

## 8.1 排序的基本概念

### 排序的定义

排序：就是重新排列表中的元素,使表中的元素满足按关键字有序的过程

算法的稳定性:若待排序表中有两个元素 $R_i$ 和 $R_j$ 其对应的关键字相同即 $key_i = key_j$ ,且在排序前  $R_i$ 在  $R_j$ 的前面,若使用某一排序算法排序后, $R_i$ 仍然在 $R_j$ 的前面,则排序算法是稳定的

根据数据是否在内存中进行分类

内部排序：在排序期间元素全部存放在内存中的排序

外部排序：在排序期间元素无法全部同时存放在内存中,必须在排序的过程中根据要求不断地在内、外存之间移动的排序

分类

基本类型

插入排序

交换排序

选择排序

归并排序

基数排序

## 8.2 插入排序



注: 仅供王道VIP学员使用 严禁外部传播!

## 8.3 交换排序

### 基本概述

根据序列中两个元素关键字的比较结果来对换这两个记录在序列中的位置

考试涉及范围

冒泡排序

快速排序

### 冒泡排序

基本思想

从后往前（或从前往后）两两比较相邻元素的值，若为逆序（即 $A[i-1] > A[i]$ ），则交换它们，直到序列比较完

将最小的元素交换到待排序列的第一个位置（或将最大的元素交换到待排序列的最后一个位置）

进行下一趟冒泡时，前一趟确定的最小元素不再参与比较，每趟冒泡的结果是把序列中的最小元素（或最大元素）放到了序列的最终位置

如果某一趟排序过程中未发生“交换”，则算法可提前结束

空间复杂度： $O(1)$

性能分析

时间效率

最好情况

时间复杂度为 $O(n)$

最坏情况

时间复杂度 $O(n^2)$

平均情况

冒泡排序时间复杂度为 $O(n^2)$

稳定算法

适用于顺序存储

### 快速排序被认为是目前基于比较的内部排序方法中最好的方法

### 快速排序

基本思想

首先选取一个元素作为枢纽，然后以此枢纽为界分为两个部分，左面小于该枢纽值，右面大于该枢纽值

然后再对这两个部分分别递归的进行上述步骤

性能分析

空间复杂度

最好情况  $O(\log_2 n)$

最坏情况  $O(n)$

平均情况  $O(\log_2 n)$

时间效率

快速排序的运行时间与划分是否对称有关

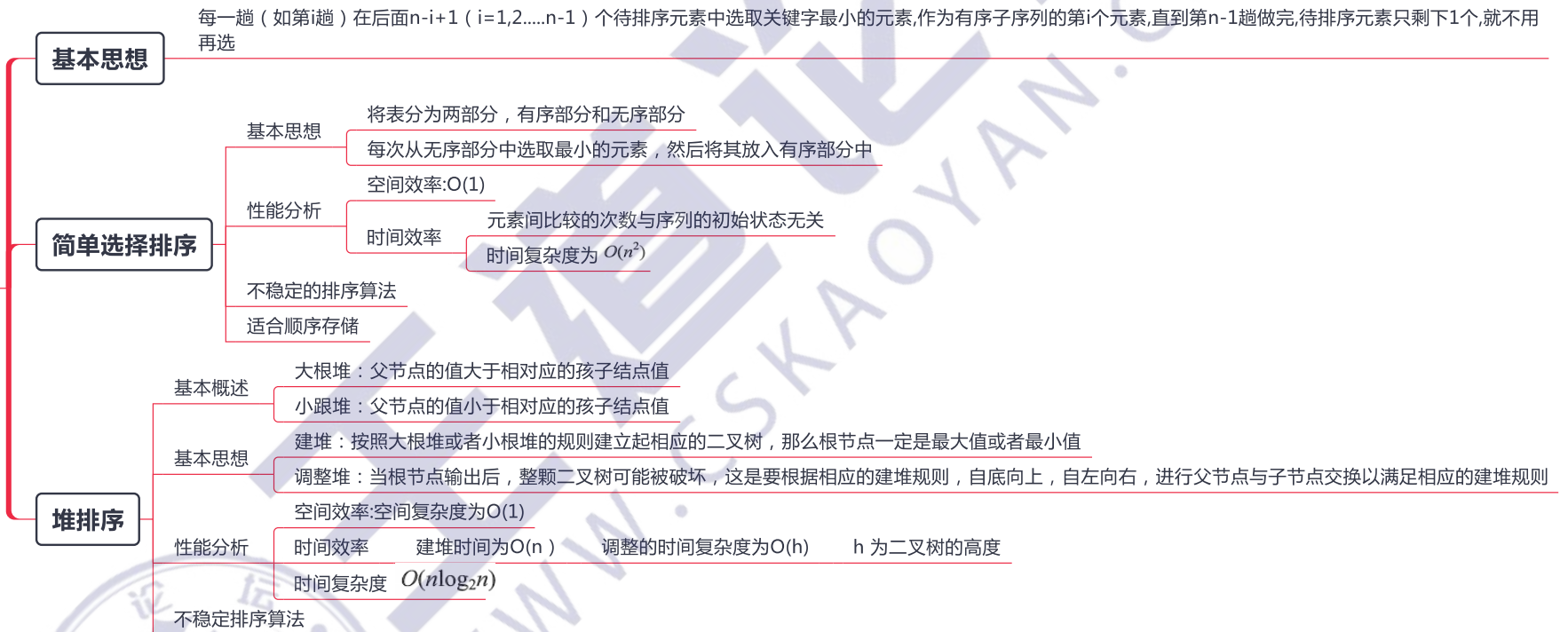
对应于初始排序表基本有序或基本逆序时，就得到最坏情况下的时间复杂度为 $O(n^2)$

快速排序的时间复杂度为 $O(n^2)$

不稳定算法

适用于顺序存储

## 8.4 选择排序

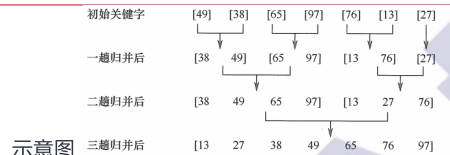


## 8.5 归并排序和基数排序

### 归并排序

基本思想

每次选定相应的元素分别合成一个新的有序表



2路归并——二合一

k路归并——k合一

性能分析

空间复杂度为 $O(n)$

时间复杂度为 $O(n\log_2 n)$

稳定排序算法

适用于顺序表

排序思想

最高位优先 (MSD) 法: 按关键字位权重递减依次逐层划分成若干更小的子序列, 最后将所有子序列依次连接成一个有序序列

最低位优先 (LSD) 法: 按关键字权重递增依次进行排序, 最后形成一个有序序列

### 基数排序

性能分析

一趟排序需要的辅助存储空间为 $r$  (r个队列: r个队头指针和r个队尾指针)

空间效率 基数排序的空间复杂度为 $O(r)$

时间效率 基数排序需要进行 $d$ 趟分配和收集, 一趟分配需要 $O(n)$ , 一趟收集需要 $O(r)$

基数排序的时间复杂度为 $O(d(n+r))$  与序列的初始状态无关

稳定排序算法

算法种类	时间复杂度			空间复杂度	是否稳定
	最好情况	平均情况	最坏情况		
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	是
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	是
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	否
希尔排序				$O(1)$	否
快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(\log_2 n)$	否
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	否
2路归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	是
基数排序	$O(d(n+r))$	$O(d(n+r))$	$O(d(n+r))$	$O(r)$	是

### 8.6 各种内部排序算法的比较及应用

若 $n$ 较小, 可采用直接插入排序或简单选择排序

当记录本身信息量较大时, 用简单选择排序较好

若文件的初始状态已按关键字基本有序, 则选用直接插入或冒泡排序为宜

快速排序被认为是目前基于比较的内部排序方法中最好的方法 待排序的关键字随机分布时, 快速排序的平均时间最短

### 排序算法小结

若 $n$ 较大, 则应采用时间复杂度为 $O(n\log_2 n)$ 的排序方法: 快速排序、堆排序或归并排序

要求排序稳定且时间复杂度为 $O(n\log_2 n)$  则可选用归并排序

若 $n$ 很大, 记录的关键字位数较少且可以分解时, 采用基数排序较好

当记录本身信息量较大时, 为避免耗费大量时间移动记录, 可用链表作为存储结构



## 8.7 外部排序（上）

### 外部排序的基本概念

对大文件进行排序,因为文件中的记录很多、信息量大,无法将整个文件复制进内存中进行排序

需要等待排序的记录存储在外存上,排序时再把数据一部分一部分地调入内存进行排序,在排序过程中需要多次进行内存和外存之间的交换

### 外部排序的方法

#### 基本概念

文件通常是按块存储在磁盘上的,操作系统也是按块对磁盘上的信息进行读写的

外部排序过程中的时间代价主要考虑访问磁盘的次数,即 I/O 次数

#### 外部排序通常采用归并排序法

#### 算法实现的两个阶段

据内存缓冲区大小,将外存上的文件分成若干长度为  $r$  的子文件,依次读入内存并利用内部排序方法对它们进行排序,并将排序后得到的有序子文件重新写回外存（归并段或顺串）

对这些归并段进行逐趟归并,使归并段（有序子文件）逐渐由小到大,直至得到整个有序文件为止

#### 耗时间

外部排序的总时间 = 内部排序所需的时间 + 外存信息读写的时间 + 内部归并所需的时间

#### 归并排序优化

增大归并路数  $k$

减少初始归并段个数  $r$

都能减少归并趟数  $s$ ,进而减少读写磁盘的次数,达到提高外部排序速度的目的

### 多路平衡归并与败者树

#### 引入败者树的背景

为了使内部归并不受  $k$ （归并路数）的增大的影响

#### 基本思想

败者树是树形选择排序的一种变体,可视为一棵完全二叉树

$k$  个叶结点分别存放  $k$  个归并段在归并过程中当前参加比较的记录,内部结点用来记忆左右子树中的“失败者”,而让胜者往上继续进行比赛,一直到根结点

若比较两个数,大的为失败者、小的为胜利者,则根结点指向的数为最小数

#### 性能分析

$k$  路归并的败者树深度  $\lceil \log_2 k \rceil$

总的比较次数  $S(n-1) \lceil \log_2 k \rceil = \lceil \log_2 r \rceil (n-1) \lceil \log_2 k \rceil = (n-1) \lceil \log_2 r \rceil$

#### 注意

归并路数  $k$  并不是越大越好。归并路数  $k$  增大时,相应地需要增加输入缓冲区的个数

当  $k$  值过大时,虽然归并趟数会减少,但读写外存的次数仍会增加

#### 优化

增加归并路数  $k$ , 进行多路平衡归并

代价1: 需要增加相应的输入缓冲区

代价2: 每次从  $k$  个归并段中选一个最小元素需要  $(k-1)$  次关键字对比

减少初始归并段数量  $r$

## 8.7 外部排序（下）

### 置换-选择排序（生成初始归并段）

实现过程

- 设初始待排文件为FI,初始归并段输出文件为FO,内存工作区为WA,FO和WA的初始状态为空,WA 可容纳  $w$  个记录
- 1) 从FI输入  $w$  个记录到工作区 WA
  - 2) 从 WA 中选出其中关键字取最小值的记录,记为 MINIMAX 记录
  - 3) 将 MINIMAX 记录输出到 FO 中去
  - 4) 若 FI 不空,则从 FI 输入下一个记录到 WA 中
  - 5) 从 WA 中所有关键字比 MINIMAX 记录的关键字大的记录中选出最小关键字记录,作为新的 MINIMAX 记录
  - 6) 重复 3) ~ 5), 直至 WA 中选不出新的 MINIMAX 记录为止,由此得到一个初始归并段,输出一个归并段的结束标志到 FO 中去
  - 7) 重复 2) ~ 6), 直至 WA 为空。由此得到全部初始归并段

结构概述

各叶结点表示一个初始归并段,上面的权值表示该归并段的长度

叶结点到根的路径长度表示其参加归并的趟数

各非叶结点代表归并成的新归并段

根结点表示最终生成的归并段

树的带权路径长度 WPL 为归并过程中的总读记录数

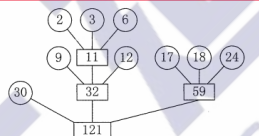
引入哈夫曼树的思想

在归并树中,让记录数少的初始归并段最先归并,记录数多的初始归并段最晚归并,就可以建立总的 I/O 次数最少的最佳归并树

算法优化

### 最佳归并树

示意图

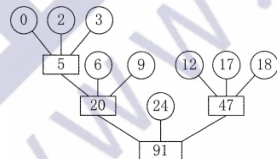


若初始归并段不足以构成一棵严格  $k$  叉树时,需添加长度为 0 的"虚段"

按照哈夫曼树的原则,权为 0 的叶子应离树根最远

算法修正

示意图



设度为 0 的结点有  $n_0 (=n)$  个,度为  $k$  的结点有  $n_k$  个

严格  $k$  叉树有  $n_0 = (k-1)n_k + 1$  变形可得  $n_k = (n_0 - 1) / (k-1)$

需要修正的条件

$(n_0 - 1) \% (k-1) = 0$  说明正好可以构造  $k$  叉归并树

$(n_0 - 1) \% (k-1) = u \neq 0$  再加上  $k-u-1$  个空归并段,就可以建立归并树