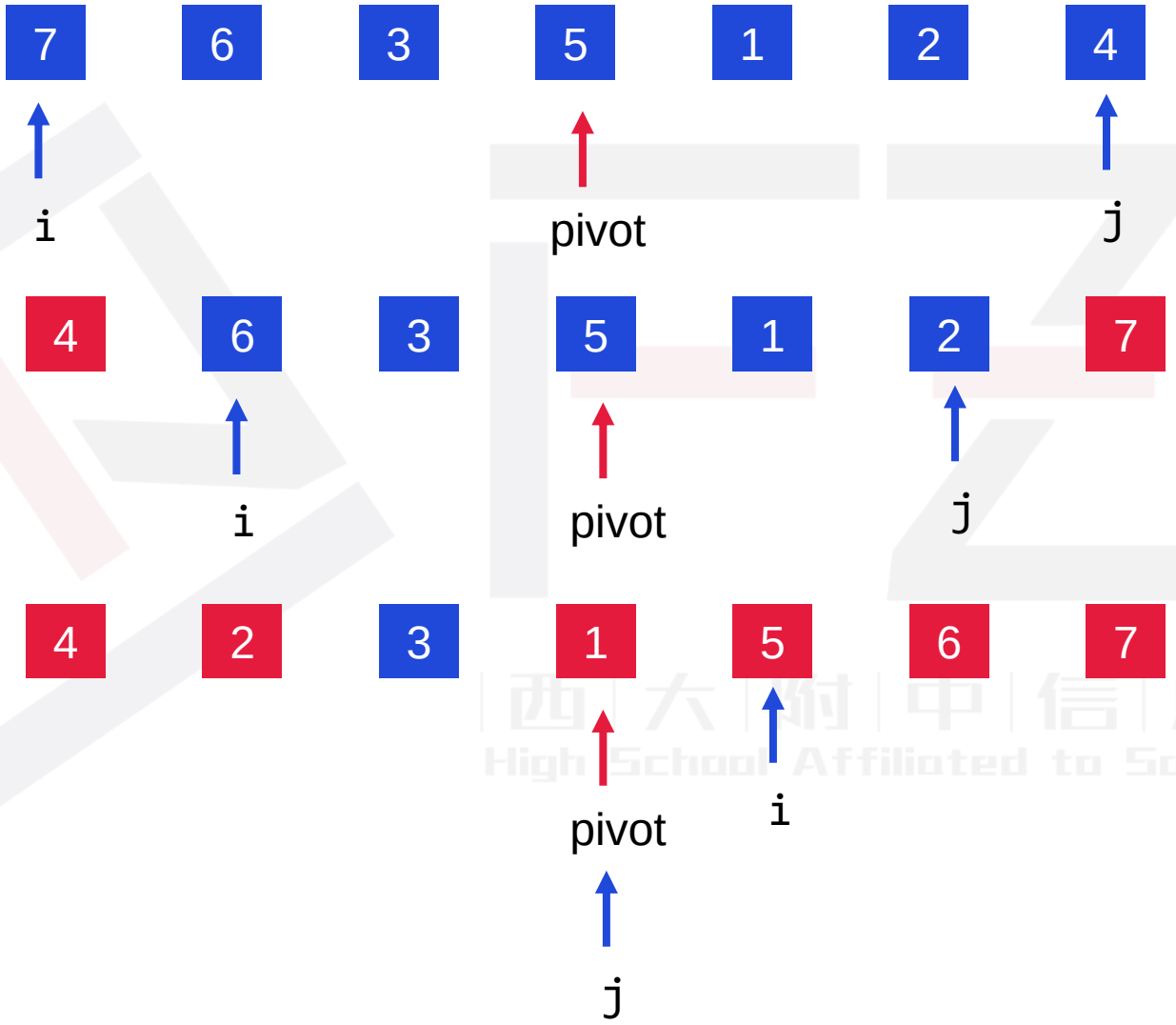
算法过程：

- 从数列中挑出一个元素，称为 “基准” （pivot)
- 将所有元素比基准值小的摆放在基准左面，所有元素比基准值大的摆在基准的右面
- 递归地对左右子数列按此方法继续排序。

实现方式有：挖坑法、交换法

排序过程

基准取中法

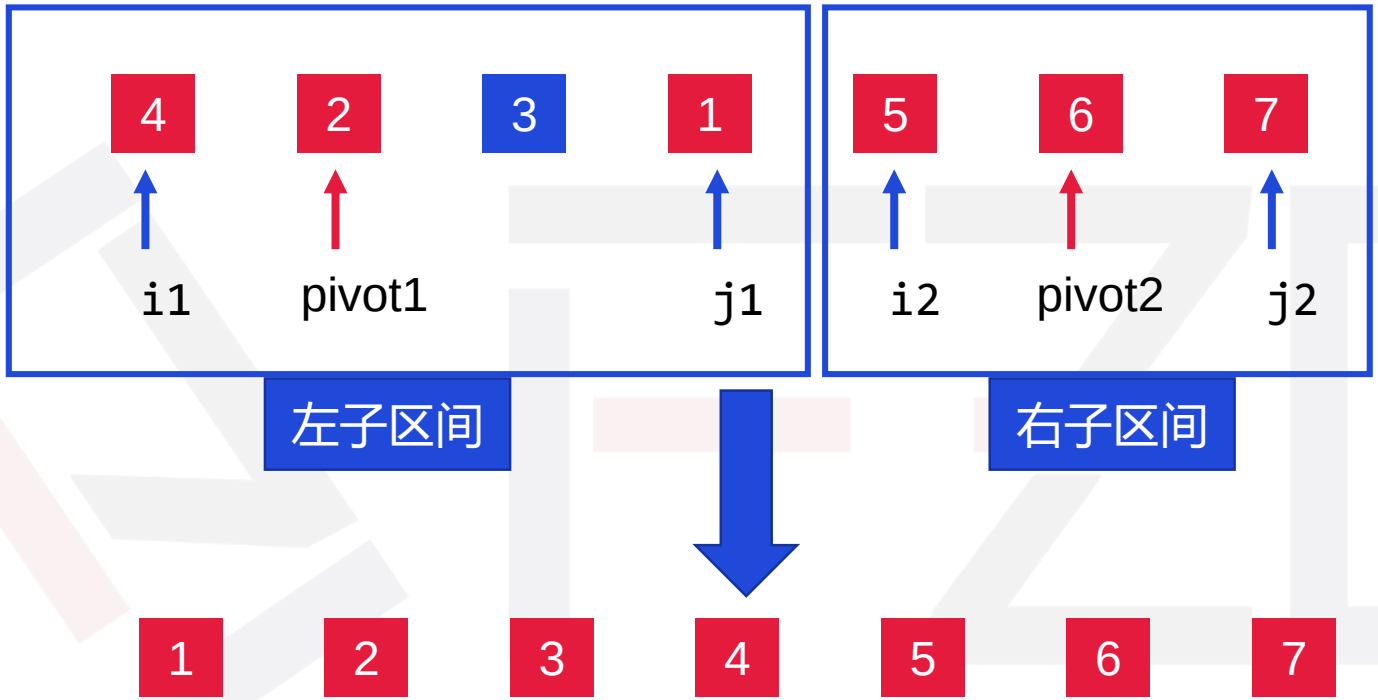


i从左往右找比基准大的数
j从右往左找比基准小的数

找到，交换a[i]和a[j]
然后继续找
i++,j--

直到j>i,停止

排序过程



继续对左右子区间排序，直到全部区间排好为止

快排的要点

快排的核心在于 **“基准”的选择**

一般的基准数会选择首尾，或者序列中间

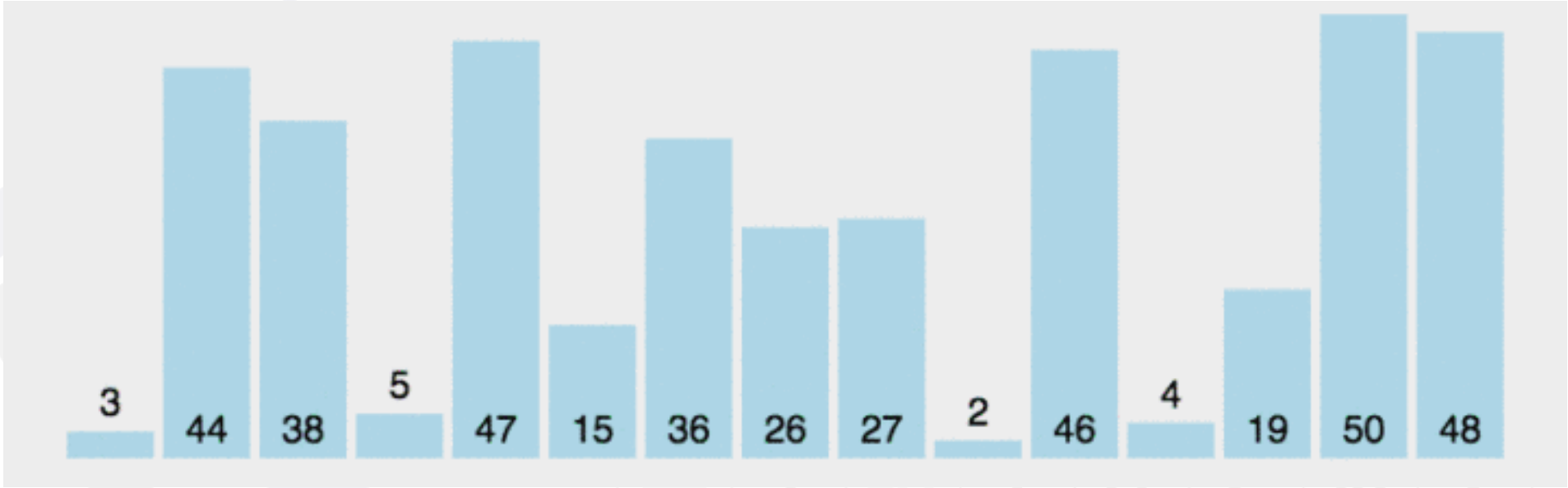
但对于这样的数列：

8	7	6	5	4	3	2	1

如果每一次都选择第一个数作为基准数，快速排序会退化为冒泡排序，最坏情况$O(n^2)$

一般情况下，选择中间是比较好的选择
最优的选择方式是每次都选择一个随机位置作为基准数

排序动画



快速排序参考代码


```
void Qsort(int left,int right){
        int mid=(left+right)>>1;
        int pivot=a[mid]; //取中间位置作为基准数
        int i=left,j=right;
        while(i<=j){
                while(a[i]<pivot) i++;    //找到交换的位置
                while(a[j]>pivot) j--;
                if(i<=j){  //交换
                        swap(a[i],a[j]);
                        i++,j--;
                }
        }
        if(left<j) Qsort(left,j); //递归快排左右子区间
        if(i<right) Qsort(i,right);

}
```

拓展了解：sort()

sort()融合了插入排序、快速排序、堆排序三种排序算法

能根据数据规模n，数据排序递归的深度等，选择不同的排序算法

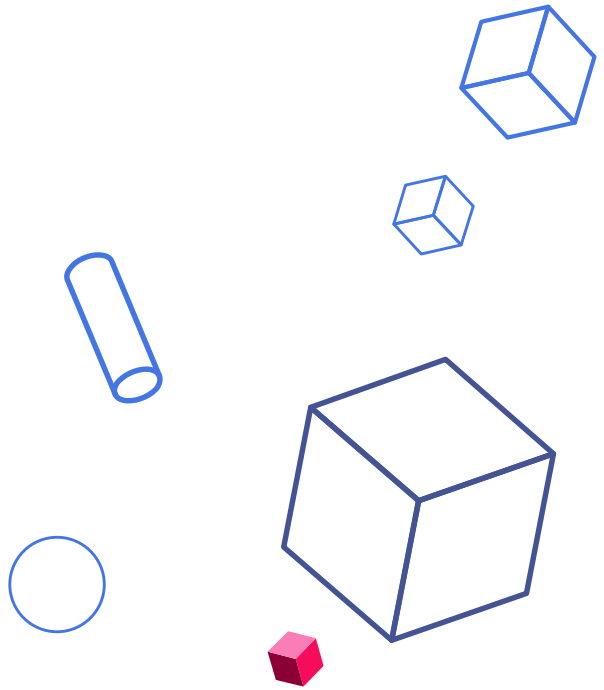
平均情况下时间复杂度：O(nlogn)

Thanks

For Your Watching

FZOI
西大附中信息学竞赛
High School Affiliated to Southwest University

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

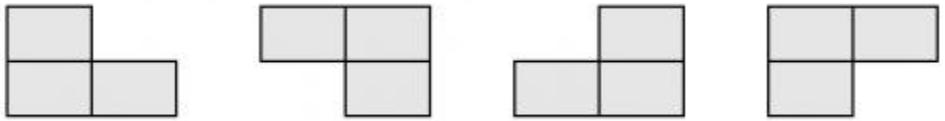


习题讲解

地毯填补

题目描述

相传在一个古老的阿拉伯国家里，有一座宫殿。宫殿里有个四四方方的格子迷宫，国王选择驸马的方法非常特殊，也非常简单：公主就站在其中一个方格子上，只要谁能用地毯将除公主站立的地方外的所有地方盖上，美丽漂亮聪慧的公主就是他的人了。公主这一个方格不能用地毯盖住，毯子的形状有所规定，只能有四种选择(如图)：



1　　2　　3　　4

并且每一方格只能用一层地毯，迷宫的大小为2的k次方见方的方形。

输入

输入文件共2行。

第一行：k，即给定被填补迷宫的大小为2k（0<k≤10）；

第二行：x y，即给出公主所在方格的坐标（x为行坐标，y为列坐标），x和y之间有一个空格隔开。

输出

将迷宫填补完整的方案：每一补（行）为 x y c (x，y为毯子拐角的行坐标和列坐标，c为使用毯子的形状，具体见上面的图，毯子形状分别用1、2、3、4表示，x、y、c之间用一个空格隔开)。

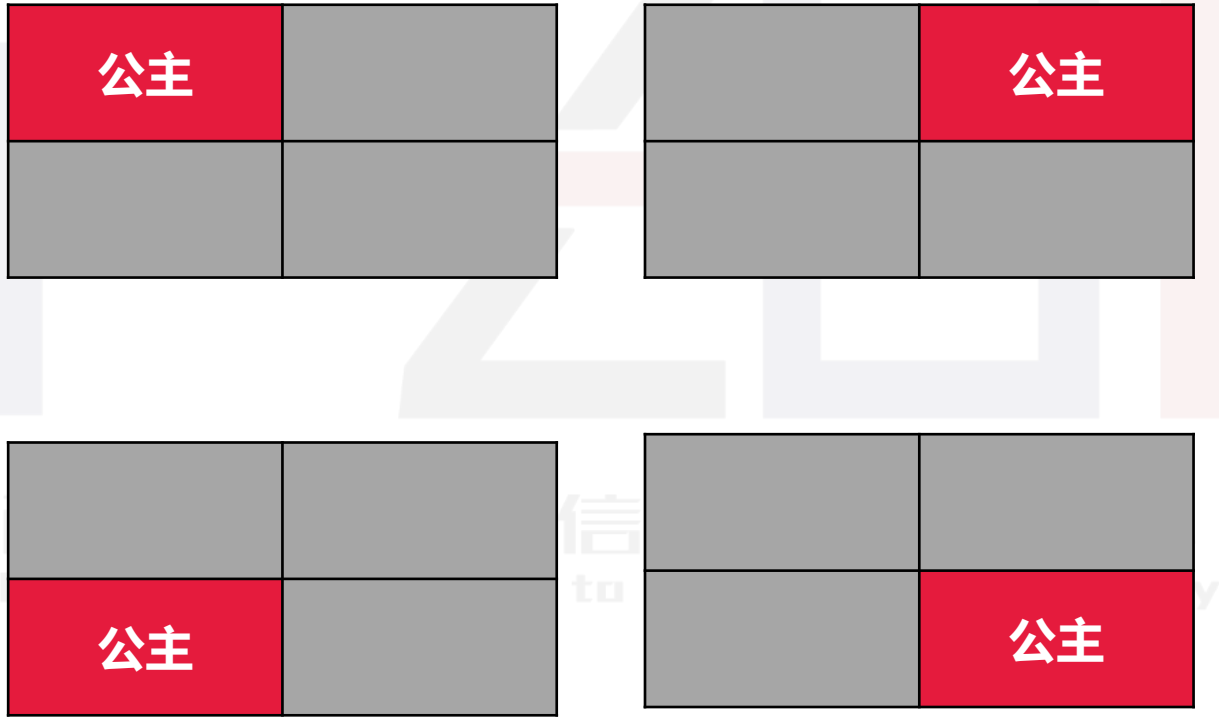
样例输入	样例输出
3 3 3	5 5 3 2 2 4 1 1 4 1 4 2 4 1 1 4 4 3 2 7 2 1 5 4 1 8 2 3 6 2 4 8 3 7 2 1 5 1 4 6 3 1 8 1 1 8 4 3 7 7 3 6 6 3 5 8 2 8 5 1 8 8 3

分析

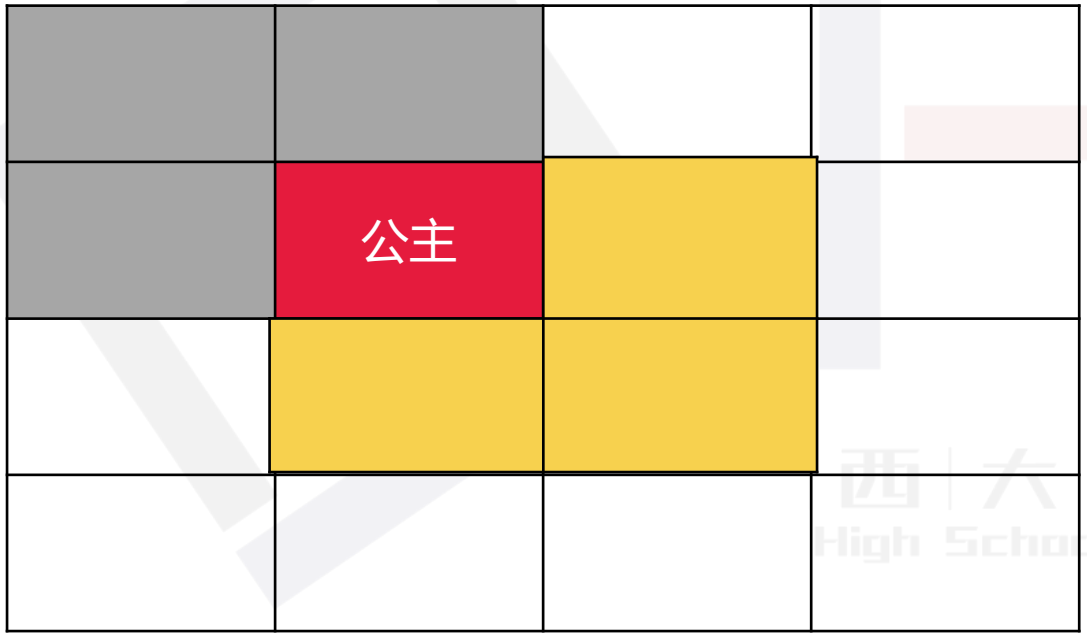
这是一道典型的棋盘覆盖问题

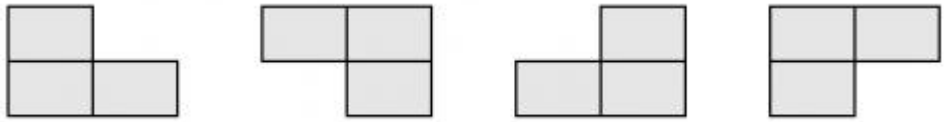
先从已知的小问题入手：

K=1



K=2



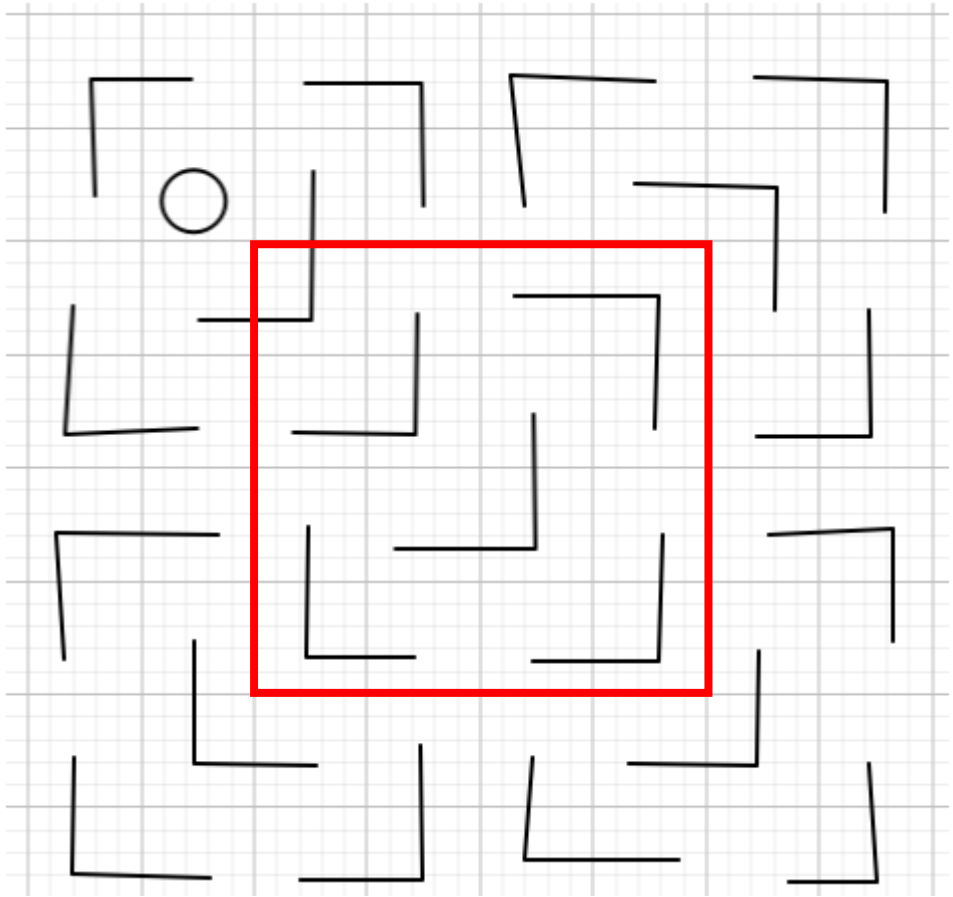


Q：如何给每个小的2*2方块制造一个障碍？

什么位置摆？ 中间的位置

摆几号图形？ 摆一个3号图形

K=3



在居中的矩形摆上地毯，制造障碍

目的：让2^k*2^k的矩形能够分解为4个$2^{k-1}*2^{k-1}$

最小的情况：2*2的矩形

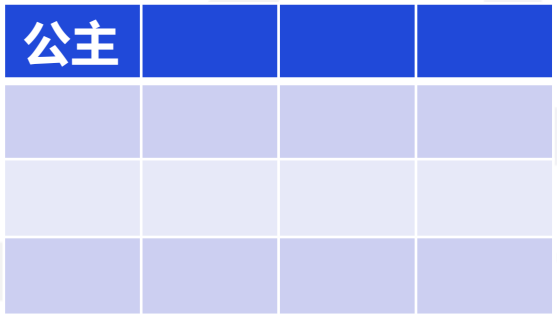
https://blog.csdn.net/Starry_Sky_Dream/article/details/110683085

https://www.cnblogs.com/crab-in-the-northeast/p/luogu-p1228.html

参考代码


```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
void solve(ll x, ll y, ll a, ll b, ll l) { //x,y为障碍点坐标;a,b为左上角点,l为边长
    if (l == 1)
        return;
    if ((x - a) <= (l / 2 - 1) && (y - b) <= (l / 2 - 1)) {  //障碍在左上角
        printf("%lld %lld 3\n", (a + l / 2), (b + l / 2));
        solve(x, y, a, b, l / 2);
        solve(a, b + l / 2, a, b + l / 2, l / 2);
        solve(a + l / 2+ l / 2 - 1, b + l / 2 - 1, a + l / 2, b, l / 2);
        solve(a + l / 2, b + l / 2, a + l / 2, b + l / 2, l / 2);
    }
    else if ((x - a) <= (l / 2 - 1) && (y - b) > (l / 2 - 1)) {  //障碍在右上角
        printf("%lld %lld 1\n", (a + l / 2), (b + l / 2 - 1));
        solve(a + l / 2 - 1, b + l / 2 - 1, a, b, l / 2);
        solve(x, y, a, b + l / 2, l / 2);
        solve(a + l / 2, b + l / 2 - 1, a + l / 2, b, l / 2);
        solve(a + l / 2, b + l / 2, a + l / 2, b + l / 2, l / 2);
    }
    else if ((x - a) > (l / 2 - 1) && (y - b) <= (l / 2 - 1)) {  //障碍在左下角
        printf("%lld %lld 2\n", (a + l / 2 - 1), (b + l / 2));
        solve(a + l / 2 - 1, b + l / 2 - 1, a, b, l / 2);
        solve(a + l / 2 - 1, b + l / 2, a, b + l / 2, l / 2);
        solve(x, y, a + l / 2, b, l / 2);
        solve(a + l / 2, b + l / 2, a + l / 2, b + l / 2, l / 2);
    }
    else {                                                  //障碍在右下角
        printf("%lld %lld 4\n", (a + l / 2 - 1), (b + l / 2 - 1));
        solve(a + l / 2 - 1, b + l / 2 - 1, a, b, l / 2);
        solve(a + l / 2 - 1, b + l / 2, a, b + l / 2, l / 2);
        solve(a + l / 2, b + l / 2 - 1, a + l / 2, b, l / 2);
        solve(x, y, a + l / 2, b + l / 2, l / 2);
    }
}
```

```
int main()
{
    ll x, y;
    int k;
    ll len; //边长
    scanf("%d %lld %lld", &k, &x, &y);
    len = (ll)1<<k;
    solve(x, y, 1, 1, len);
    return 0;
}
```



总结

分治是一种解题思路，可以解决类似于地毯填补、南蛮图腾这样的自相似问题

归并排序和快速排序是分治思想的经典应用，也非常重要

在后面的算法中，分治也会有所体现