分治法的应用

最接近点对问题

最接近点对问题

- ✓ 在应用中，常用诸如点、圆等简单的几何对象代表现实世界中的实体。在涉及这些几何对象的问题中，常需要了解其邻域中其他几何对象的信息。
- ✓ 最接近点对问题的提法是：

给定平面上n个点，找其中的一对点，使得在n个点的所有点对中，该点对的距离最小。

- ✓ 严格地说，最接近点对可能多于1对。为了简单起见，这里只限于找其中的一对。

最接近点对问题

问题提出：

◆在**直线上**的n个点中，找出其中的一对点，使得在n个点组成的所有点对中，该点对的距离最小！

思路1:
只要将每一点与其他n-1个点距离算出，找出达到最小距离的两点即可。

效率:
$O(n^2)$

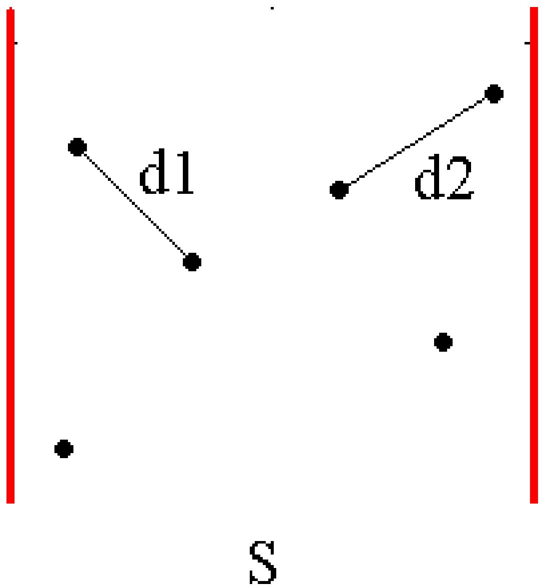
思路2:
在直线上，最近点对问题可以通过1次排序和1维扫描来解决。

效率:
$O(nlogn)$

最接近点对问题

问题提出：

◆在二维平面上的n个点中，找出其中的一对点，使得在n个点组成的所有点对中，该点对的距离最小！



思路2:
在直线上，最近点对问题可以通过1次排序和1维扫描来解决。

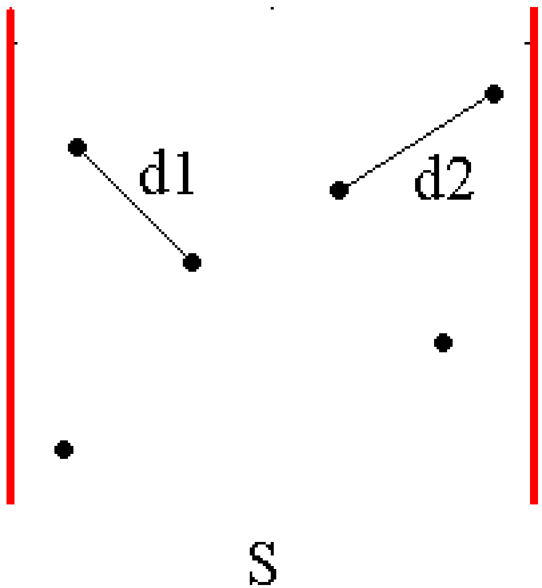
在二维空间中，这个思路是行不通的。

体会分治的策略： **缩小问题的规模**

最接近点对问题

问题提出：

◆在二维平面上的n个点中，找出其中的一对点，使得在n个点组成的所有点对中，该点对的距离最小！



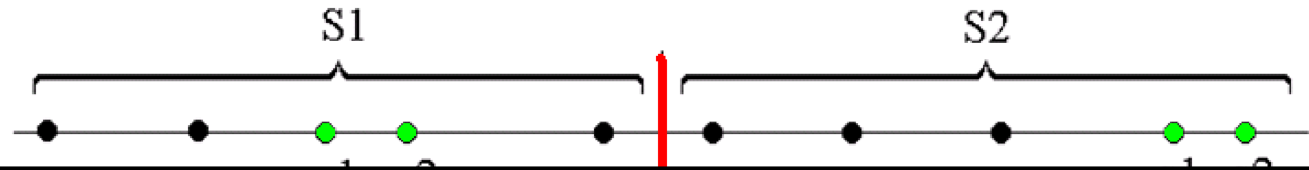
分治法的思路:

① 划分成子规模点集
② 找到极值情况
③ 合并

④提高效率（平衡子问题）

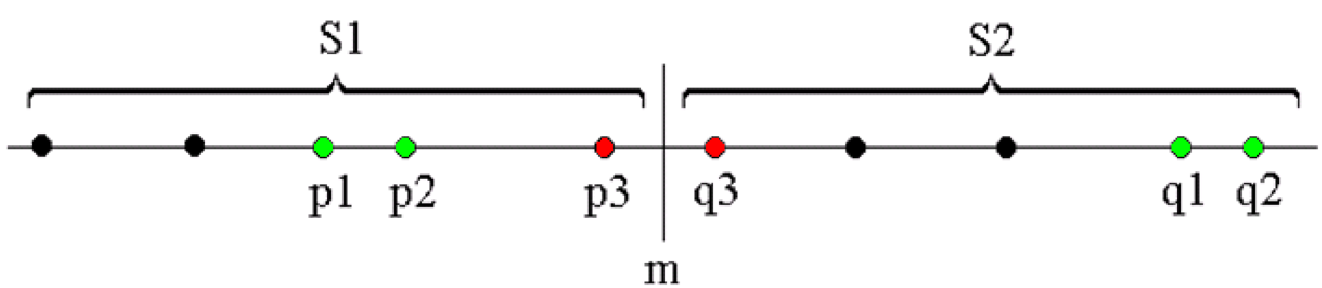
一维：最接近点对问题

- 此时，S中的n个点退化为x轴上的n个实数 x1,x2,…,xn。最接近点对即为这n个实数中相差最小的2个实数。

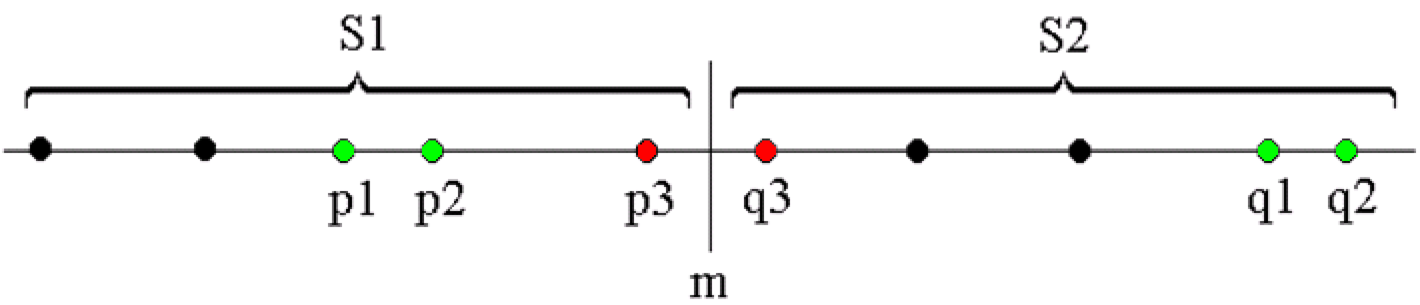


思考：此时找到的距离是全集S的最小距离吗?

- **假设我们用x轴上某个点m将S划分为2个子集S1和S2；**
- **递归地在S1和S2上找出其最接近点对{p1,p2}和{q1,q2}，并设d=min{|p1-p2|,|q1-q2|}，S中的最接近点对或者是{p1,p2}，或者是{q1,q2}**
- **合并**



如何找p3,q3?



◆如果S的最接近点对是{p3,q3}，即|p3-q3|<d，则p3和q3两者与m的距离不超过d，即**p3∈(m-d,m]，q3∈(m,m+d]。**

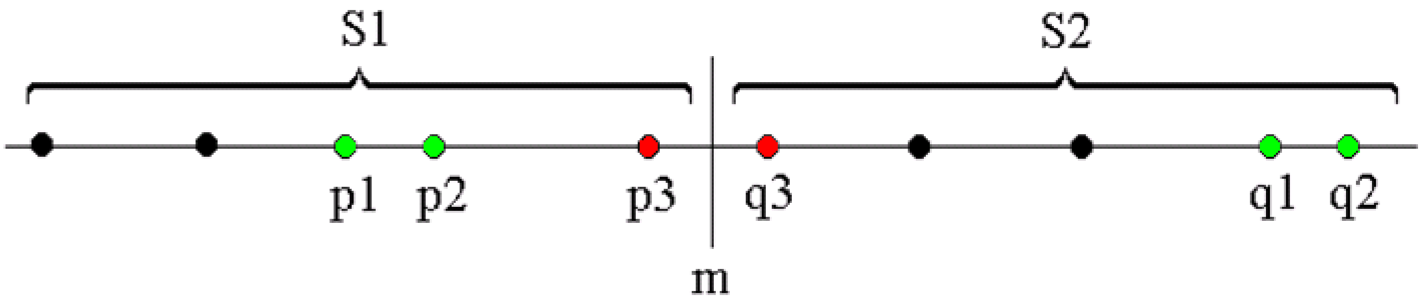
◆由于在S1中，每个长度为d的半闭区间至多包含一个点（否则必有两点距离小于d），并且m是S1和S2的分割点，因此(m-d,m]中至多包含S中的一个点。由图可以看出，**如果(m-d,m]中有S中的点，则p3就是S1中最大点。**

◆**同理，q3就是S2中的最小点。**

分治法的思路:
① 划分成子规模点集S1和S2
② 找到S1和S2的最小距离d。③ 合并S1和S2：并利用d在S1的(m-d,m]和S2的(m,m+d]中找最大点和最小点，即p3和q3。选出最小距离，完成合并

分治法的算法复杂性:
① 划分成子规模点集S1和S2
②我们用线性时间就能找到区间(m-d,m]和(m,m+d]中所有点，即p3和q3。从而我们用线性时间就可以将S1的解和S2的解合并成为S的解。

算法的最坏情况?



如何选择m?

$$m = \frac{\max(s) + \min(s)}{2}$$

最坏情况:

① 划分成子规模点集S1和S2

② S1的个数=1，S2的个数=n-1

效率:

$O(n^2)$

改进方案:

基于平衡子问题的思想，用S中各点坐标的中位数来作分割点。

效率:

=线性时间选择= O(n)

```
//一维
bool Cpairl( s, d )
{
        n = |s|;
        if (n<2) {d为无穷大; return false;}
        m = s中各个坐标的中位数;
        //构造S1 和 S2
        //S1 = { x<m} s2 = {x > m }
        Cpairl( s1, d1 );
        Cpairl( s2, d2 );
        p = max (s1);
        q = min (s2);
        d = min( d1, d2, q-p);
        return true;
}
```

一维情况下的算法复杂性

复杂度分析

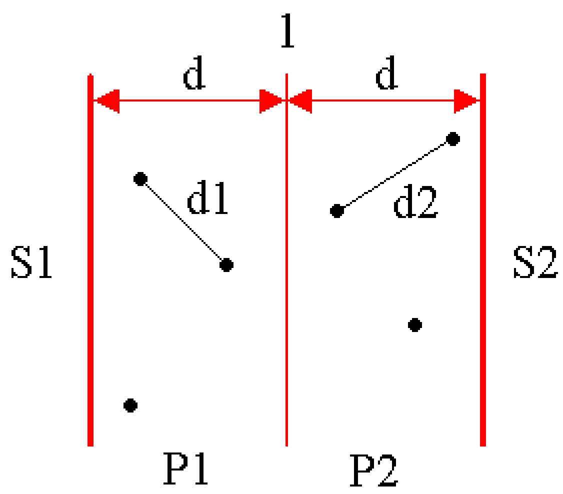
$$T(n)=\begin{cases} O(1) & n<4 \\ 2T(n/2)+O(n) & n\ge 4 \end{cases}$$

***T*(*n*) = *O*(*nlogn*)**

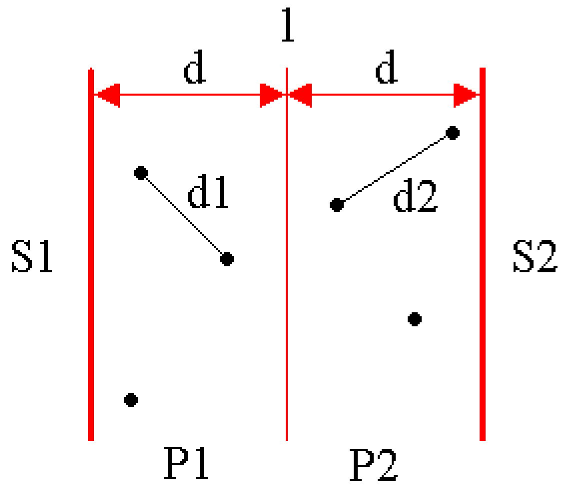
二维：最接近点对问题

- 下面来考虑二维的情形。

➢选取一垂直线**l : x = m**来作为分割直线。其中m为S中各点x坐标的中位数。由此将S分割为S1和S2。
➢递归地在S1和S2上找出其最小距离d1和d2，并设**d=min{d1,d2}**，S中的最接近点对或者是d，或者是某个{p,q}，其中p∈P1且q∈P2。
➢**合并：能否在线性时间内找到p,q，完成合并?**



算法的最坏情况?



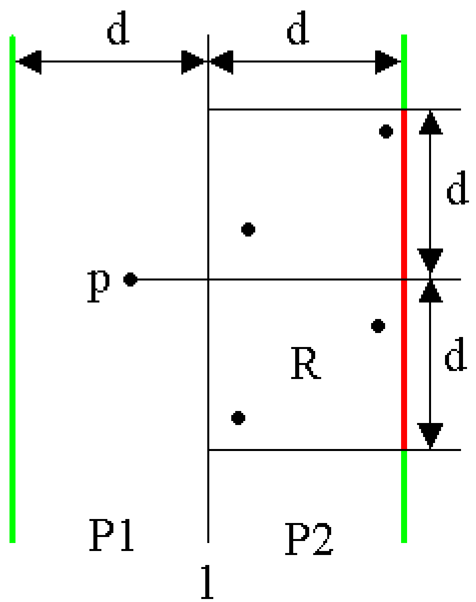
最坏情况:
① 设P1中的一个点；对应p，在P2中共有n/2个候选点；
② 则对应P1，共需要排查比较n/2*n/2个对候选点。

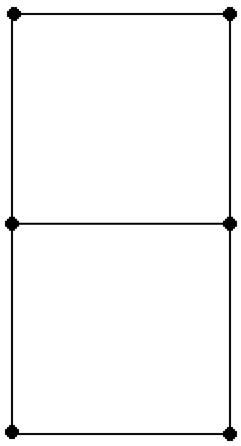
改进方案:
缩小排查的规模，降低幂次。

效率:
=在线性时间里查找= O(n)

能否在线性时间内找到p3,q3?

- 考虑P1中任意一点p，它若与P2中的点q构成最接近点对的候选者，则必有distance(p，q) < d。**满足这个条件的P2中的点一定落在一个d×2d的矩形R中**
- 由d的意义可知，P2中任何2个S中的点的距离都不小于d。由此可以推出**矩形R中最多只有6个S中的点。**
- 因此，在分治法的合并步骤中**最多只需要检查n/2×6=3n个候选者**

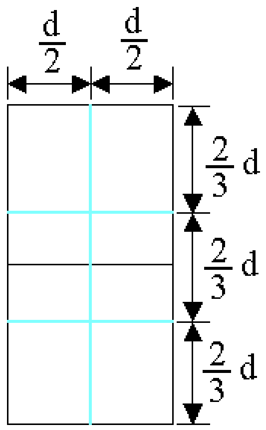




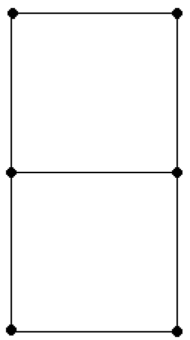
(b)

为什么只有6个候选点？？

- 考虑P1中任意一点p，它若与P2中的点q构成最接近点对的候选者，则必有distance(p，q) < d。**满足这个条件的P2中的点一定落在一个d×2d的矩形R中**
- 由d的意义可知，P2中任何2个S中的点的距离都不小于d。由此可以推出**矩形R中最多只有6个S中的点**。
- 因此，在分治法的合并步骤中**最多只需要检查6×n/2=3n个候选者**



(a)



(b)

证明:将矩形R的长为2d的边3等分，将它的长为d的边2等分，由此导出6个(d/2)×(2d/3)的矩形。若矩形R中有多于6个S中的点，则由鸽舍原理易知至少有一个(d/2)×(2d/3)的小矩形中有2个以上S中的点。设u，v是位于同一小矩形中的2个点，则

$$(x(u)-x(v))^2+(y(u)-y(v))^2 \le (d/2)^2+(2d/3)^2=\frac{25}{36}d^2$$

distance(u,v)<d。这与d的意义相矛盾。

最接近点对问题

- 为了确切地知道要检查哪6个点，可以将p和P2中所有S2的点投影到垂直线l上。由于能与p点一起构成最接近点对候选者的S2中点一定在矩形R中，所以它们在直线l上的投影点距p在l上投影点的距离小于d。由上面的分析可知，这种投影点最多只有6个。
- 因此，若将P1和P2中所有S中点按其y坐标排好序，则对P1中所有点，对排好序的点列作一次扫描，就可以找出所有最接近点对的候选者。对P1中每一点最多只要检查P2中排好序的相继6个点。

最接近点对问题

double **cpair2**(S)

{

n=|S|;

if (n < 2) **return** false;

1、 m=S中各点x间坐标的中位数;

构造S1和S2;

//S1={p∈S|x(p)<=m},

//S2={p∈S|x(p)>m}

2、 d1=**cpair2**(S1);

d2=**cpair2**(S2);

3、 dm=**min**(d1,d2);

4、 设P1是S1中距垂直分割线l的距离在dm之内的所有点组成的集合;

P2是S2中距分割线l的距离在dm之内所有点组成的集合;

将P1和P2中点依其y坐标值排序;

并设X和Y是相应的已排好序的点列;

5、 通过扫描X以及对于X中每个点检查Y中与其距离在dm之内的所有点(最多6个)可以完成合并;

当X中的扫描指针逐次向上移动时，Y中的扫描指针可在宽为2dm的区间内移动;

设dl是按这种扫描方式找到的点对间的最小距离;

6、 d=**min**(dm,dl);

return d;

}

最接近点对算法的复杂性分析

最接近点对算法的复杂性分析

```
double cpair2(S)
{
    n=|S|;
    if (n < 2) return false;
1、 m=S中各点x间坐标的中位数;
    构造S1和S2；
    //S1={p∈S|x(p)<=m},
    //S2={p∈S|x(p)>m}
2、 d1=cpair2(S1);
    d2=cpair2(S2);
3、 dm=min(d1,d2);
4、 设P1是S1中距垂直分割线l的距离在dm之内的所有点组成的集合；
    P2是S2中距分割线l的距离在dm之内所有点组成的集合；
    将P1和P2中点依其y坐标值排序；
    并设X和Y是相应的已排好序的点列；
5、 通过扫描X以及对于X中每个点检查Y中与其距离在dm之内的所有点(最多6个)可以完成合并；
    当X中的扫描指针逐次向上移动时，Y中的扫描指针可在宽为2dm的区间内移动；
    设dl是按这种扫描方式找到的点对间的最小距离；
6、 d=min(dm,dl);
    return d;
}
```

第3步和第6步是常数时间。

最接近点对算法的复杂性分析

```
double cpair2(S)
{
    n=|S|;
    if (n < 2) return false;
1、 m=S中各点x间坐标的中位数;
    构造S1和S2;
    //S1={p∈S|x(p)<=m},
    //S2={p∈S|x(p)>m}
2、 d1=cpair2(S1);
    d2=cpair2(S2);
3、 dm=min(d1,d2);
4、 设P1是S1中距垂直分割线l的距离在dm之内的所有点组成的集合;
    P2是S2中距分割线l的距离在dm之内所有点组成的集合;
    将P1和P2中点依其y坐标值排序;
    并设X和Y是相应的已排好序的点列;
5、 通过扫描X以及对于X中每个点检查Y中与其距离在dm之内的所有点(最多6个)可以完成合并;
    当X中的扫描指针逐次向上移动时，Y中的扫描指针可在宽为2dm的区间内移动;
    设dl是按这种扫描方式找到的点对间的最小距离;
6、 d=min(dm,dl);
    return d;
}
```

第1步和第5步是线性时间。

最接近点对算法的复杂性分析

double **cpair2**(S)

{

n=|S|;

if (n < 2) **return** false;

1、 m=S中各点x间坐标的中位数;

构造S1和S2;

//S1={p∈S|x(p)<=m},

//S2={p∈S|x(p)>m}

2、 d1=**cpair2**(S1);

d2=**cpair2**(S2);

3、 dm=**min**(d1,d2);

4、 设P1是S1中距垂直分割线l的距离在dm之内的所有点组成的集合;

P2是S2中距分割线l的距离在dm之内所有点组成的集合;

将P1和P2中点依其y坐标值排序;

并设X和Y是相应的已排好序的点列;

5、 通过扫描X以及对于X中每个点检查Y中与其距离在dm之内的所有点(最多6个)可以完成合并;

当X中的扫描指针逐次向上移动时，Y中的扫描指针可在宽为2dm的区间内移动;

设dl是按这种扫描方式找到的点对间的最小距离;

6、 d=**min**(dm,dl);

return d;

}

第2步时间 = *2T(n/2)*。

最接近点对算法的复杂性分析

double **cpair2**(S)

{

n=|S|;

if (n < 2) **return** false;

1、 m=S中各点x间坐标的中位数;

构造S1和S2；

//S1={p∈S|x(p)<=m},

//S2={p∈S|x(p)>m}

2、 d1=**cpair2**(S1);

d2=**cpair2**(S2);

3、 dm=**min**(d1,d2);

4、 设P1是S1中距垂直分割线l的距离在dm之内的所有点组成的集合；

P2是S2中距分割线l的距离在dm之内所有点组成的集合；

将P1和P2中点依其y坐标值排序；

并设X和Y是相应的已排好序的点列；

5、 通过扫描X以及对于X中每个点检查Y中与其距离在dm之内的所有点(最多6个)可以完成合并；

当X中的扫描指针逐次向上移动时，Y中的扫描指针可在宽为2dm的区间内移动；

设dl是按这种扫描方式找到的点对间的最小距离；

6、 d=**min**(dm,dl);

return d;

}

第4步时间 = *O(nlogn)*。

合并时间的线性优化

- 设计算法时常采用的预排序技术，即在使用分治法之前，预先将S中n个点依其y坐标值排好序，设排好序的点列为P*。
- 在执行分治法的第4步时，只要对P*作一次线性扫描，即可抽取出我们所需要的排好序的点列P1*和P2*。
- 在第5步中再对P1*作一次线性扫描，即可求得dl。因此，第4步和第5步的两遍扫描合在一起只要用O(n)时间。

最接近点对问题

复杂度分析

$$T(n)=\begin{cases} O(1) & n<4 \\ 2T(n/2)+O(n) & n\geq 4 \end{cases}$$

$$\boldsymbol{T(n)=O(nlogn)}$$

分治法的应用

循环赛问题

循环赛日程表

设计一个满足以下要求的比赛日程表：

有$n=2^k$个运动员

(1) 每个选手必须与其他n-1个选手各赛一次；

(2) 每个选手一天只能赛一次；

(3) 循环赛一共进行n-1天。

循环赛日程表

设计一个满足以下要求的比赛日程表：

(1) 每个选手必须与其他n-1个选手各赛一次；

(2) 每个选手一天只能赛一次；

(3) 循环赛一共进行n-1天。

按分治策略，将所有的选手分为两半，n个选手的比赛日程表就可以通过为n/2个选手设计的比赛日程表来决定。递归地用对选手进行分割，直到只剩下2个选手时，比赛日程表的制定就变得很简单。这时只要让这2个选手进行比赛就可以了。

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

归纳总结

分治法的基本思想

- 分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。
- **一对孪生兄弟**：
  - 如果原问题可分割成k个子问题，$1<k\leq n$ ，且这些子问题都可解，并可利用这些子问题的解求出原问题的解，那么这种分治法就是可行的。由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。
  - 在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解。这自然导致递归过程的产生。分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。

分治法的使用条件

- 该问题的规模缩小到一定的程度就可以容易地解决；
- 该问题可以分解为若干个规模较小的相同子问题，即该问题具有最优子结构性质。
- 利用该问题分解出的子问题的解可以合并为该问题的解；
- 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

分治法的基本步骤

- 分治法在每一层递归上都有三个步骤：
  - 分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题；
  - 解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题；
  - 合并：将各个子问题的解合并为原问题的解。

分治法的算法复杂性和求解

- 利用递推代入的方法

$$T(n)=\begin{cases} c & n=1 \\ aT(n/b)+cn^k & n>1 \end{cases}$$

$$T(n)=\begin{cases} O(n^{\log_b a}) & a>b^k \\ O(n^k \log_b n) & a=b^k \\ O(n^k) & a<b^k \end{cases}$$

编程作业（DDL: 11.07）

- 线性时间选择算法
- 平面最接近点对算法