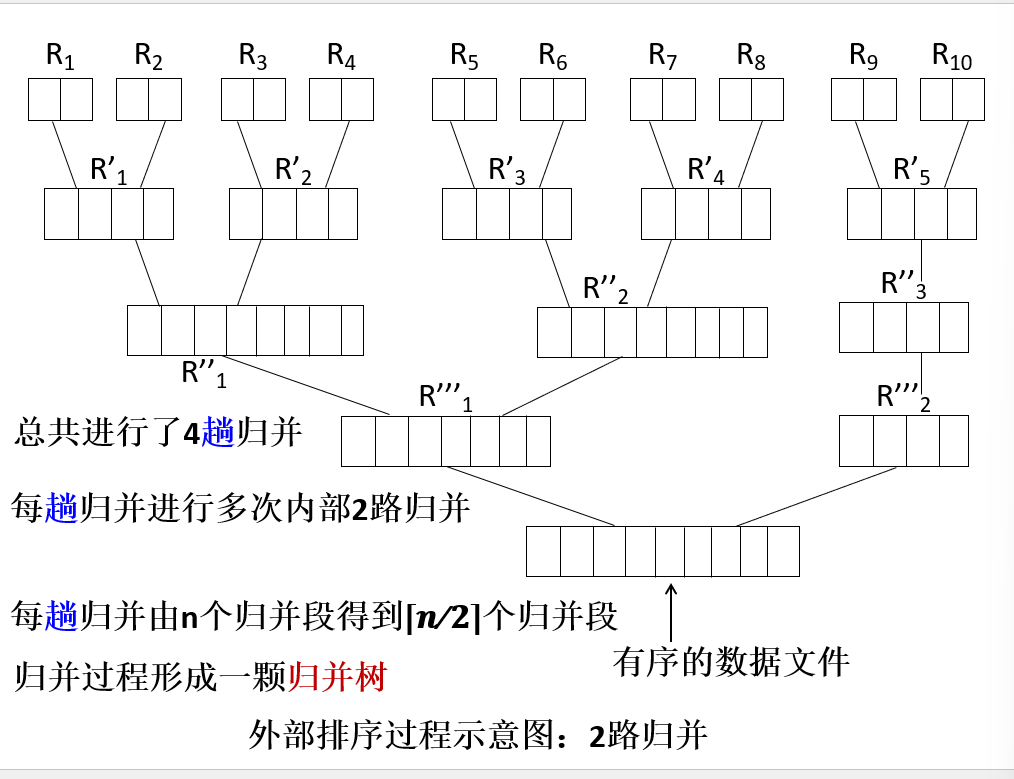
**1. 外排序**

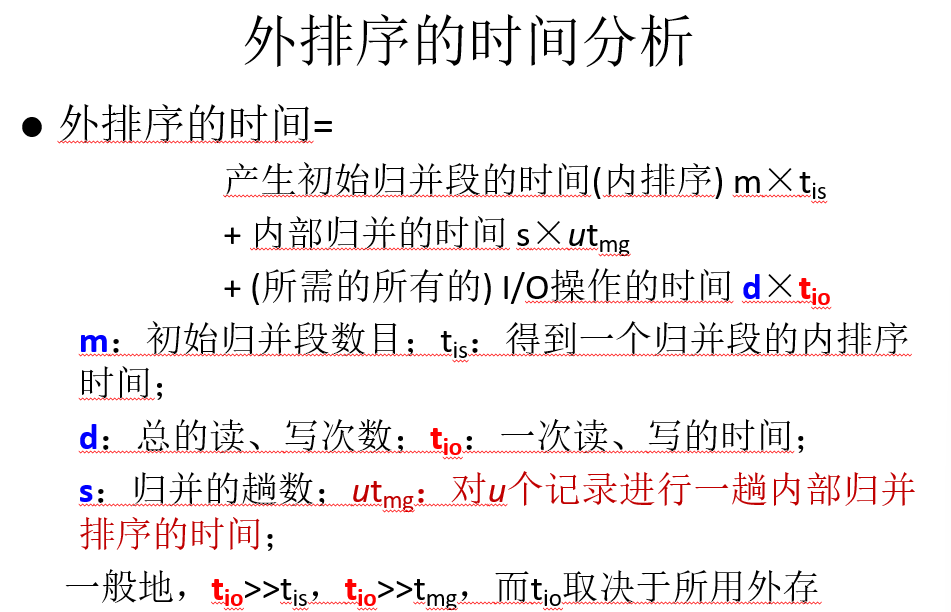
方法：基于磁盘进行的排序多使用归并排序

过程：归并排序的基本过程分为两个阶段：

第一步：建立用于外排序的内存缓冲区，并根据它们的大**小将输入文件划分为若干段**，用某种内排序方法**对各段进行排序**。经过排序的段叫做**初始归并段** (Run)。当它们生成后就被写到外存中去

第二步：**把第一步生成的初始归并段加以归并(内部归并)**，一趟趟扩大归并段和减少归并段数，直到最后归并成一个大归并段为止





d=(S+1)×总块数

I/O次数=(归并趟数+1)×总块数

影响外排序效率的主要原因是内、外存之间数据交换(读、写外存)

**提高外排序效率的方法**

对m个初始归并段，进行k-路平衡归并，归并的趟数s为logk𝑚

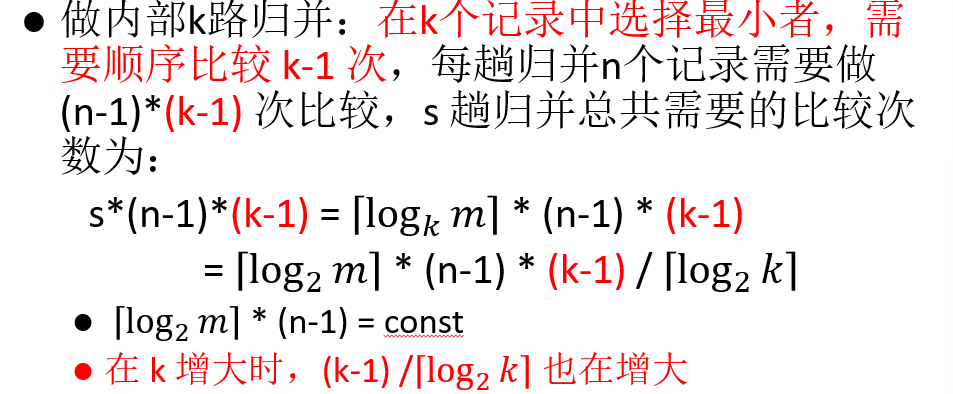
增大k：进行多路归并，减少文件归并的趟数

减小m：增加归并段的长度，减少初始归并的数目

根据不同归并段的长度，采取最佳归并方案

**2. k路平衡归并 (k-way balanced merging**

做 k 路平衡归并时, 如果有 m 个初始归并段, 则相应的归并树有 logk𝑚 +1 层, 需要归并logk𝑚 趟

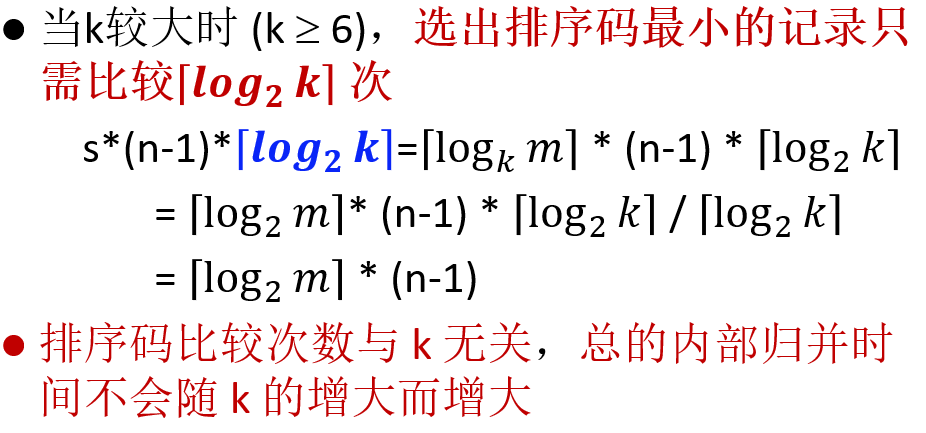


增大归并路数 k，会使得内部归并的时间增大

**内部归并的改进：引入败者树**

“败者树(tree of losers)”从k个归并段中选最小者

当k较大时 (k ≥ 6)，选出排序码最小的记录只需比较𝒍𝒐𝒈𝟐𝒌 ⌉次

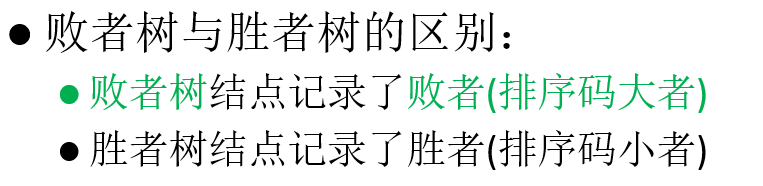


**败者树**

败者树是一棵完全二叉树

每个**叶结点**存放各归并段在归并过程中**当前参加比较的记录**；

每个**非叶结点**存放败者即它**两个子女结点中排序码大**的结点；



**数据结构**

败者树的非叶结点：（k: 归并路数）

Typedef int LoserTree[k];

LoserTree ls[]，

记录的排序码指向败者，但用于下一次比较的是上一次的胜者

败者树的叶(外)结点：仅存放待归并记录的关键字

Typedef struct { keyType key; } ExNode, **External[k+1]**;

**External** &**b**；// 叶节点数组，每个位置对应一个归并段。b[0]到b[k-1]存放各归并段当前参加归并的记录的排序码（个归并段每次取一个参加比赛）

1. ​​首次构建树​​（找出最小值）

两两比较叶子节点（如3 vs 5 → 败者5记入父节点）。

胜者继续向上比较（如3 vs 1 → 败者3记入父节点）。

最终根节点记录最小值来源（如序列3的1是冠军）

2. ​​更新最小值后调整​​：

移出冠军1，序列3补入新元素（如6）。

从序列3的叶子节点向上调整：

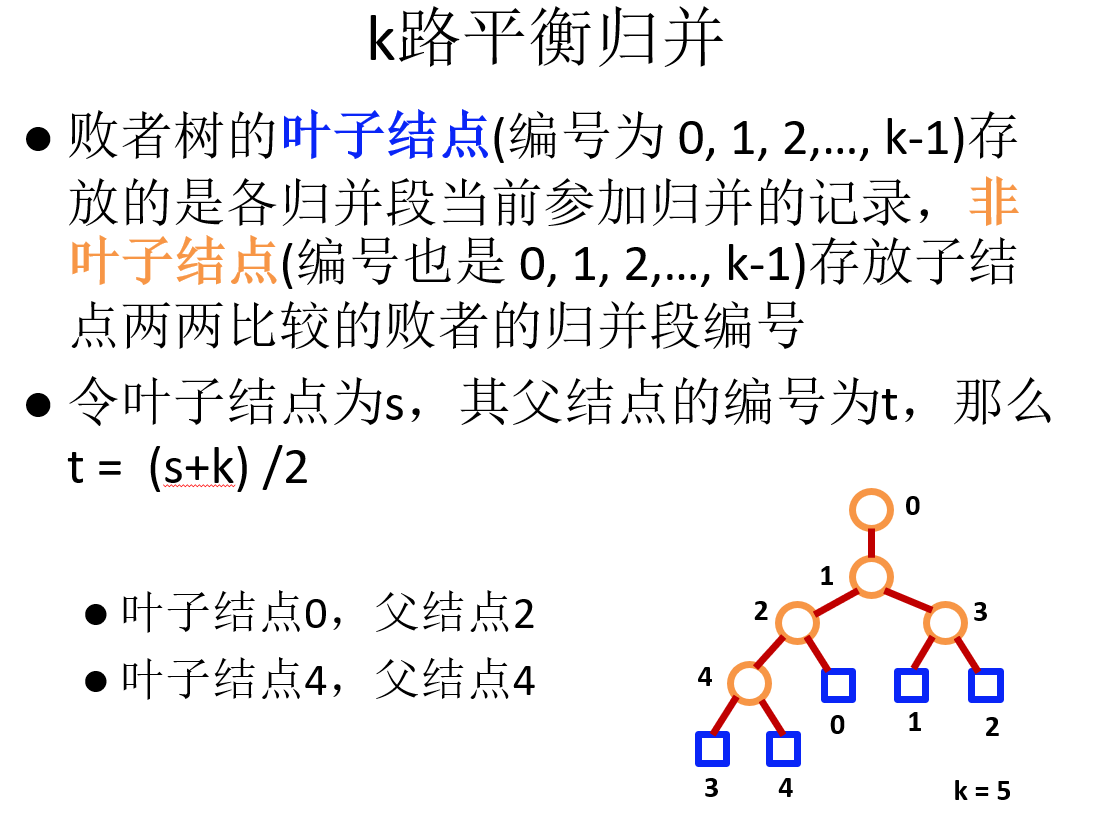
**与父节点记录的败者比较**（如6 vs 父节点记录的败者3），**胜者（3）晋级，败者（6）更新到父节点**。

重复至根节点，更新全局胜者（如新冠军是序列1的3）。

​​效率​​：只需比较路径上的节点（4路归并只需2次比较），而非全树。

排序算法





**3. 置换-选择排序**

减少初始归并段个数m

在总记录数n一定时，要减少m，必须增大初始归并段长度

(初始归并段的生成) 为了突破等长归并段的限制，可采用败者树来生成初始归并段。在使用同样大的内存工作区的情况下，可以生成平均比原来等长情况下大一倍的初始归并段，从而减少初始归并段个数

选择和置换过程的步骤

1. 从输入文件FI中把 k 个记录读入内存中的数组r，并构造败者树
   * 内存中存放记录的数组r可容纳的记录个数为k
2. 利用败者树在r中**选择**一个排序码最小的记录在r[q]，其排序码存入LastKey作为门槛，**以后再选出的排序码比它大的记录归入本归并段，比它小的归入下一归并段**
   * q是败者树叶结点序号
3. 将此r[q]记录写到输出文件FO中
4. 若FI未读完，则从FI读入下一个记录，**置换**r[q]及败者树中的key[q]
5. 调整败者树，从**所有排序码比LastKey大**的记录中**选择**一个排序码最小的记录r[q]作为门槛，其排序码存入LastKey
6. 重复(3)-(5)，直到在败者树中**选不出排序码比LastKey大的记录**为止。此时，在输出文件FO中得到一个初始归并段，在它最后加一个归并段结束标志
7. 重复(2)-(6) ，重新开始选择和置换，产生新的初始归并段，直到输入文件FI中所有记录选完为止

注：在利用败者树生成**不等长初始归并段**的算法和调整败者树并选出最小记录的算法中，用两个条件来决定谁为败者，谁为胜者:

* + 首先比较两个记录所在归并段的**段号**，段号小者为胜者，**段号**大者为败者
  + 在归并段的段号相同时，排序码小者为胜者，**排序码**大者为败者

比较后，把败者记录在记录数组r中的序号记入它的父结点中，把胜者记录在记录数组r中的序号记入工作单元 s 中，向更上一层进行比较，最后的胜者记入 loser[0]中

在一般情况下，若输入文件有 n 个记录，生成初始归并段的时间开销是O(nlog2𝑘)，因为每输出一个记录，对败者树进行调整需要时间为O(log2𝑘)。

**4. 最佳归并树**

归并树是描述归并过程的k叉树(正则 k 叉树)

使用带权值的归并树：

**叶结点上的权值**为该**初始归并段中的记录个数**

各非叶结点代表归并出来的新归并段

根结点代表最终生成的归并段

**叶结点到根结点的路径长度表示在归并过程中的读记录次数**

归并树的带权路径长度 WPL 即为归并过程中的总读记录数，因而，在归并过程中总的读写记录次数为 2\*WPL

最佳归并树：归并树的带权路径长度最小

**为使归并树成为一棵正则多叉树（节点度数要么为0要么为k），可能需要补入空归并段**

可将Huffman树的思想扩充到 k叉树的情形：在归并树中，**让记录个数少的初始归并段最先归并**，**记录个数多的初始归并段最晚归并**，就可建立总读写次数达到最少的最佳归并树

（归并树是倒置的树，从叶节点的初始归并段出发，后归并的离根结点更近）

