算分 Ch-02 分治

1. 基本概念 **分治策略**是一种算法设计技术，其主要思想是：将原问题划分（或规约）为彼此独立的、规模较小而结构相同的子问题，递归地求解所有的子问题并将子问题的解组合从而得到原问题的解。分为**分解**，**递归求解**，**组合**三步。

2. 典型的分治算法

二分检索 $W(n) = O(\log n)$.

```
BinarySearch(T,x)
输入：排好序的数组 T[1..n]，要查找的数字 x
输出：x在T中时，输出x在T中的下标j；x不在T中时，输出j=0.
l <- 1; r <- n
while l <= r do
    m <- (l+r)/2
    if T[m] = x then return m
    else if T[m] > x then r <- m-1
    else l <- m+1
return 0
```

二分归并排序 $W(n) = O(n \log n)$.

```
MergeSort(A,p,r)
输入：数组A[p..r]，1 <= p <= r <= n
输出：从A[p]到A[r]按照递增顺序排好序的数组A
if p < r
then q <- (p+r)/2
    MergeSort(A,p,q)
    MergeSort(A,q+1,r)
    MergeSort(A,p,q,r)
```

```
Merge(A,p,q,r)
输入：按照递增顺序排好序的数组A[p..q]与A[q+1..r]
输出：按照递增顺序排序的数组A[p..r]
x <- q-p+1, y <- r-q
将A[p..q]复制到B[1..x]，将A[q+1..r]复制到C[1..y]
i <- 1, j <- 1, k <- p
while i<=x and j<=y do
    if B[i] <= C[j]
    then A[k] <- B[i]
         i <- i+1
    else A[k] <- C[j]
         j <- j+1
    k <- k+1
if i>x then 将C[j..y]复制到A[k..r]      // B已空
else 将B[i..x]复制到A[k..r]             // C已空
```

芯片测试 在 n 片芯片中找出 1 片好芯片，$W(n) = O(n)$.

```
Test(n)
输入：n片芯片构成的数组，其中好芯片至少比坏芯片多1片
输出：1片好芯片
k <- n
while k > 3 do
    将芯片分成 k/2 组
    for i = 1 to k/2 do
        if 2片好 then 任取一片留下
        else 2片都丢掉
    k <- 剩下的芯片数
if k = 3 then
    任取2片芯片测试
    if 1好1坏 then 取没坏的芯片
    else 任取1片被测芯片
if k=2 or 1 then 任取1片
```

大整数相乘 两个 n 位整数相乘，$W(n) = O(n^{\log 3}) = O(n^{1.59})$.

Strassen **矩阵乘法** 两个 n 阶矩阵相乘，$W(n) = O(n^{\log 7}) = O(n^{2.8075})$.

快速排序 $A(n) = O(n \log n)$.

```
QuickSort(A,p,r)
输入：数组A[p..r]，1 <= p <= r <= n
输出：从A[p]到A[r]按照递增顺序排好序的数组A
if p < r
    q <- Partition(A,p,r)
    A[p] <-> A[q]
    QuickSort(A,p,q-1)
    QuickSort(A,q+1,r)
```

```
Partition(A,p,r)
输入：数组A[p,r]
输出：j，A的首元素在排好序的数组中的位置
x <- A[p]
i <- p, j <- r+1
while true do
    repeat j <- j-1
    until  A[j] <= x
    repeat i <- i+1
    until  A[i] > x
    if i < j then A[i] <-> A[j]
    else return j
```

最邻近点对　求平面 n 个点中距离最近的 2 个点，$W(n) = O(n \log n)$.

```
MinDistance(P,X,Y)
输入：n个点的集合P，X和Y分别给出P中点的横、纵坐标
输出：最近的两个点及距离
1. 如果P中点数小于等于3，则直接计算其中的最小距离
2. 排序X,Y
3. 做垂直线l将P近似划分为大小相等的点集Pl和Pr
4. MinDistance(Pl,Xl,Yl)
5. MinDistance(Pr,Xr,Yr)
6. d <- min(dl,dr)
7. 对在线l左边距离d范围内的每个点，检查l右边是否有点与它的距离小于
   d，如果存在则修改d为新值
```

求第 k 小　在数组 S 中选第 k 小元素，$W(n) = O(n)$.

```
Findmax
输入：n个数的数组L
输出：max, k
1. max <- L[1]; k <- 1
2. for i <- 2 to n do
3.     if max < L[i]
4.     then max <- L[i]
5.          k <- i
6. return max, k
```

```
FindMaxMin
输入：n个数的数组L
输出：max, min
1. 将n个元素两两一组分成 n/2 组
2. 每组比较，得到 n/2 个较小和 n/2 个较大
3. 在 n/2 个较小中找最小 min
4. 在 n/2 个较大中找最大 max
```

```
FindSecondMax
输入：n个数的数组L
输出：SecondMax
1. k <- n
2. 将k个元素两两一组，分成 k/2 组
3. 每组的两个数比较，找到较大的数
4. 将被淘汰的较小的数在淘汰它的数所指向的链表中做记录
5. if k 为奇数 then k <- k/2 + 1
6. else k <- k/2
7. if k > 1 then goto 2
8. max <- 剩下的一个数
9. SecondMax <- max的链表中FindMax
```

```
Select(S,k)
输入：n个数的数组S，正整数k
输出：S中的第k个元素
1. 将S划分成5个一组，共 n/5 组
2. 每组找一个中位数，把这些中位数放到集合M中
3. m* <- Select(M, M/2)  // 划分成A，B，C，D
4. 把A和D中的每个元素与m*比较，小的构成S1，大的构成S2
5. S1 <- S1|C; S2 <- S2|B
6. if k = |S1|+1 then 输出m*
7. else if k <= |S1|
8.      then Select(S1, k)
9.      else Select(S2, k-|S1|-1)
```