第三章 分 治 法

算法的基本思想
关于排序问题
选择问题
关于矩阵乘法
快速**Fourier**变换
最接近点对问题

从折半搜索谈起

```
BiFind(a,n)
•  //在数组a[1..n]中搜索x，数组满足a[1] ≤ a[2] ≤ … ≤ a[n]。
  //如果找到x，则返回所在位置（数组元素的下标），否则返回 –1
   global a[1..n], n;
   integer left,right,middle;
   left:=1; right:=n;
•  while left ≤ right do
     middle:=(left+right)/2;
     if x=a[middle] then return(middle); end{if}
     if x>a[middle] then left:=middle+1;
     else right:=middle-1;
     end{if}
•  end{while}
   return(–1); //未找到x
end{BiFind}
```

$T(n)=T(n/2)+1$; $T(n)=\Theta(\log n)$

分治法控制流程

```
DiCo(p,q)
  global n，A[1..n];
  integer m,p,q;  // 1≤p≤q≤n
  if Small(p,q) then return(Sol(p,q));
  else m:=Divide(p,q); // p≤m<q
    return(Combine(DiCo(p,m),DiCo(m+1,q)));
  end{if}
end{DiCo}
```

Small(p,q)－规模判定；　Sol(p,q)－子问题直接求解函数；
Divide(p,q)－分割函数；　Combine(x,y)是解的合成函数。
时间复杂性递推关系：

$$T(n)=\begin{cases} g(n) & \text{当输入规模 n 比较小时，直接求解Sol(n)的用时} \\ 2T(n/2)+f(n) & f(n)\text{是组合Combine的用时} \end{cases}$$

求最小、最大值的二分算法

MaxMin(i,j,fmax,fmin) //A[1:n]是n元数组，参数i，j： 1≤i≤j≤n，
//使用该过程将数组A[i..j]中的最大最小元分别赋给fmax和fmin。
global n, A[1..n];
integer i, j;

- **if** i=j **then**
  fmax:=A[i]; fmin:=A[i]; //子数组A[i..j]中只有一个元素
- **elif** i=j-1 **then** //子数组A[i..j]中只有两个元素
  if A[i]<A[j] **then**
  fmin:=A[i]**;** fmax:=A[j]；
  else fmin:=A[j]; fmax:=A[i];
  end{if}
- **else**
  mid:=⌊(i+j)/2⌋; //子数组A[i..j]中的元素多于两个
  MaxMin(i, mid, lmax, lmin);
  MaxMin(mid+1, j, rmax, rmin);
  fmax:=max(lmax, rmax)；
  fmin:=min(lmin, rmin)；
- **end{if}**

end{MaxMin}

MaxMin复杂性分析

- T(n)来表示MaxMin所用的元素比较数，则上述递归算法导出一个递归关系式

$$T(n)=\begin{cases}0 & n=1\\ 1 & n=2\\ T(\lceil n/2\rceil)+T(\lfloor n/2\rfloor)+2 & n>2\end{cases}$$

- 当n是2的方幂时，设 $n=2^k$，有

$$\begin{aligned}T(n)&=2T(n/2)+2\\ &=2(2T(n/4)+2)+2\\ &\cdots\\ &=2^{k-1}T(2)+\sum_{1\le i\le k-1}2^i\\ &=3n/2-2\end{aligned}$$

*注意递归栈所用空间进出站所用时间

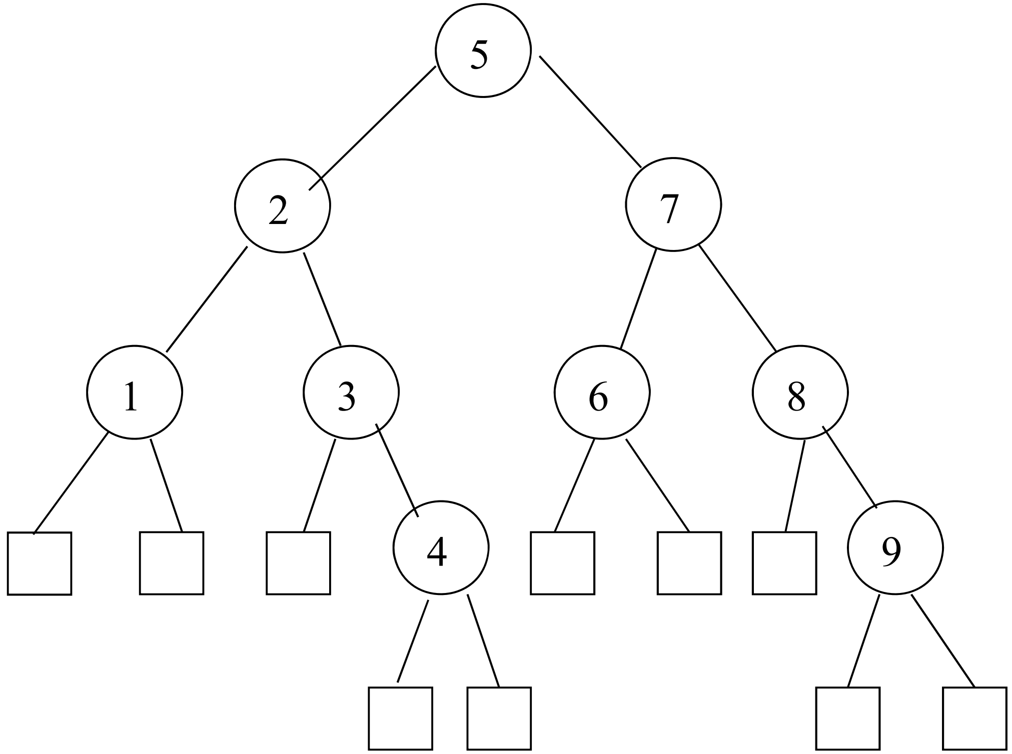
搜索算法的时间下界

- 数组A[1..n]满足：

  A[1]< A[2]< …< A[n]

要搜索元素x是否在A中。当$n \in [2^{k-1}, 2^k)$ 时，成功的折半搜索至多做k次比较，而不成功的折半搜索或者做k-1次比较，或者做k次比较。在n=9的情形，k=4 恰好是右侧二叉比较树的高。

- 任何一种比较为基础的搜索都形成一棵二叉比较树，它有n个内顶点，对应x在数组中的n个可能位置。所以，内顶点的深度不大于树的高度减1，即$n \le 2^k-1$ 。

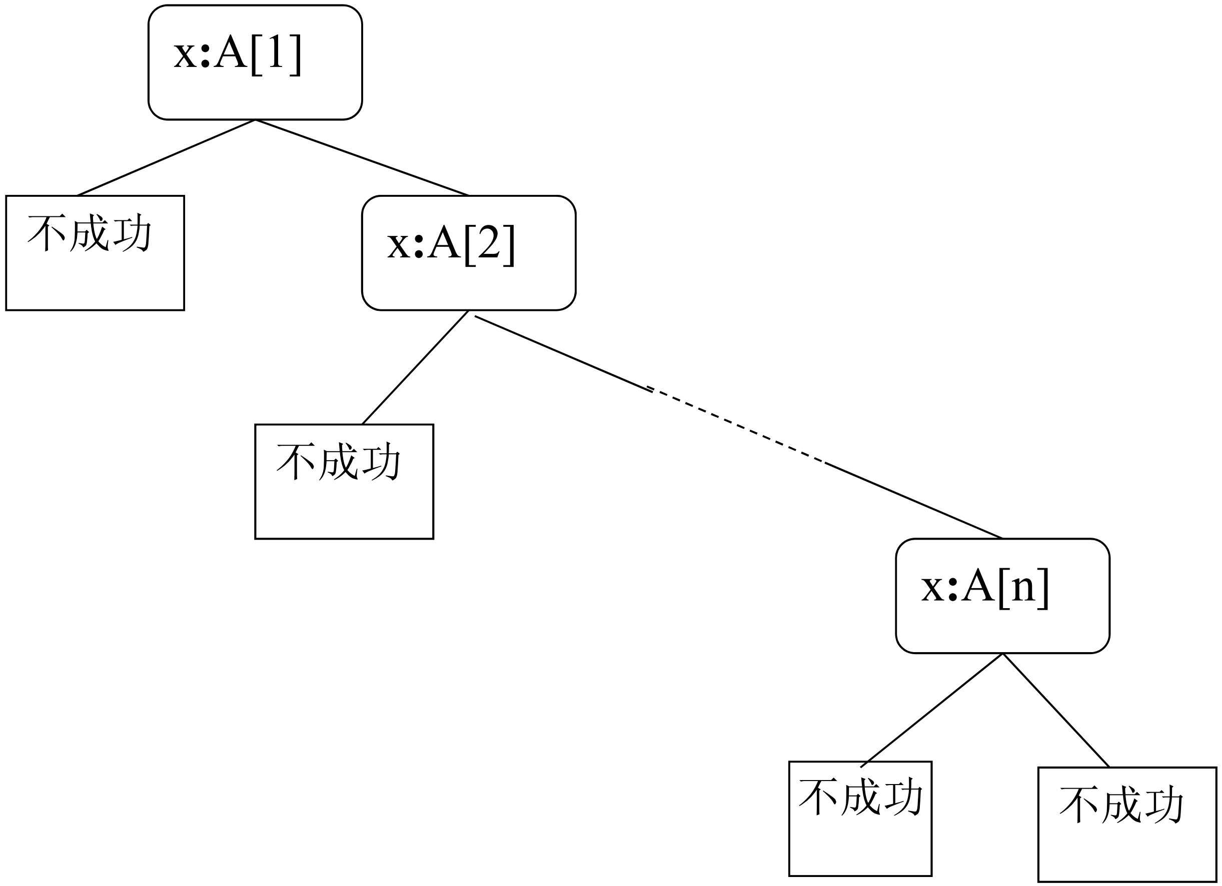


n = 9 时的折半搜索的二元比较树

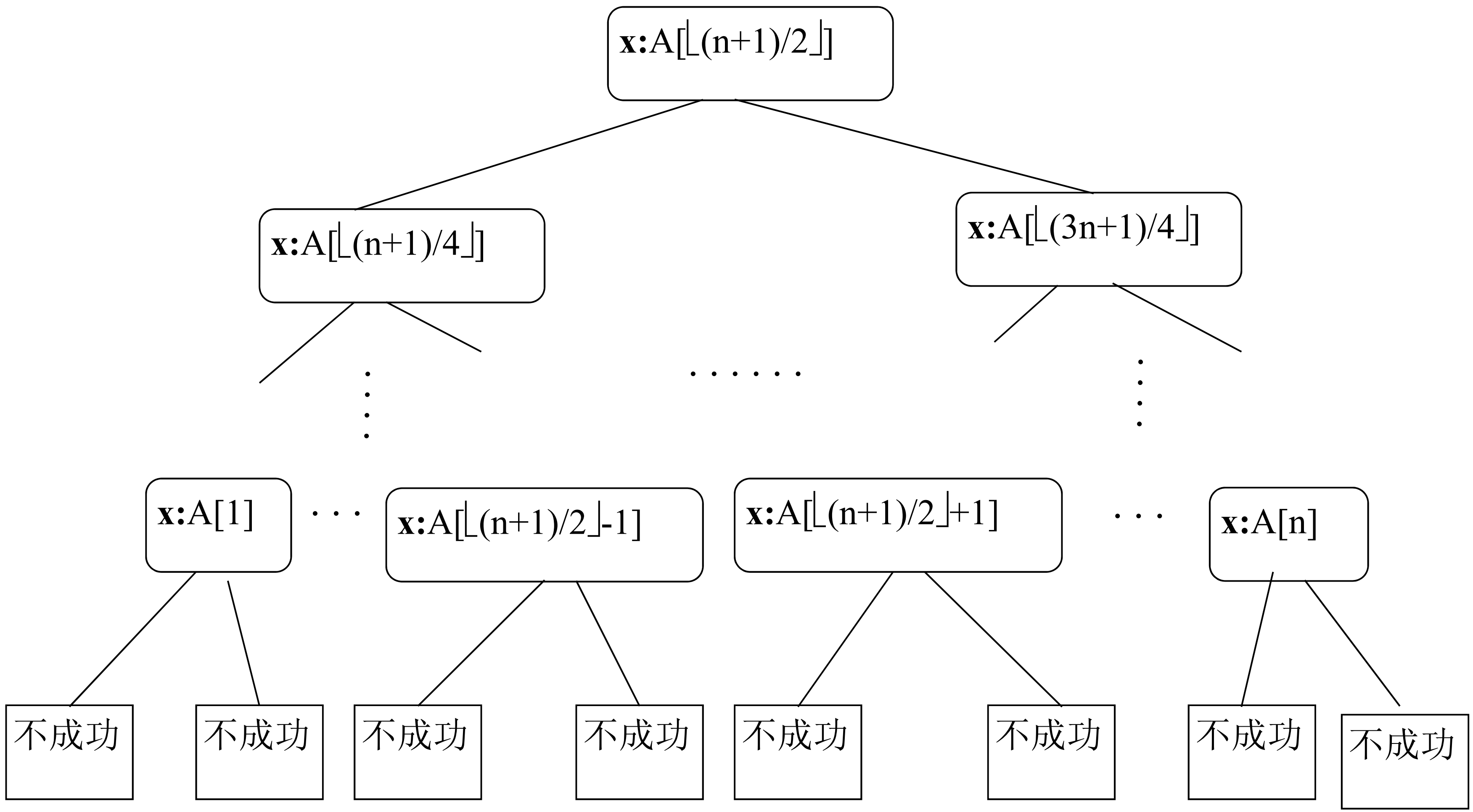
$$2^3 \le 9 < 2^4$$

所以，以比较为基础的搜索算法复杂度是Ω(logn)

模拟线性搜索过程



模拟折半搜索过程



x:A[⌊(n+1)/2⌋]
x:A[⌊(n+1)/4⌋]
x:A[⌊(3n+1)/4⌋]
x:A[1]
x:A[⌊(n+1)/2⌋-1]
x:A[⌊(n+1)/2⌋+1]
x:A[n]
不成功
不成功
不成功
不成功
不成功
不成功
不成功
不成功

关于排序算法

- 从插入排序算法谈起

InSort(a, n)
- **for** i **from** 2 **to** n **do**
-     x:=a[i];
-     **integer** j;
-     **for** j **from** i-1 by -1 **to** 1 **do**
-       **if** x<a[j] **then** a[j+1]:=a[j]; **end{if}**
-     **end{for}**
-     a[j+1]:=x;
- **end{for}**

end{InSort}

归并排序算法

MergeSort(low, high)

// A[low .. high]是一个全程数组，含有 high-low+1个

- // 待排序的元素。
- **integer** low, high;
- **if** low < high **then**

  mid:=⌊(low+high)/2⌋ //求当前数组的分割点

  MergeSort(low, mid) //将第一子数组排序

  MergeSort(mid+1, high) //将第二子数组排序

  Merge(low, mid, high) //归并两个已经排序的子数组
- **end{if}**

end{MergeSort}

合并过程1

Merge(low, mid, high) //已知全程数组A[low .. high], 其由
//两部分已经排好序的子数组构成：
// A[low .. mid]和A[mid+1 .. high].
//本程序的任务是将这两部分子数组合并成一个整体排好序
//的数组，再存于数组A[low .. high].
integer h, i, j, k, low, mid, high;
global A[low .. high];
local B[low .. high]; //借用临时数组B
h:=low, i:=low, j:=mid+1;
// h, j是拣取游标, i是向B存放元素的游标

- **while** h≤mid **and** j≤high **do** //当两个集合都没有取尽时
- **if** A[h]≤A[j] **then** B[i]:=A[h], h:=h+1;
  else B[i]:=A[j], j:=j+1;
- **end{if}**
- i:＝i+1;
- **end{while}**

合并过程2

```
if h>mid then
      //当第一子组元素被取尽，而第二组元素未被取尽时
   for k from j to high do
      B[i]:=A[k]; i:=i+1;
   end{for}
else
      //当第二子组元素被取尽，而第一组元素未被取尽时
   for k from h to mid do
      B[i]:=A[k]; i:=i+1;
   end{for}
end{if}
      //将临时数组B中元素再赋给数组A
for k from low to high do
   A[k]:=B[k];
end{for}
end{Merge}
```

归并排序算法的时间复杂度

- T(n)表示归并排序所用的时间，由于合并过程所用时间与n成正比：cn，其中c是一个正数，则有
- $$T(n)=\begin{cases} a & n=1 \\ 2T(n/2)+cn & n>1 \end{cases}$$
- 若n是2的方幂：$n=2^k$，直接推导可得

$$\begin{aligned} T(n) &= 2(2T(n/4)+cn/2)+cn \\ &= 4T(n/4)+2cn \\ &\cdots\cdots \\ &= 2^k T(1)+kcn \\ &= an+cn\log n \end{aligned}$$

- 对于一般的整数n，可以假定 $2^k < n \le 2^{k+1}$，于是 $T(n) \le T(2^{k+1})$

$$\begin{aligned} T(2^{k+1}) &= 2^{k+1}T(1)+(k+1)c2^{k+1} \\ &< 2na+2cn(\log n+1) \\ &= (2a+2c)n+2cn\log n \end{aligned}$$

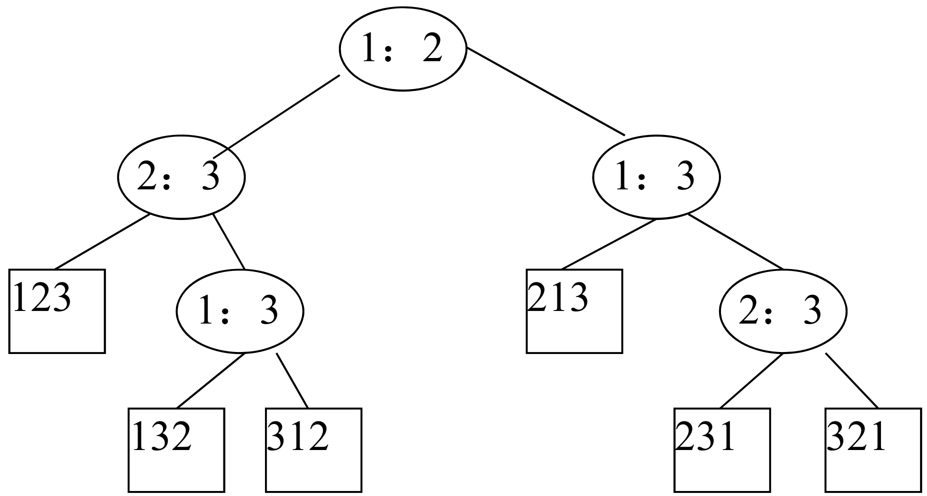
$$T(n)=O(n\log n)$$

以比较为基础的排序时间下界

每两个元素A[i]和A[j]的比较只有两种可能：

A[i]<A[j]或A[j]<A[i]，

形成二叉树。当A[i]<A[j]时进入左分支，当A[i]>A[j] 进入右分支。各个叶结点表示算法终止。从根到叶结点的每一条路径与一种唯一的排列相对应。由于n个不同元素的不同排列共有n！个因此比较树有n！个外部结点 。比较树中最长路径的长度（其是比较树的高）即是算法在最坏情况下所做的比较次数。要求出所有以比较为基础的排序算法在最坏情况下的时间下界，只需求出这些算法所对应的比较树的高度的最小值。



n=3时的比较树

高是k的二叉树的外结点至多是 2^k个，得

$n! \le 2^k$ 又 $n! \ge n(n-1)\cdots(\lceil n/2 \rceil) \ge (n/2)^{n/2-1}$

$$2^k \ge (n/2)^{n/2-1}$$

$$k \ge (n/2-1)\log(n/2) = \Theta(n\log n)$$

$$T(n) = \Omega(n\log n) \quad T(n) = \Theta(n\log n)$$

归并排序算法的思考

1.适当限制归并起点的规模；

2.避免元素来回移动(减低拷贝次数），采用(地址)邻接链表。

LINK:

- 数据 50 10 25 30 15 70 35 55
- 位置 (0) (1) (2) (3) (4) (5) (6) (7) (8)
- ↑k=0 ↑i =2 ↑j=5
- 指针 2 0 3 4 1 7 0 8 6
- q=2 r=5

- if A[i]≤A[j] then
- LINK[k]:=i; k:=i; i:=LINK[i];
- else
- LINK[k]:=j; k:=j; j:=LINK[j];
- end{if} k: 0, 1, 2, 3, 4, 5, 6, 7, 8
- LINK : 2, 8, 5, 4, 7, 3, 0, 1, 6

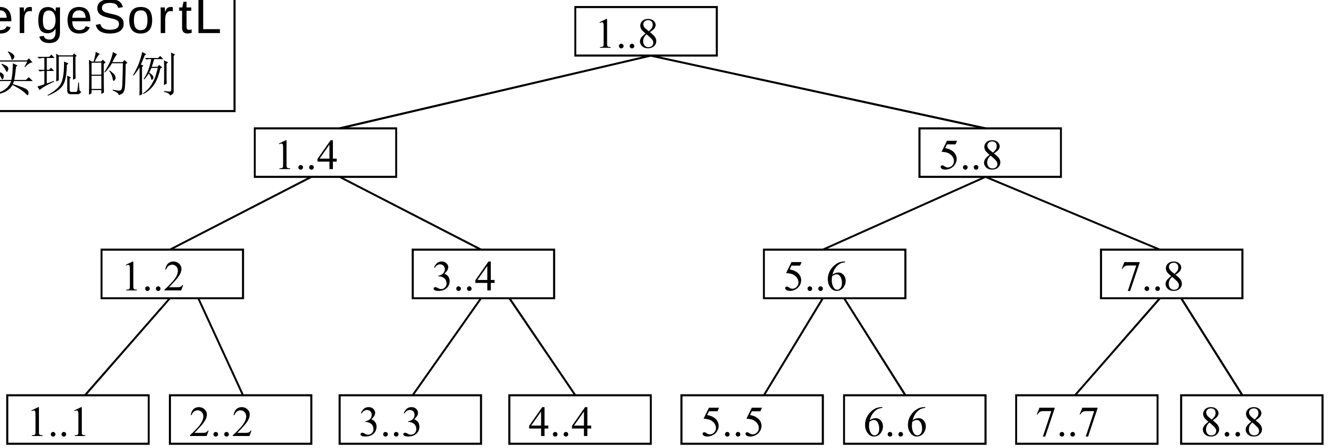
使用链接表的归并排序程序

```
MergeSortL(low, high, p) // Link是全程数组A[low..high]
  //的下标表, p指示这个表的开始处。利用Link将A按非降
  //顺序排列。
  global A[low..high]; Link[low..high];
  if high-low+1<16 then //设定子问题的最小规模Small
    InSort(A,Link, low,high,p)；
  else mid:=⌊(low+high)/2⌋；
    MergeSortL(low,mid,q); //返回q表
    MergeSortL(mid+1,high,r); //返回r表
    MergeL(q,r,p); 将表q和r合并成表p
  end{if}
end{MergeSortL}
```

使用邻接链表的合并过程

```
MergeL(q,r,p) // 由链接表q和r构造新的连接表。p、q、r是全程数组
  //Link[0..n]中两个表指针，这两个链表指出被划分的两个子组的地址排序，
  //而p指针指出两组归并后的地址排序。
 global n,A[1..n], Link[0..n];
 local integer i, j, k;
 i:=q; j:=r; k:=0; // 初始化，新表在Link[0]处开始
 while i≠0 and j≠0 do //当两个表皆非空时
    if A[i]≤A[j] then
      Link[k]:=i; k:=i; i:=Link[i]; //加一个新元素到此表
    else Link[k]:=j; k:=j; j:=Link[j];
    end{if}
  end{while}
  if i=0 then
    Link[k]:=j;
  else Link[k]:=i;
  end{if}
  p:=Link[0];
end{MergeL}
```

MergeSortL
实现的例



MergeSortL
调用过程
（分拆）

A = [50 , 10 , 25 , 30 , 15 , 70 , 35 , 55]　　数据表

(0) , (1) , (2) , (3) , (4) , (5) , (6) , (7) , (8)　　各元素位置

Link:=[0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0]　　初始化为零，逐步修改

- q r p
- 1>2→ 2, 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 [10,50]
- 3<4→ 3, 0 , 1 , 4 , 0 , 0 , 0 , 0 , 0 [10,50], [25,30]
- 2<3→ 2, 0 , 3 , 4 , 1 , 0 , 0 , 0 , 0 [10,25,30,50]
- 5<6→ 5, 0 , 3 , 4 , 1 , 6 , 0 , 0 , 0 [10,25,30,50],[15,70]
- 7<8→ 7, 0， 3 , 4 , 1 , 6 , 0 , 8， 0 [10,25,30,50],[15,70],[35,55]
- 5<7→ 5, 0， 3 , 4 , 1 , 7 , 0 , 8， 6 [10,25,30,50],[15, 35,55,70]
- 2<5→ 2, 8 , 5 , 4 , 7 , 3 , 0 , 1， 6 [10, 15,25,30, 35,50, 55,70]

链接表的合并过程

快速排序：Hoare 提出，划分数组

- 数组元素划分

```
proc Partition(m,p) // 被划分的数组是A[m,p-1]，选定做划分元素的是v:=A[m]
    integer m, p, i;
   global A[m ..p-1];
   v:=A[m]; i:=m;
•  loop
      loop i:=i+1; until A[i]>v; end{loop}   //自左向右查
      loop p:=p-1; until A[p]≤v; end{loop} //自右向左查
      if i<p then
        Swap(A[i],A[p]); //交换A[i]和A[p]的位置
      else go to *;
      end{if}
•  end{loop}
   *: A[m]:=A[p]; A[p]:= v; // 划分元素在位置p
 end{Partition}
```

数组A[1:9]=[65,70,75,80,85,60,55,50,45]划分过程

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	*i*	*p*
65	70	75	80	85	60	55	50	45	+∞	2	9
65	45	75	80	85	60	55	50	70	+∞	3	8
65	45	50	80	85	60	55	75	70	+∞	4	7
65	45	50	55	85	60	80	75	70	+∞	5	6
										6 > 5	
65	45	50	55	60	85	80	75	70	+∞	1	5
60	45	50	55	|65 |	85	80	75	70	+∞		

快速排序算法

QuickSort(p,q) //将数组A[1..n]中的元素A[p], A[p+1], … ,
// A[q] 按不降次序排列，并假定A[n+1]是一个确定数，且大
//于A[1..n]中所有的数。

integer p,q;

global n, A[1..n];

- **if** p<q **then**

  j:=q+1; Partition(p,j); // 划分后j成为划分元素的位置

  QuickSort(p,j-1);

- QuickSort(j+1,q);
- **end{if}**

end{QuickSort}

快速排序算法的平均时间复杂度

- 最坏情况下的时间复杂度为 $O(n^2$。)
- 平均时间复杂性$C_A(n)$，

约定：参加排序的n个元素互不相同；Partition中的划分元素v是随机选取的。

调用Partition(m,p)时，所取划分元素v是A[m, p-1]中第*i*小元素的概率均为1/(p-m)，1≤i≤p-m，因而留下待排序的两个子组为A[m..j-1]和A[j+1..p-1]的概率是1/(p-m)，m≤j≤p-1。由此得递归关系式

$$C_A(n) = n - 1 + \frac{1}{n}\sum_{1 \le k \le n}(C_A(k-1) + C_A(n-k)), \quad C_A(0) = C_A(1) = 0$$

其中，n-1是Partition第一次被调用时所需要的元素比较次数。

$$nC_A(n) = n(n-1) + 2(C_A(0) + C_A(1) + \cdots + C_A(n-1))$$

用n-1替换上式中的n得

$$(n-1)C_A(n-1) = (n-1)(n-2) + 2(C_A(0) + C_A(1) + \cdots + C_A(n-2))$$

$$C_A(n)/(n+1) = C_A(n-1)/n + \frac{2(n-1)}{n(n+1)} \le C_A(n-1)/n + 2/n$$

$$C_A(n)/(n+1) \le C_A(1)/2 + 2\sum_{2 \le i \le n} 1/i, \qquad \sum_{2 \le i \le n} 1/i < \int_1^n \frac{dx}{x} = \ln n$$

$$C_A(n) < 2(n+1)\ln n = O(n\log n)$$

选择问题：选出数组中第k小元素

采用排序方法: $T(n)=\Theta(n\log n)$ 利用划分算法: $T(n)=O(n^2)$

```
PartSelect(A, n, k) //在数组A[1..n]中找第k小元素 t，并将其
  //存放于位置k，即A[k]=t。而剩下的元素按着以t为划分元素
  //的划分规则存放。
   integer n, k, m, r, j;
   m:=1; r:=n+1; A[n+1]:= +∞；
   loop
      j:= r ;
      Partition(m,j);
      case:
         k=j : return // 返回j,当前数组的元素A[j]是第j小元素
         k<j : r:=j; // j是新的下标上界
      else : m:=j+1;  //j+1是新的下标下界
      end{case}
   end{loop}
end{PartSelect}
```

选择算法

```
Select(A, m, p, k) // 返回一个i值，
  //使得A[i]是A[m..p]中第k小元素。 r
  //是固定正整数。
  global r;
  integer n, i, j;
  if p-m+1≤r then
    InSort(A, m, p);
    return(m+k-1);
  end{if}
  loop
   n:=p-m+1;
   for i to ⌊n/r⌋ do  //计算中间值
     InSort(A, m+(i-1)*r, m+i*r-1);
     //将中间值收集到A[m..p]的前部:
     Swap(A[m+i-1], A[m+(i-1)*r
                       +⌊r/2⌋-1]);
   end{for}
```

```
•  j:=Select(A, m, m+⌊n/r⌋-1, ⌈⌊n/r⌋/2⌉);
•  Swap(A[m], A[j]); //产生划分元素
•  j:=p+1;
•  Partition(m, j);
•  case:
•     j-m+1=k :   return(j);
•     j-m+1>k :   p:=j-1;
•     else    m:=j+1;
•  end{case}
  end{loop}
 end{Select}
```

选择算法时间复杂性分析：r=5

假设数组A中的元素都是互不相同的。

具有5个元素的数组的中间值u是该数组的第3小元素，此数组至少有3个元素不大于u；$\lfloor n/5 \rfloor$个中间值中至少有$\lceil \lfloor n/5 \rfloor /2 \rceil$个不大于这些中间值的中间值v。因而，在数组A中至少有$3*\lceil \lfloor n/5 \rfloor /2 \rceil \geq 1.5*\lfloor n/5 \rfloor$个元素不大于v。即，A中至多有 $n-1.5*\lfloor n/5 \rfloor = n-1.5*(n/5-e/5) \leq 0.7n+1.2$ 个元素大于v。同理，至多有0.7n+1.2个元素小于v。这样，以v为划分元素所产生的新的数组至多有0.7n+1.2个元素.当$n \geq 24$时,$0.7n+1.2 \leq 0.75n=3n/4$

程序Select中，从一层到下一层递归时，实际上相当于两次调用了Select：一次体现在语句 j:=Select(A, m, m+$\lfloor n/r \rfloor$-1, $\lceil \lfloor n/r \rfloor /2 \rceil$); 另一次体现在Partition(m, j)及后面的case语句组,完成这些操作$\Theta(n)$次，但主程序接着就要调用自身，执行规模不超过3n/4的选择问题。这两步涉及的数组规模分别是n/5和$\leq 3n/4$。程序中其它执行步的时间复杂度都至多是n的倍数。如果用T(n)表示数组长度为n的时间复杂度，则当$n \geq 24$时，有递归关系式

$$T(n) \leq T(n/5) + T(3n/4) + cn$$

其中c是常数。从递推关系式出发，用数学归纳法可以证明

$$T(n) \leq 20cn$$

- 所以，在最坏情况下，Select算法的时间复杂度是 $T(n) = O(n)$

矩阵乘法与大整数乘法

- 矩阵的加法和乘法

$$S(i,j)=A(i,j)+B(i,j), \qquad 1\le i,j\le n, \quad T(n)=O(n^2)$$

$$C(i,j)=\sum_{k=1}^{n}A(i,k)*B(k,j) \qquad 1\le i,j\le n, \quad T(n)=O(n^3)$$

- 分块矩阵乘法

$$A*B=\begin{bmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{bmatrix}\begin{bmatrix}B_{11} & B_{12}\\ B_{21} & B_{22}\end{bmatrix}=\begin{bmatrix}C_{11} & C_{12}\\ C_{21} & C_{22}\end{bmatrix}$$

$$C_{11}=A_{11}B_{11}+A_{12}B_{21} \qquad C_{12}=A_{11}B_{12}+A_{12}B_{22}$$

$$C_{21}=A_{21}B_{11}+A_{22}B_{21} \qquad C_{22}=A_{21}B_{12}+A_{22}B_{22}$$

- 时间复杂度函数

$$T(n)=\begin{cases}b & n\le 2\\ 8T(n/2)+dn^2 & n>2\end{cases}, \qquad T(n)=bn^3/8+dn^2(\frac{n}{2}-1)$$

Strassen 算法

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = (A_{21} + A_{22})B_{11}$$
$$R = A_{11}(B_{12} - B_{22})$$
$$S = A_{22}(B_{21} - B_{11})$$
$$T = (A_{11} + A_{12})B_{22}$$
$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + U$$

$$T(n) = \begin{cases} b & n \le 2 \\ 7T(n/2) + an^2 & n > 2 \end{cases}$$

$$\begin{aligned} T(n) &= an^2(1 + 7/4 + (7/4)^2 + \cdots + (7/4)^{k-2}) + 7^{k-1}T(2) \\ &= an^2\left(\frac{16}{21}\left(\frac{7}{4}\right)^{\log n} - \frac{4}{3}\right) + \frac{b}{7}(7)^{\log n} \\ &= an^2\left(\frac{16}{21}(n)^{\log\frac{7}{4}} - \frac{4}{3}\right) + \frac{b}{7}(n)^{\log 7} \\ &= \left(\frac{16a}{21} + \frac{b}{7}\right)n^{\log 7} - \frac{4a}{3}n^2 \\ &= \Theta\left(n^{2.81}\right) \end{aligned}$$

7次乘法是必要的

(Hoperoft and Kerr, 1971)

目前最好的矩阵乘法的时间复

杂度为 $O(n^{2.36})$

大整数乘法

大整数

$$a = \sum_{0 \le i \le m-1} a_i \cdot B^i \qquad b = \sum_{0 \le i \le n-1} b_i \cdot B^i \qquad m \ge n$$

乘法格式

$$\begin{array}{l}
B^0 \quad B^1 \cdots B^{n-1} \cdots B^{m-1} \quad B^m \quad \cdots \quad B^{m+n-2} \\
a_0 b_0 \quad a_1 b_0 \cdots a_{n-1} b_0 \cdots a_{m-1} b_0 \\
\qquad a_0 b_1 \cdots a_{n-2} b_1 \cdots a_{m-2} b_1 \quad a_{m-2} b_1 \\
\qquad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \\
\qquad\qquad a_0 b_{n-1} \cdots a_{m-n} b_{n-1} \; a_{m-n+1} b_{n-1} \cdots a_{m-1} b_{n-1}
\end{array}$$

$$a_i b_j + q_{i+j-1} = q_{i+j} B + r_{i+j}$$

时间复杂度: $O(mn)$

Karatsuba 算法

$$a=\sum_{i=0}^{n/2-1}a_i\cdot B^i+\left(\sum_{i=n/2}^{n-1}a_i\cdot B^{i-n/2},\right)B^{n/2}\qquad b=\sum_{i=0}^{n/2-1}b_i\cdot B^i+\left(\sum_{i=n/2}^{n-1}b_i\cdot B^{i-n/2}\right)B^{n/2}$$

$$v_1=\sum_{i=0}^{n/2-1}a_i\cdot B^i,\quad u_1=\sum_{i=n/2}^{n-1}a_i\cdot B^{i-n/2},\ v_2=\sum_{i=0}^{n/2-1}b_i\cdot B^i,\quad u_2=\sum_{i=n/2}^{n-1}b_i\cdot B^{i-n/2}$$

$$a=u_1B^{n/2}+v_1,\,b=u_2B^{n/2}+v_2$$

$$\begin{aligned}ab&=u_1u_2B^n+(u_1v_2+u_2v_1)B^{n/2}+v_1v_2\\&=u_1u_2B^n+\big((u_1-v_1)(v_2-u_2)+v_1v_2+u_1u_2\big)B^{n/2}+v_1v_2\end{aligned}$$

$$T(n)=O(n^{1.585})$$

$$T(1)=1$$

$$T(n)\le 3T(n/2)+cn$$

$$T(n)\le 3^k+2cn\frac{(3/2)^k-1}{3/2-1}<3^k+2cn(3/2)^k$$

$$\begin{aligned}T(n)&<3^{\log_2 n}+4cn(3/2)^{\log_2 n}\\&=n^{\log_2 3}+4cn^{\log_2 3}\\&=(4c+1)n^{\log_2 3}\\&\approx(4c+1)n^{1.585}\end{aligned}$$

快速Fourier变换

- 连续函数$a(t)$的Fourier变换

$$A(f)=\int_{-\infty}^{\infty}a(t)e^{2\pi ift}dt$$

- $A(f)$ 的逆变换为

$$a(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty}A(f)e^{-2\pi ift}dt$$

- N个离散数据 $a=(a_0,a_1,\cdots,a_{N-1})$的Fourier变换

$$A_j=\sum_{0\le k\le N-1}a_k e^{2\pi ijk/N},\quad 0\le j<N$$

- 其逆变换为

$$a_k=\frac{1}{N}\sum_{0\le j\le N-1}A_j e^{-2\pi ijk/N},\quad 0\le k<N$$

- 令 $\omega=e^{2\pi i/N}$，多项式 $a(x)=\sum_{0\le k\le N-1}a_k x^k$ ，则 $A_j=a(\omega^j),\ j=0,1,\cdots,N-1$

当$N=2n$时，ω^2是n次本原单位根，而且 $\omega^{j+n}=-\omega^j,\ j=0,1,\cdots,n-1$

$$a(\omega^j)=b(\omega^{2j})\omega^j+c(\omega^{2j}),\quad a(\omega^{j+n})=-b(\omega^{2j})\omega^j+c(\omega^{2j})$$

快速Fourier变换算法

FFT(N, a,w,A)

- # $N=2^m$,w是n次单位根，a是已知的N元数组，代表多项式a(x)的系数，
- # A是计算出来的N元数组，$A[j]=a(w^j)$, j=0,1,…,N-1.
- **real** b[ ], c[ ]; **int** j;
- **complex** B[ ], C[ ], wp[ ];
- **if** N=1 **then** A[0]:=a[0];
- **else**
- n:=N/2;
- **for** j **from** 0 **to** n-1 **do**
- b[j]:=a[2*j+1]; c[j]:=a[2*j];
- **end**{for}
- **end**{if}
- FFT(n,b,w*w,B);
- FFT(n,c,w*w,C);
- wp[0]:=1;
- **for** j **from** 0 **to** n-1 **do**
- wp[j+1]:=w*wp[j];
- A[j]:= C[j]+B[j]*wp[j]; A[j+n]:= C[j]-B[j]*wp[j];
- **end**{for}
- **end{FFT}**

- 以$T(N)$记算法FFT的时间复杂度，则

$$T(N)=\begin{cases} a, \text{if } N=1 \\ 2T(N/2)+cN, \text{if } N>1 \end{cases}$$

- 所以，算法的复杂度为 $T(N)=O(N\log N)$ 。
-

最接近点对问题

- 一维最近点对问题

直线上的n个点采用排序+扫描方法，时间复杂度为 $O(n\log n)$

采用二分算法，

$$T(n)=\begin{cases} a, & n<4; \\ 2T(n/2)+cn, & n\geq 4 \end{cases}$$

$$O(n\log n)$$

- **求一维最近点对距离分治算法**

- **proc ClosPair1**(S，d)
  //S是实轴上点的集合，参数d表示S中最近
- //点对的距离
- **global** S,d;
- **integer** n;
- **float** m,p,q;
- n:=|S|;
- **if** n<2 **then** d:=∞; return(false); **end{if}**
- m:=S中各点坐标的中位数； //划分集合S
- S1:={x∈S| x≤m}； S2:={x∈S| x>m}；
- ClosPair1(S1,d1);
- ClosPair1(S2,d2);
- p:=max(S1); q:=min(S2);
- d:=min(d1,d2,q-p);
- return(true);
- **end{ClosPair1}**

二维最近点对问题

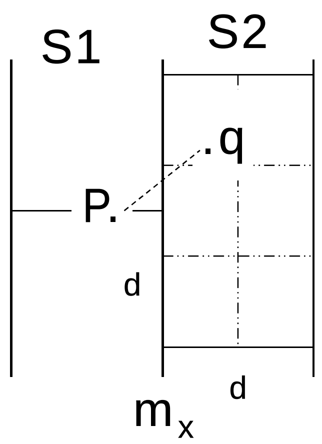
- 用x-坐标的中位数去划分点集

$$S1 := \{p \in S \mid x(p) \le m_x\}, \quad S2 := \{p \in S \mid x(p) > m_x\}$$

$$d = \min\{d_1, d_2\}$$

- 压缩最近点对搜索范围

$$S_p = \{q \in S \mid m_x < x(q) \le m_x + d \text{ and } y(p) - d \le y(q) \le y(p) + d\}$$



$$\sqrt{(d/2)^2 + (2d/3)^2} = 5d/6$$

- 每个方格中至多有一个S2中的点
- S_p 中至多含有S2中的6个点。

- 只需检查至多 $6 \times \lceil n/2 \rceil = 3n + 3$ 个点对

求二维最近点对距离分治算法

```
proc ClosPair2(S,d) //S是平面上点的
  //集合,按照y−坐标不降的次序排
  //好,假定不同点的x−坐标是不同
  //的.参数d表示S中最近点对的距
  //离,dist(p,q)是点对(p,q)间的距离
 global S,d;
 integer n;  float m,p,q; n:=|S|;
 if n<2 then d:=∞; return(false); end{if}
 mx:=S中各点x−坐标的中位数;
  //划分集合S成S1和S2，它们也都
  //是y−坐标不降的。
  S1:={p∈S| x(p)≤mx};
  S2:={q∈S| x(q)>mx};
  ClosPair2(S1,d1);
  ClosPair2(S2,d2);
   d:=min{d1,d2};
```

```
 //检查距离直线x=mx不远于d的两
 //个条形区域中的点对
 P1:={p∈S1|mx-d≤x(p)};
 P2:={q∈S2| x(q)≤mx+d};
 flag:=1;
 for i to |P1| do
   k:=flag;
   while y(P2[k])<y(P1[i])-d do
       k:=k+1;
   end{while}
   flag:=k;
   for j from flag to |P2| do
     if y(P2[j])>y(P1[i])+d then
        break;
     else d:=min{d,dist(P1[i],P2[j])};
     end{if}
   end{for}
  end{for}
  return(true);
 end{ClosPair2}
```

时间复杂度

$$T(n) \le \begin{cases} a, & n < 4 \\ 2T(n/2) + cn, & n \ge 4 \end{cases}$$

$$T(n) = O(n \log n)$$