《算法设计与分析》

第三章 分治法

马丙鹏

2024年9月30日

中国科学院大学
University of Chinese Academy of Sciences

第三章 分治法

- 3.1 一般方法
- 3.2 二分检索
- 3.3 找最大和最小元素
- 3.4 归并排序
- 3.5 快速排序
- 3.6 选择问题
- 3.7 最接近点对问题
- 3.8 斯特拉森矩阵乘法
- 3.9 大整数乘法

3.4 归并排序

- 排序问题
  - 给定一n个元素的集合A，按照某种方法将A中的元素按非降或非增次序排列。
  - [例]
    - 输入：(8, 5, 4, 9)
    - 输出：(4, 5, 8, 9)

      或 (9, 8, 5, 4)
  - 二分检索
    - 已知一个按非降次序排列的元素表$a_1, a_2, \ldots, a_n$,判定某给定的元素x是否在该表中出现。

3.4 归并排序

- 插入排序
  - 基本思想

```
for j=2 to n do
    将A(j)放到已排序集合
    A(1:j-1)的正确位置上
repeat
```

(8, 5, 4, 9)
(8, 5, 4, 9)
(5, 8, 4, 9)
(5, 8, 4, 9)
(4, 5, 8, 9)
(4, 5, 8, 9)
(4, 5, 8, 9)

3.4 归并排序

■ 插入排序

算法3.7 插入排序

```
procedure INSERTIONSORT(A, n)
    A(0)←-∞//设置初始边界值
    for j←2 to n do //A(1:j-1)已排序//
        item←A(j); i←j-1
        while item<A(i) do  //0≤i<j//
            A(i+1)←A(i); i←i-1
        repeat
        A(i+1) ←item;
    repeat
end INSERTIONSORT
```

i指示的是j之前的一位，即当前已排序子表的最末一个元素的下标

数组元素的比较和移动

将本次考虑的元素插入相应位置

3.4 归并排序

■ 插入排序

例: 用插入排序对6, 9, 3, 4, 1, 5进行按非降排序

i	0	1	2	3	4	5	6
a[i]	-	6	9	3	4	1	5

插入“9”：9>6，故9置于6之后

i	0	1	2	3	4	5	6
a[i]	-	6	9	3	4	1	5

插入“3”：3<6，故6，9往后移，3置于原6的位置

i	0	1	2	3	4	5	6
a[i]	-	3	6	9	4	1	5

插入“4”：3<4<6，故6，9往后移，4置于原6的位置

i	0	1	2	3	4	5	6
a[i]	-	3	4	6	9	1	5

插入“1”：1<3，故3，4，6，9均往后移，1置于原3的位置

i	0	1	2	3	4	5	6
a[i]	-	1	3	4	6	9	5

插入“5”：4<5<6，故6，9往后移，5置于原6的位置

i	0	1	2	3	4	5	6
a[i]	-	1	3	4	5	6	9

3.4 归并排序

- 插入排序
  - 性能分析
    - 最好情况：输入数据已按非降序排列，不进入while循环，则最好情况下计算时间为$\Omega(n)$
    - 最坏情况：输入数据按非增次序排列，每次内层while循环执行j次(j=2, 3, …, n)。则有，

$$\sum_{2 \le j < n} j = (n(n+1))/2 - 1$$

$$= \Theta(n^2)$$

3.4 归并排序

- 分治策略求解
  - 基本设计思想：
    - 将原始数组A中的元素分成两个子集合： A1(1: ⌊n/2⌋)和A2(⌊n/2⌋+1: n)。
    - 分别对这两个子集合单独排序，
    - 然后将已排序的两个子序列归并成一个含有n个元素的排序好的序列。
  - 这样的一个排序过程称为归并排序。

3.4 归并排序

■ 算法流程:算法3.8归并排序

```
procedure MERGESORT(low, high)
//A(low: high)是一个全程数组，low和high分别指示当前待排序区间的最小下标和最大下标，它含有high-low+1≥0个待排序的元素//
    integer low, high
    if low<high then
        mid ← ⌊(low + high)/2⌋          //计算中分点//
        call MERGESORT(low, mid)        //在第一个子集合上排序(递归)//
        call MERGESORT(mid+1, high)     //在第二个子集合上排序(递归)//
        call MERGE(low, mid, high)      //归并已排序的两子集合//
    endif
end MERGESORT
```

3.4 归并排序

//A(low, high)是一个全程数组,它含有两个放在A(low, mid)和A(mid+1,high)中的已排序的子集合.目标是将这两个已排序的集合归并成一个集合，并存放到A(low, high)中//

■ 算法流程:算法3.9 使用辅助数组归并两个已排序的集合

```
procedure MERGE(low, mid, high)
    integer h, i, j, k, low, mid, high; //low≤mid＜high//
    global A(low, high); local B(low, high)
    h←low; i←low; j←mid+1;
    while h≤mid and j≤high do //当两个集合都没有取尽时,将较小的元素先存放到B中//
        if A(h)≤A(j) then B(i) ←A(h); h←h+1  //如果前一个数组中的元素较小//
                     else B(i) ←A(j); j←j+1      //如果后一个数组中的元素较小//
        endif
        i ← i+1
    repeat
    if h>mid then for k←j to high do B(i) ←A(k); i←i+1; repeat
             else   for k←h to mid do B(i) ←A(k); i←i+1; repeat
    endif
    for k ←low to high do  A(k) ←B(k) repeat
end MERGE
```

处理两个已排序的序列

剩余元素处理过程

将已排序的集合复制到A数组

3.4 归并排序

■ 性能分析

- □过程MERGE的归并时间与两数组元素的总数成正比 (可表示为：cn，其中n为元素数，c为某正常数)
- □MERGESORT的排序时间用递推关系式表示如下：

$$T(n)=\begin{cases} a & n=1,\ a\text{是常数} \\ 2T(n/2)+cn & n>1,\ c\text{是常数} \end{cases}$$

递归调用一直进行到子区间仅含一个元素时为止

化简：若$n=2^k$, 则有,

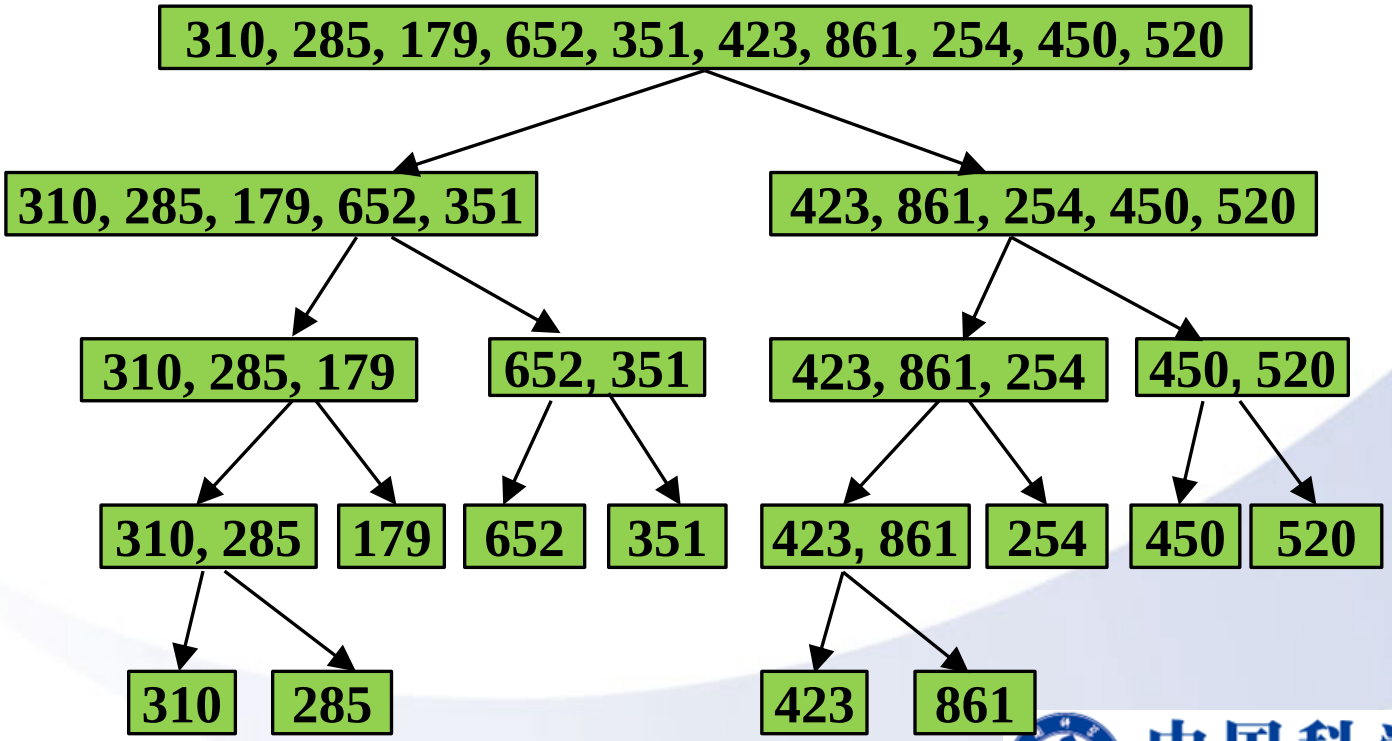
$$T(n) = 2(2T(n/4)+cn/2)+cn = 4T(n/4) + 2cn$$
$$= 4(2T(n/8) + cn/4) + 2cn = \ldots = 2^kT(1)+kcn$$
$$= an+cnlogn \quad //k=logn//$$

若$2^k<n<2^{k+1}$, 则有$T(n)\le T(2^{k+1})$。

所以得：$T(n) = O(nlogn)$

3.4 归并排序

- 归并排序示例
  - 设A=(310, 285, 179, 652, 351, 423, 861, 254, 450, 520)
  - 划分过程



3.4 归并排序

■ 归并排序示例

□左分枝的划分与归并，

(310 | 285 | 179 | 652, 351 | 423, 861, 254, 450, 520)

第一次归并：(285, 310 | 179 | 652, 351 | 423, 861, 254, 450, 520)

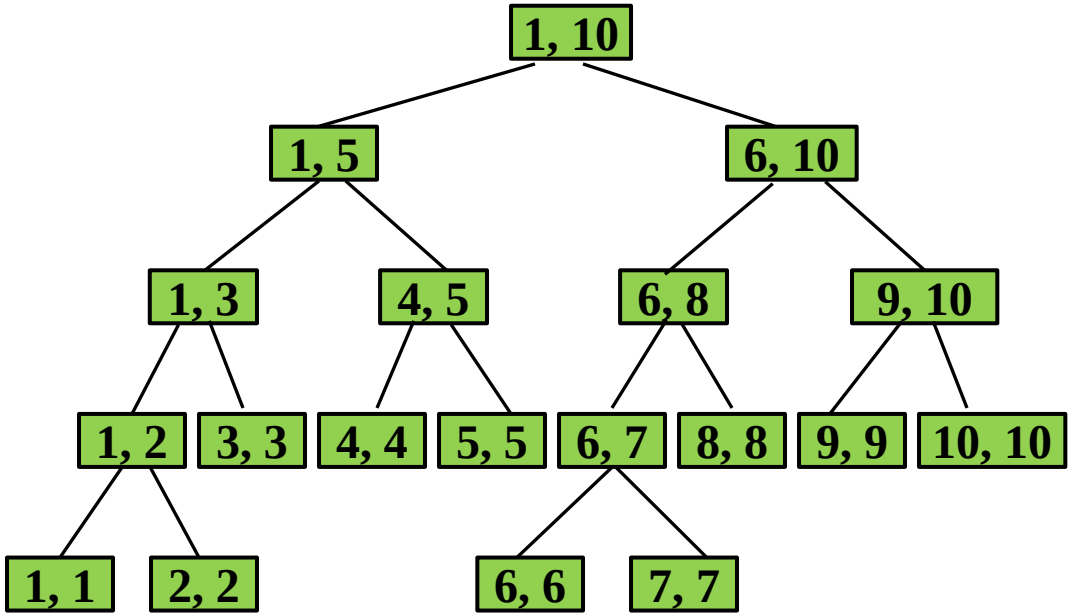
第二次归并：(179, 285, 310 | 652, 351 | 423, 861, 254, 450, 520)

第三次归并：(179, 285, 310 | 351, 652 | 423, 861, 254, 450, 520)

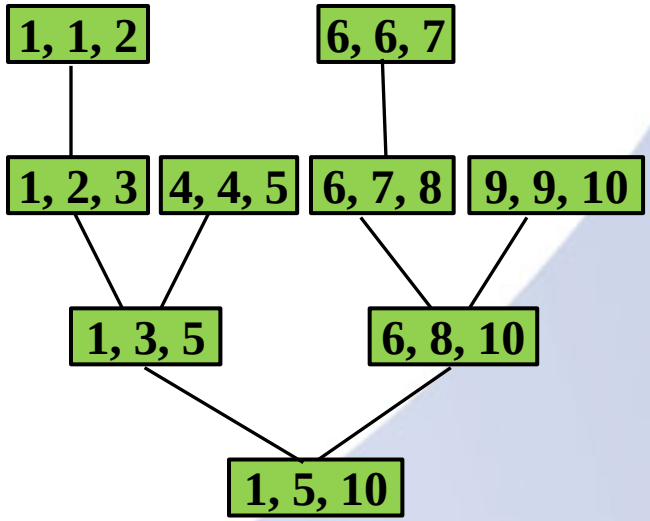
第四次归并：(179, 285, 310, 351, 652 | 423, 861, 254, 450, 520)

3.4 归并排序

- 用树结构描述归并排序过程



MergeSort(1, 10)的调用



Merge的调用

3.4 归并排序

- 算法3.8存在的问题
  - 递归层次太深
    - 在MERGESORT的执行过程中，当集合仅含有两个元素时，仍要进一步做递归调用，直至每个集合仅含有一个元素时才退出递归调用。
    - 在集合含有仅相当少的元素时，较深层次的递归调用使得时间过多地消耗在处理递归上。
  - 元素在数组A和辅助数组B之间的频繁移动
    - 每次归并，都要首先将A中的元素移到B中，在由B复制到A的对应位置上。

3.4 归并排序

- 改进措施
  - 针对递归层次问题
    - 采用能在小规模集合上有效工作的其它算法，直接对小规模集合处理。
    - 如INSERTIONSORT算法。
  - 针对元素频繁移动问题
    - 采用一个称为链接信息数组LINK(1:n)的数据结构，记录归并过程中A中的元素相对于其排序后在排序表中位置坐标的链接关系。
    - LINK(i)取值于[1, n]，是指向A的元素的指针：在排序表中它指向下一个元素在A中的位置坐标。0表示表的结束。

3.4 归并排序

■ 链接信息表

例：

i	1	2	3	4	5	6	7	8
Link[i]	6	4	7	1	3	0	8	0

- 该表中含有两个子表，0表示表的结束。
- 设置表头指针Q和R分别指向两个子表的起始处：

  Q=2，R=5；

则有，

表1：Q(2, 4, 1, 6)，经过排序后有：A(2)≤A(4) ≤A(1) ≤A(6)

表2：R(5, 3, 7, 8)，同样有：A(5)≤A(3) ≤A(7) ≤A(8)

3.4 归并排序

- 链接信息表
  - 将归并过程中元素在A和B之间移动的过程变成更改LINK所指示的链接关系，从而避免移动元素的本身。
  - 排序表可以通过LINK的表头指针和读取LINK中的链接关系取得。

3.4 归并排序

■ 改进后的归并排序模型

算法3.10 使用链接信息数组的归并排序模型

```
procedure MERGESORT1(low, high, p)
    //利用链接信息数组LINK(1:n)将全程数组A(low:high)按非降次序排序。LINK中值表示按排序次序给出A下标的表，并把p置于该表的开始处//
  global A(low: high), LINK(low, high)
  if high-low+1<16  //当集合中的元素个数足够少(<16)时，采用更有效的小规模集合上的排序算法直接排序//
    then call INSERTSORT1(A, LINK, low, high, p) //算法3.7的改进型//
    else mid←⌊(low + high)/2⌋
      call MERGESORT1(low, mid, q)        //返回q表//
      call MERGESORT1(mid+1, high, r)     //返回r表//
      call MERGE1(q, r, p) //将表q和表r归并成表p//
  endif
end MERGESORT1
```

3.4 归并排序

■ 改进后的归并排序模型

算法3.11 使用链接表归并已排序的集合

```
procedure MERGE1(q, r, p)                    //q和r是全程数组LINK(1:n)中两个表的指针。归并这两个表，得到一个由p所指示的新表。此表将A中的元素按非降次序排序。LINK(0)被定义//
   global n, A(1:n), LINK(1:n)
   local integer i, j, k
   i←q; j←r; k←0  //新表在LINK(0)处开始//
   while i≠0 and j≠0 do //当两表均非空时//
      if A(i) ≤ A(j) //找较小的关键字//
         then LINK(k) ←i; k←i; i←LINK(i)  //加一个新关键字到表中//
         else  LINK(k) ←j; k←j; j←LINK(j)  //加一个新关键字到表中//
      endif
   repeat
   if i=0 then LINK(k) = j   else  LINK(k) = i   endif
   p = LINK(0)
end MERGE1
```

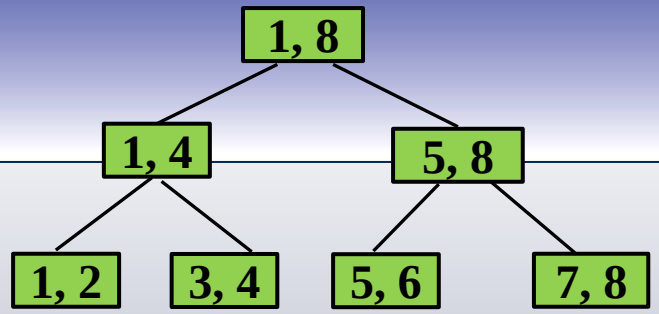
3.4 归并排序

- 改进后的归并排序模型
  - MERGESORT1的调用
    - 在初次调用时，待排序的n个元素放于A(1:n)；
    - LINK(1:n)初始化为0；
    - 初次调用：call MERGESORT1(1, n, p)；
    - p作为按排序次序给出A中元素的指示表的指针。

3.4 归并排序

- 改进后的归并排序模型
  - 改进归并排序算法示例
    - 设元素表:(50, 10, 25, 30, 15, 70, 35, 55)
    - 采用MERGESORT1对上述元素表排序(不做小规模集合的单独处理)
    - 下表给出在每一次调用MERGESORT1结束后，LINK数组的变化情况。

3.4 归并排序



i			0	1	2	3	4	5	6	7	8	
a[i]				50	10	25	30	15	70	35	55	
Link[i]			0	0	0	0	0	0	0	0	0	
q	r	p										
1	2	2	2	0	1	0	0	0	0	0	0	(10, 50)
3	4	3	3	0	1	4	0	0	0	0	0	(10, 50), (25, 30)
2	3	2	2	0	3	4	1	0	0	0	0	(10, 25, 30, 50)
5	6	5	5	0	3	4	1	6	0	0	0	(10, 25, 30, 50), (15, 70)
7	8	7	7	0	3	4	1	6	0	8	0	(10,25,30, 50),(15, 70),(35, 55)
5	7	5	5	0	3	4	1	7	0	8	6	(10, 25, 30, 50), (15, 35, 55, 70)
2	5	2	2	8	5	4	7	3	0	1	6	(10, 15, 25, 30, 35, 50, 55, 70)

在上表的最后一行，p=2返回，得到链表(2, 5, 3, 4, 7, 1, 8, 6)即：

A(2)≤ A(5)≤ A(3)≤ A(4)≤ A(7)≤ A(1)≤ A(8)≤ A(6)

3.4 归并排序

- 以比较为基础排序的时间下界
  - 排序的“实质”
    - 给定n个记录$R_1, R_2, \ldots, R_n$, 其相应的关键字是$k_1, k_2, \ldots, k_n$。
      排序就是确定1, 2, …, n的一种排列$P_1, P_2, \ldots, P_n$, 使得记录的关键字满足以下非递减(或非递增)排列关系

$$k_{P1} \le k_{P2} \le \ldots \le k_{Pn}$$

从而使n个记录成为一个按关键字有序的序列：

$$\{ R_{P1}, R_{P2}, \ldots, R_{Pn} \}$$

3.4 归并排序

- 以比较为基础排序的时间下界
  - 新归并算法在时间复杂度在最坏情况下仍为Ω(nlogn)
  - 从数量级的角度上看，归并排序算法是最坏情况的最优算法
  - 任何以关键字比较为基础的排序算法，其最坏情况下的时间下界都为：Ω(nlogn)

3.4 归并排序

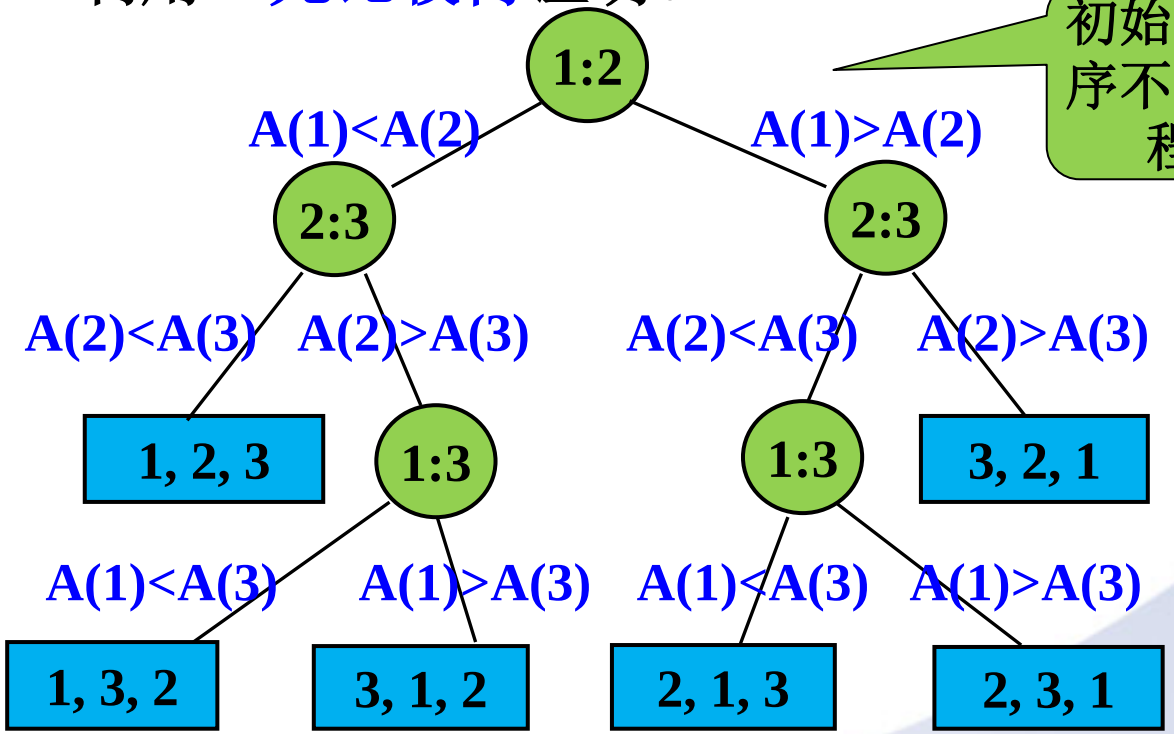
- 以比较为基础排序的时间下界
  - 利用二元比较树证明。
    - 假设参加排序的n个关键字A(1), A(2), …, A(n)互异。任意两个关键字的比较必导致A(i)<A(j)或A(i)>A(j)的结果。
    - 以二元比较树描述元素间的比较过程：

      若A(i)<A(j)，进入下一级的左分支

      若A(i)>A(j)，进入下一级的右分支
    - 算法在外部结点终止。

3.4 归并排序

- 以比较为基础排序的时间下界
  - 利用二元比较树证明。



  - 算法在外部结点终止。

3.4 归并排序

- 以比较为基础排序的时间下界
  - 利用二元比较树证明。
    - 从根到某外结点的路径代表某个特定输入情况下一种唯一的排序序列。
    - 路径长度表示生成该序列代表的排序表所需要的比较次数。
    - 而最长的路径代表算法在最坏情况下的执行情况，该路径的长度即是算法在最坏情况下所作的比较次数。
    - 故，以比较为基础的排序算法的最坏情况下界等于该算法对应的比较树的最小高度。

3.4 归并排序

- 以比较为基础排序的时间下界
  - 利用二元比较树证明。

    ①由于n个关键字有n!种可能的排列，所以二元比较树中将有n!个外部结点：每种排列对应于某种特定输入情况下的排序情况，每个外部结点表示一种可能的排序序列。

    ②设一棵二元比较树的所有内结点的级数均小于或等于k，则该树中最多有2^k个外结点。

    - 记算法在最坏情况下所作的比较次数为T(n)，则有T(n)=k：生成外结点所代表的排序序列所需的比较次数等于该外结点所在的级数-1；

3.4 归并排序

- 以比较为基础排序的时间下界
  - 利用二元比较树证明。

根据①和②的分析，有：

$$n! \leq 2^k = 2^{T(n)}$$

化简：

当n>1时，有$n! \geq n(n-1)(n-2)\ldots(\lceil n/2 \rceil) \geq (n/2)^{n/2}$

当n≥4时，有$T(n) \geq (n/2)\log(n/2) \geq (n/4)\log n$

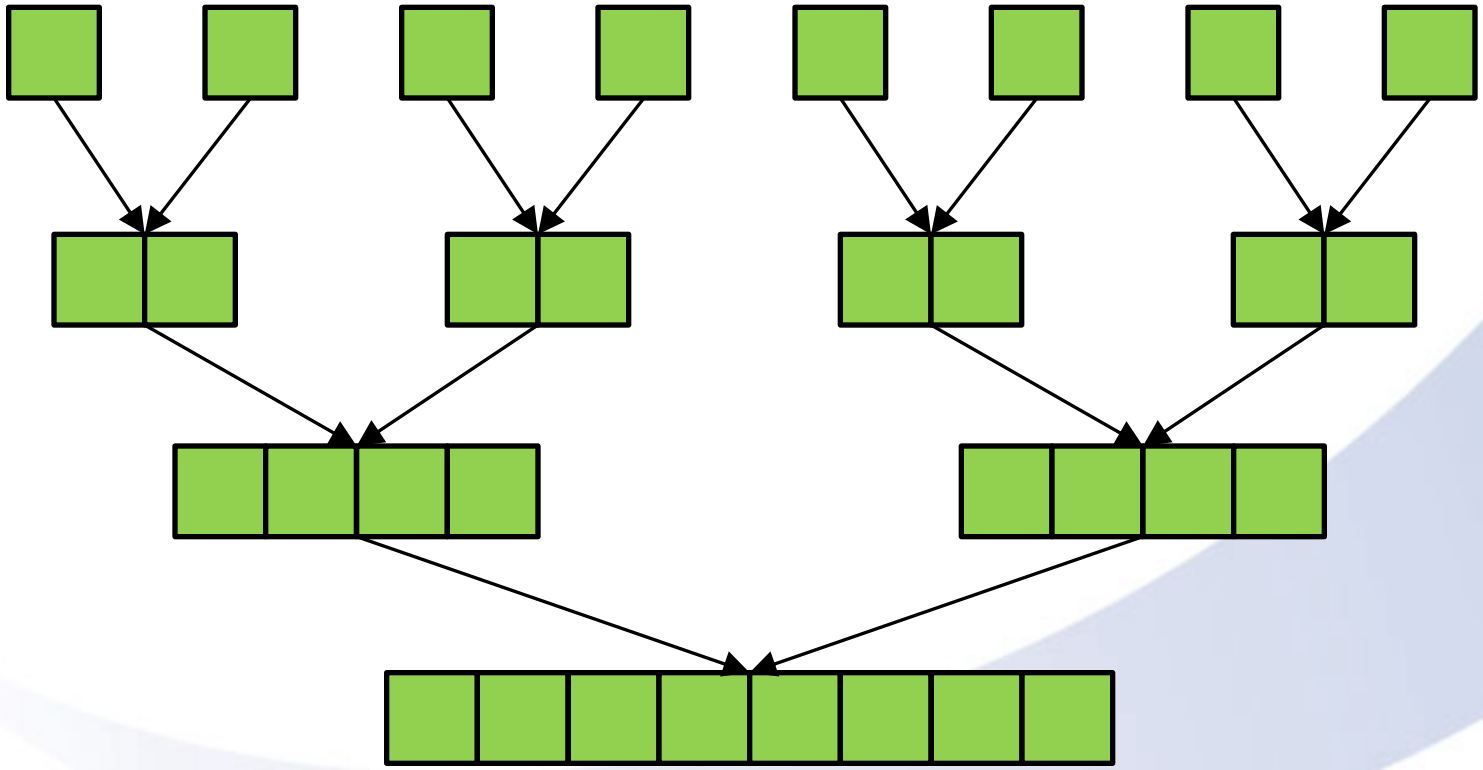
$$n^2 \geq 4n \Rightarrow n \geq 2n^{0.5}$$

故，任何以比较为基础的排序算法的最坏情况的时间下界为：

$$\Omega(n\log n)$$

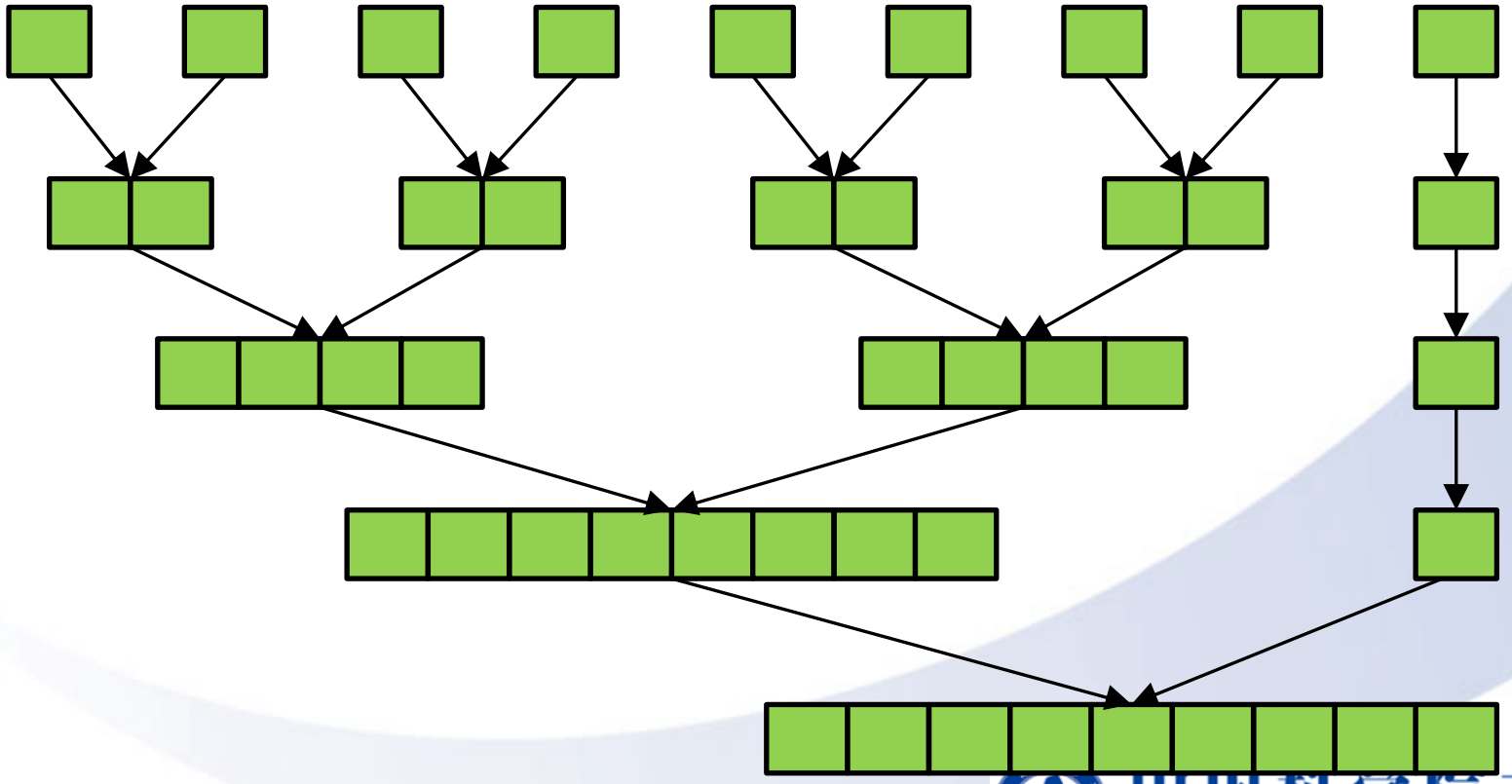
3.4 归并排序

- 归并排序的非递归算法
  - 由底向上的方法，将元素两两配对



3.4 归并排序

- 归并排序的非递归算法
  - 由底向上的方法，将元素两两配对



3.4 归并排序

■ 归并排序的非递归算法：C语言实现

```
public static void MergeSort(int n, int DataLength)
{   //n为待合并数据个数
   int i, t; i=1;     //循环计数变量
   while(i<=(n-2*DataLength+1)) //还有两段长度为DataLength的list可合并
   {  Merge(i, i+DataLength-1, i+2*DataLength-1);
      i=i+2*DataLength;  }
   if(i+DataLength<n)
   {//合并两段list，一段长度为DataLength，另一段长度不足DataLength
      Merge(i, i+DataLength-1, n); }
   else
   {//将剩下一段长度不足DataLength的list中的值不变}
}
```

例：要排序的数据如下

i	1	2	3	4	5	6	7	8	9	10
a[i]	9	8	7	6	5	4	3	2	1	0

步骤1：length=1

i	1	2	3	4	5	6	7	8	9	10
a[i]	8	9	6	7	4	5	2	3	0	1

步骤2：length=2

i	1	2	3	4	5	6	7	8	9	10
a[i]	6	7	8	9	2	3	4	5	0	1

步骤3：length=4

i	1	2	3	4	5	6	7	8	9	10
a[i]	2	3	4	5	6	7	8	9	0	1

步骤4：length=8

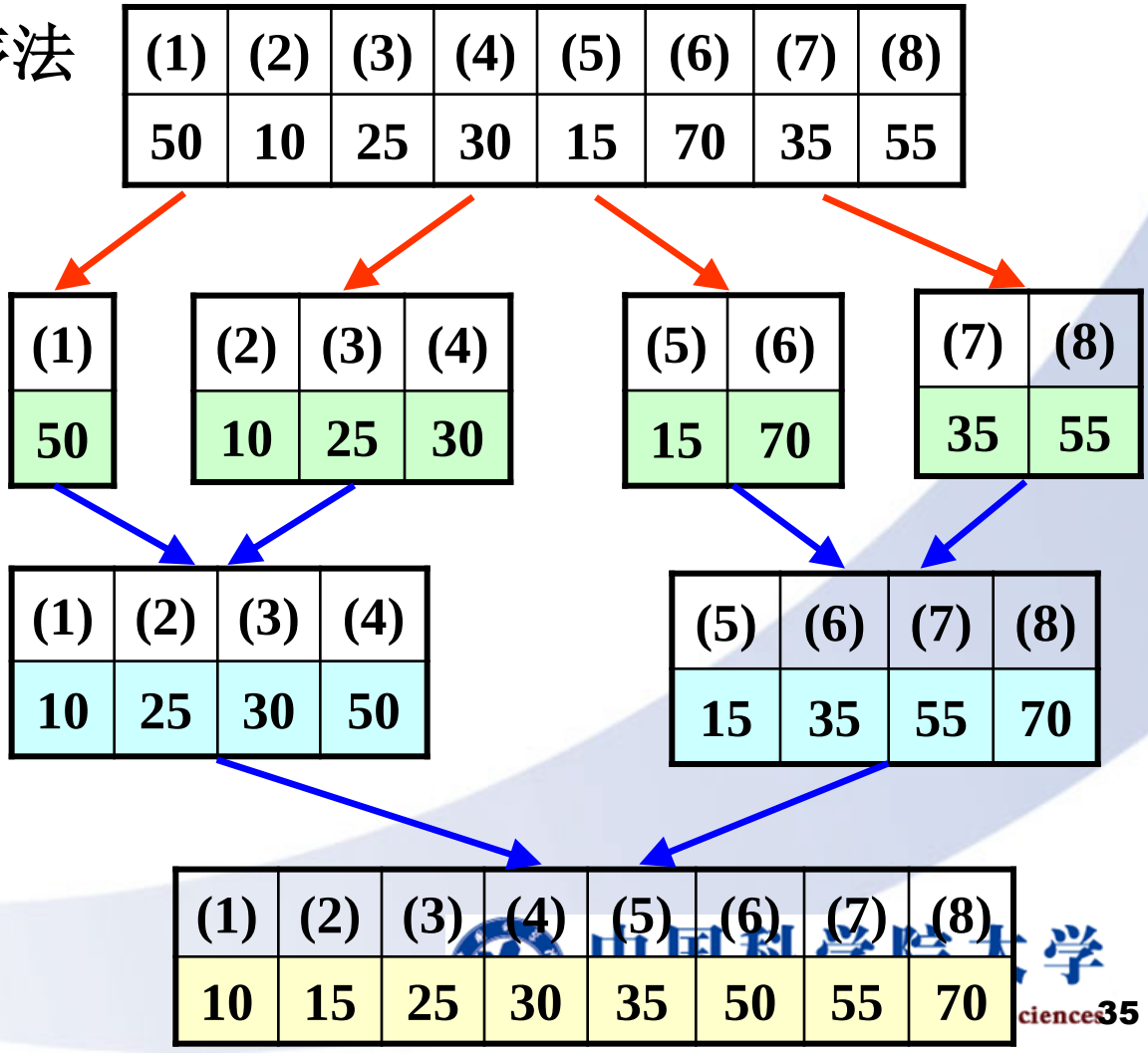
i	1	2	3	4	5	6	7	8	9	10
a[i]	0	1	2	3	4	5	6	7	8	9

3.4 归并排序

- 归并排序的非递归算法
  - 自然合并排序法

首先对数组A进行一遍扫描, 找出所有已排序的子数组段

然后将相邻的子数组段两两合并, 重复这种合并过程, 直到整个数组排好序



3.4 归并排序

■ 冒泡排序

- 冒泡排序和气泡在水中不断往上冒的情况有些类似。气泡大的（大的数据）在下面，气泡小的（小的数据）在上面。
- 基本原理
  - 对存放原始数据的数组，按从前往后的方向进行多次扫描，每次扫描称为一趟。
  - 当发现相邻两个数据的次序与排序要求的大小次序不符合时，即将这两个数据进行互换。
  - 这样，较小的数据就会逐个向前移动，好象气泡向上浮起一样。

3.4 归并排序

- 冒泡排序
  - 例：
    - 用冒泡排序的方法将下面一组无序数组排成从小到大的顺序。
    - { 49, 38, 65, 97, 76, 13, 27, 49 }

原数据和序号

序号	1	2	3	4	5	6	7	8
数据	49	38	65	97	76	13	27	49

第一趟排序的步骤：

序号	1	2	3	4	5	6	7	8
数据	38	49	65	76	13	27	49	97

49>38，交换位置；49<65，保持不变；65<76，交换位置；76>13，交换位置；76>27，交换位置；76>49，交换位置；97>76，交换位置

第一趟排序，一共进行了多少次比较？

经过第一趟排序，把最大的数沉到最底了！

第一趟排序后的数据和序号

序号	1	2	3	4	5	6	7	8
数据	38	49	65	76	13	27	49	97

第二趟排序的步骤：

序号	1	2	3	4	5	6	7	8
数据	38	49	65	13	27	49	76	97

38<49，保持不变；49<65，保持不变；65>13，交换位置；65>27，交换位置；65<49，保持不变；49<76，保持不变

经过第二趟排序，把第二大的数沉到倒数第二个位置了！

观察原数据与第一、二趟排序后的数据

序号	1	2	3	4	5	6	7	8
数据	49	38	65	97	76	13	27	49

序号	1	2	3	4	5	6	7	8
数据	38	49	65	76	13	27	49	97

序号	1	2	3	4	5	6	7	8
数据	38	49	65	13	27	49	76	97

问：那么我们预计最多一共要经过多少次排序呢？

初始

序号	1	2	3	4	5	6	7	8
数据	49	38	65	97	76	13	27	49

1趟

序号	1	2	3	4	5	6	7	8
数据	38	49	65	76	13	27	49	97

2趟

序号	1	2	3	4	5	6	7	8
数据	38	49	65	13	27	49	76	97

3趟

序号	1	2	3	4	5	6	7	8
数据	38	49	13	27	49	65	76	97

4趟

序号	1	2	3	4	5	6	7	8
数据	38	13	27	49	49	65	76	97

5趟

序号	1	2	3	4	5	6	7	8
数据	13	27	38	49	49	65	76	97

6趟

序号	1	2	3	4	5	6	7	8
数据	13	27	38	49	49	65	76	97

7趟

序号	1	2	3	4	5	6	7	8
数据	13	27	38	49	49	65	76	97

3.4 归并排序

■ 冒泡排序

```
void BubbleSort(int a[  ], int n)
{   //对表a做冒泡排序。
    int i, j, t;
    for(i=0;  i<n;  i++) {
          for(j=0;  j<n-i-1;  j++) {
               if( a[j]> a[j+1])
                 {  temp =a[j];  a[j] = a[j+1]; a[j+1] = temp;  }
           }
       }
  }
```

作业-课后练习5

- 对下面的三组数据进行归并排序
  - (45, 36, 18, 53, 72, 30, 48, 93, 15, 36)
  - (1, 1, 1, 1, 1)
  - (5, 5, 8, 3, 4, 3, 2)
  - 给出每一次归并后的结果。

作业-课后练习6

- 稳定排序:
  - 排序前两个相等的数其在序列的前后位置顺序和排序后它们两个的前后位置顺序相同。即，如果$A_i = A_j$，A_i原来在位置前，排序后A_i还是要在A_j位置前。
- 归并算法是稳定的算法吗？
  - 回答该问题，并给出理由

小结