1.递归与分治

递归

关键点：边界条件+递归方程

排列问题

- 问题描述：设计递归算法生成n个元素的全排列，Perm(X)为X个元素的全排列，rPerm(X)为在全排列Perm(X)的每一个排列前加前缀r得到的排列
- 伪代码展示：

```
template <class Type>
     void Perm(Type list[], int k, int m){
     //求列表从第k到第m个元素的全排列
     if(k==m){
     //列表只剩一个元素时，全排列只有一种，即自身
         cout<<list[k]<<endl;
     }
     for (int i=k; i<=m; i++){
         Swap(list[k], list[i]);//从列表中选出元素作为前缀
         Perm(list[], k+1, m);//将列表剩余的元素进行全排列
         Swap(list[k], list[i]);//将元素换回，防止错选和漏选
     }
 }
```

整数划分

- 问题描述：将正整数n表示成一系列正整数之和：n=n1+n2+...+nk 且 n1≥n2≥...≥nk，求正整数n的不同划分个数
- 解题思路：递归计算最大加数为x的划分方式+最大加数不大于x-1的划分方式
- 伪代码展示：

```
template <class Type>
int nSplit(int n,int m){//正整数你，最大加数为m
  if (n==m==1) return 1;
  else if (m>n) return nSplit(n,n);//最大加数≤需要划分的正整数
  else if (m==n) return 1+nSplit(n,m-1);//n划分为n只有一种方法
  else{
       return nSplit(n-m,m)+nSplit(n,m-1);
  }
}
```

汉诺塔问题

- 问题描述：设有A、B、C三个塔座，在A上有n个圆盘，需要将A上的圆盘移动到B上且遵守以下规则：
  - 每次只能移动一个圆盘
  - 大圆盘不能叠加在小圆盘上
- 解题思路：借助辅助塔座进行移动
- 伪代码展示

```
template <class Type>
void hanoi(int n, int a, int b, int c){
//a上n个圆盘借助c移动到b上
    if(n>0){
        hanoi(n-1,a,c,b);//a上n-1个圆盘借助b移动到c上
        move(a,b);//将a上遗留的圆盘移动到目标b上
        hanoi(n-1,c,b,a);//c上n-1个圆盘借助a移动到b上
    }
}
```

非空子集划分

- 问题描述：将含有n个元素的集合，划分成多个非空子集，求划分的个数
- 解题思路：将需要划分的第n个元素独立讨论，分为两种情况：这个元素占一个单独的集合，这个元素加到前面n-1个元素已划分的集合中
- 伪代码展示：

```
template <class Type>
int S(n,k){
//n为元素个数，k为集合个数
    //边界条件
    if (n==k || k==1) return 1;
    else if (n==0 || k==0 || n<=k) return 0;
    else {
        return S(n-1,k-1)+k*S(n-1,k);
    }
}
```

分治

将一个大问题分解成若干个规模较小的相同问题，相互独立，可合并

计算时间复杂性

主定理

- $T(n)=\begin{cases}O(1) & n=1\\ kT(\frac{n}{m})+f(n) & n>1//有k个子问题，每个子问题的规模是\frac{n}{m}\end{cases}$
- $f(n)$与$\log_m k$比较阶数等级$\begin{cases}跟大的走\\ 若相等，则O(n^{\log_m k}\log n)\end{cases}$
- 无法使用主定理时，可用递归计算时间复杂性
- 常用$O(n)$大小比较
  $O(1)<O(logn)<O(n)<O(nlogn)<O(n^2)<O(2^n)<O(n!)<O(n^n)$

二分搜索算法

金块问题

- 问题描述：有n个金块（n为2的幂），用一台比较重量的仪器，找出最轻和最重的金块，求尽可能少的比较次数
- $T(n)=\begin{cases}0 & n=1\\ 1 & n=2\\ 2T(\frac{n}{2})+2 & n>2\end{cases}$
- 时间复杂度：$O(logn)$（主定理）

二分查找

- 伪代码展示：

```
template <class Type>
int BinarySearch(Type a[], int k, int n){//a[]中有n个元素，查找k
    int left=0;
    int right=n-1;
    while(left<=right){
        int mid=(left]+right)/2;//二分
        if(x==a[mid]) return middle;
        else if(x<a[mid]) right=mid-1;//改变查找范围
        else left=mid+1;
    }
    return -1;
}
```

- 时间复杂性：$T(n)=\begin{cases}1 & n=1\\ T(\frac{n}{2})+1 & n>1\end{cases}$
- 时间复杂度：$O(logn)$

循环赛日程表

- 问题描述：
  - 每位选手要跟其他n-1个选手比赛一次
  - 每个选手一天只能比赛一次
  - 循环赛一共进行n-1天
- **选手人数n可能为偶数，也可能为奇数-->假设1人为轮空，正常按照偶数排列**

- 方法1：不断缩小问题规模直至2个人
- 方法2：半三角画表法（类似数独填入）
- 方法3：多边形法（两两配对，n为偶数时n-1条边，选出一点在中间与其中一个顶角配对，其余相对点配对）

棋盘覆盖

- 问题描述：用4种L型骨牌覆盖2^k x 2^k 个方格组成的棋盘，且有1残缺方格
- 时间复杂性：
  $T(n)=\begin{cases}O(1) & k=1//覆盖他需要常数时间\\ T(k-1)+O(1) & k>1//测试哪个棋盘残缺以及形成3个残缺子棋盘时间\end{cases}$
- 时间复杂度：$O(4^k)$ （递归法）

大整数的乘法

- 问题描述：X和Y都是n位的二进制数，计算XY

-
- 减少做乘法的次数达到优化效果
- $$XY=(A2^{\frac{n}{2}}+B)\times(C2^{\frac{n}{2}}+D)=AC2^n+(AD+BC)2^{\frac{n}{2}}+BD$$
- $$XY=AC2^n+((A-B)(D-C)+AC+BD)2^{\frac{n}{2}}+BD$$
- 时间复杂性：$T(n)=\begin{cases}O(1) & n=1\\ 3T(\frac{n}{2})+O(n) & n>1\end{cases}$
- 时间复杂度：$O(n^{log3})$ （主定理）
- 伪代码展示

```
function Mult(X, Y, n){
    int S=sign(X)*sign(Y);//存储XY计算结果
    X=abs(X);
    Y=abs(Y);
    if (n==1) then
        if (X==1 & Y==1) then return S;
        else return 0;
    else{
        A=X的左边n/2位;
        B=X的右边n/2位;
```

```
        C=Y的左边n/2位;
        D=Y的右边n/2位;
        m1=Mult(A,C,n/2);
        m2=Mult(B,D,n/2);
        m3=Mult(A-B, D-C, n/2);
        S=S*(m1*2^n+(m3+m1+m2)*2^(n/2)+m2);
        return S;
    }
}
```

Strassen矩阵乘法（不断优化中）

- 时间复杂性：$T(n)=\begin{cases}O(1) & n=1\\ 7T(\frac{n}{2})+O(n^2) & n>1\end{cases}$
- 时间复杂度：$O(n^{log7})$

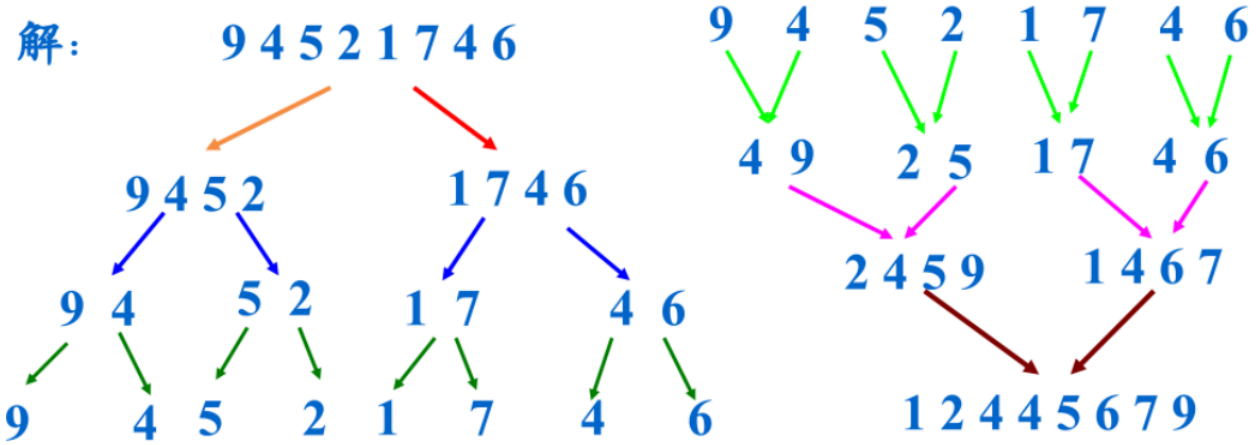
最大子段和

- 问题描述：找数组里一个连续片段，让这个片段的和最大
- 解题思路：走到当前位置，看下一个数$\begin{cases}\text{和为负数，，丢掉，从自己重新开始}\\ \text{和为正数，加上，让自己更大}\end{cases}$
- 伪代码展示：

```
cur_sum=max(num[i],num[i]+cur_sum);//当前子数组和
max_sum=max(cur_sum,max_sum);//全局最大和
```

- 时间复杂性：$T(n)=\begin{cases}O(1) & n\le 2\\ 2T(\frac{n}{2})+O(n) & n\ge 3//\text{分成两段，从中间向两边找}\end{cases}$
- 时间复杂度：$O(nlogn)$

合并排序



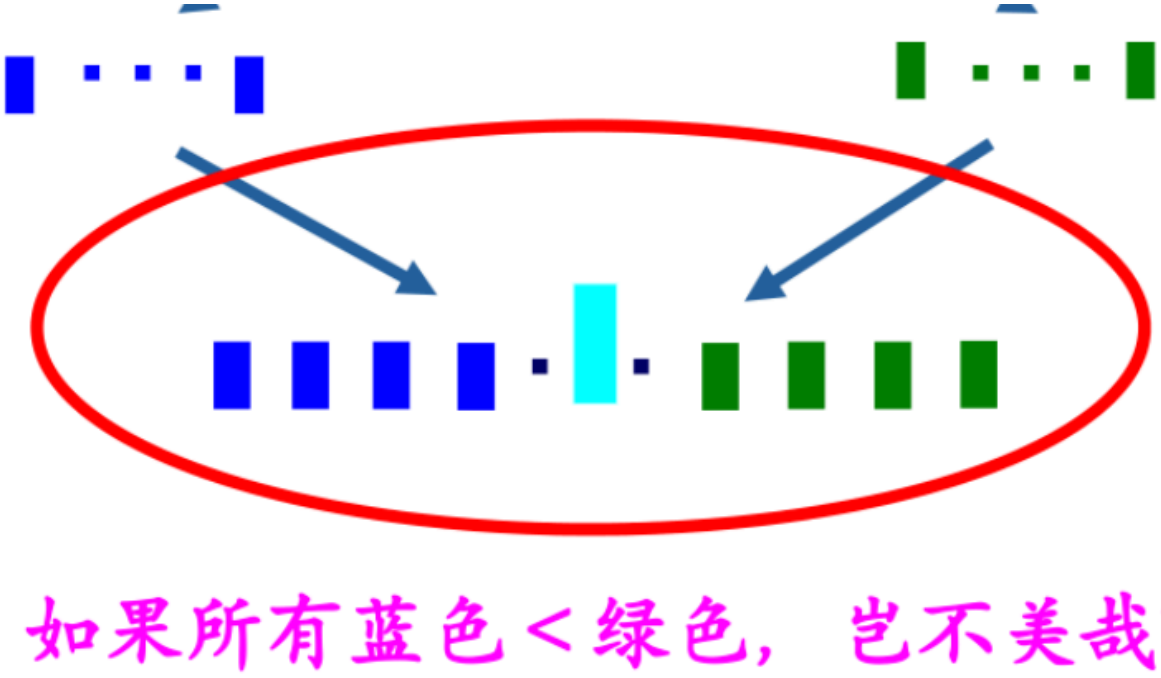
- 伪代码展示：

```
template <class Type>
void MergeSort(Type a[],int left, int right){
```

```
    if(left<right){
        int mid=(left+right)/2;
        MergeSort(a,left,mid);
        MergeSort(a,mid+1,right);
        Merge(a,b,left,mid,right);//从a合并到数组b
        Copy(a,b,left,right);//将数组b复制回数组a
    }
}
```

- 时间复杂性：$T(n)=\begin{cases}O(1) & n=1\\ 2T(\frac{n}{2})+O(n) & n>1\end{cases}$
- 时间复杂度：$O(nlogn)$

快速排序



- 性能取决于划分的对称性
- 伪代码展示：

```
template <class Type>
void QuickSort(Type a[],int left, int right){
    if(left<right){
        int mid=partation(a,left,right);
        QuickSort(a,left,mid-1);
        QuickSort(a,mid+1,right);
    }
}
int partation(Type a[],int left,int right){
    int i=left,j=right+1;
    Type x=a[left];
    while(true){
    //小的放左边，大的放右边
```

```
        while(a[++i]<x &&i<right);
        while(a[--j]>x);
        if(i>=j) break;
        swap(a[i],a[j]);
        a[left]=a[j];
        a[j]=x;
        return j;
    }
}
```

- 时间复杂性：
- $T_{max}=\begin{cases}O(1) & n=1\\ T(n-1)+O(n) & n>1\end{cases}$
- $T_{min}=\begin{cases}O(1) & n=1\\ 2T(\frac{n}{2})+O(n) & n>1\end{cases}$

线性时间选择

- 问题描述：找出这n个元素（无序排列）中第k小的数
- 性能取决于基准点的选择，基准点最好选择中位数
- 伪代码展示

```
template <class Type>
Type RandomizedSelect(Type a[],int left,int right,int k){
    if(left==right) return a[left];
    int i=RandomPartation(a,left,tight);
    int j=i-left+1;
    if(k>j) return RandomizedSelect(a,i+1,right,k-j);
    else return RandomizedSelect(a,left,i,k);
}
```

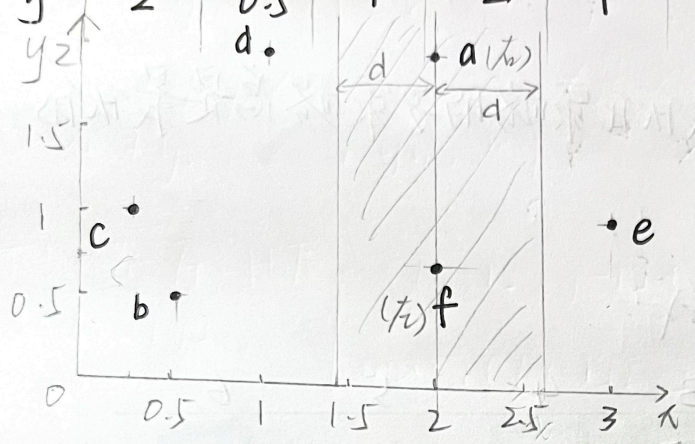
- 时间复杂性：
- $T_{max}=\begin{cases}O(1) & n=1\\ T(n-1)+O(n) & n>1\end{cases}$
- $T_{min}=\begin{cases}O(1) & n=1\\ T(\frac{n}{2})+O(n) & n>1\end{cases}$

最接近点对

- 给定平面上n个点，找出其中一对点，使得在n个点组成的所有点对中，该点对间的距离最小

点	a	b	c	d	e	f
x	2	0.5	0.25	1	3	2
y	2	0.5	1	2	1	0.7

中位线上有2个点时，将两点均分到左右2个区域



左：$cd=\frac{5}{4}=1.25$

$cb=\cdots=\frac{\sqrt{5}}{4}$

$df=\cdots$

右：~~ef~~

$ae=\sqrt{2}\approx1.4$

x坐标的中位数：2，将a点分到右侧，f点分到左侧

令 $d=\min\{|p_1-p_2|,|q_1-q_2|\}$，即单独两侧最短距离

$\Rightarrow d=cb=\frac{\sqrt{5}}{4}\approx0.56$

由图可知，落在中间区域 $[L-d,L+d]$ 中仅有a,f两点

$\because af=2-0.7=1.3>d=cb=\frac{\sqrt{5}}{4}$

$\therefore$ 最接近点对为(b,c)