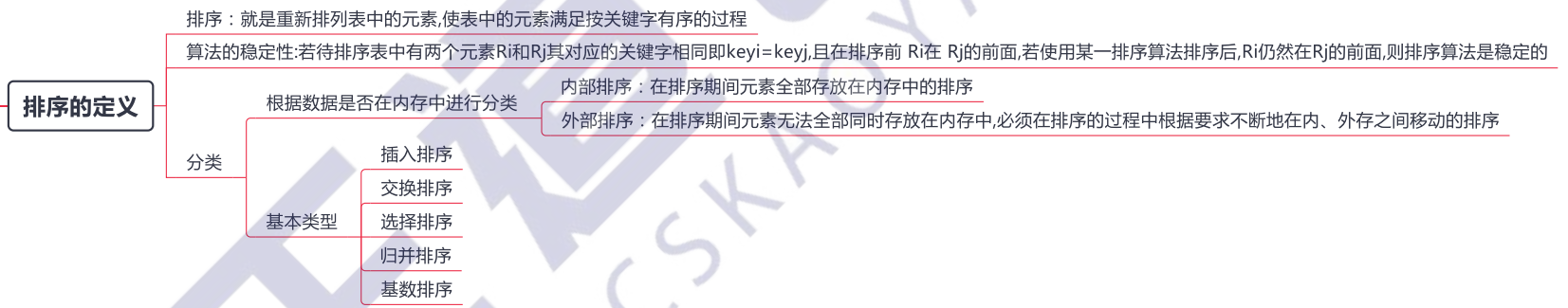
8.1排序的基本概念



8.2 插入排序

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
  - 性能分析
    - 空间复杂度：O(1)
    - 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，然后以此枢轴为界分为两个部分，左面小于该枢轴值，右面大于该枢轴值
    - 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

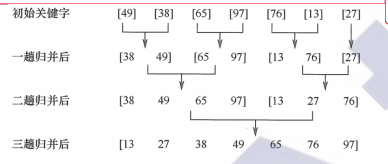
8.4选择排序

- **基本思想**
  - 每一趟（如第i趟）在后面n-i+1（i=1,2.....n-1）个待排序元素中选取关键字最小的元素,作为有序子序列的第i个元素,直到第n-1趟做完,待排序元素只剩下1个,就不用再选
- **简单选择排序**
  - 基本思想
    - 将表分为两部分，有序部分和无序部分
    - 每次从无序部分中选取最小的元素，然后将其放入有序部分中
  - 性能分析
    - 空间效率:O(1)
    - 时间效率
      - 元素间比较的次数与序列的初始状态无关
      - 时间复杂度为 $O(n^2)$
  - 不稳定的排序算法
  - 适合顺序存储
- **堆排序**
  - 基本概述
    - 大根堆：父节点的值大于相对应的孩子结点值
    - 小跟堆：父节点的值小于相对应的孩子结点值
  - 基本思想
    - 建堆：按照大根堆或者小根堆的规则建立起相应的二叉树，那么根节点一定是最大值或者最小值
    - 调整堆：当根节点输出后，整颗二叉树可能被破坏，这是要根据相应的建堆规则，自底向上，自左向右，进行父节点与子节点交换以满足相应的建堆规则
  - 性能分析
    - 空间效率:空间复杂度为O(1)
    - 时间效率　建堆时间为O(n）　调整的时间复杂度为O(h)　h 为二叉树的高度
    - 时间复杂度 $O(n\log_2 n)$
  - 不稳定排序算法

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
  - 示意图



- 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

基数排序

- 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
- 性能分析
  - 空间效率
    - 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）
    - 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

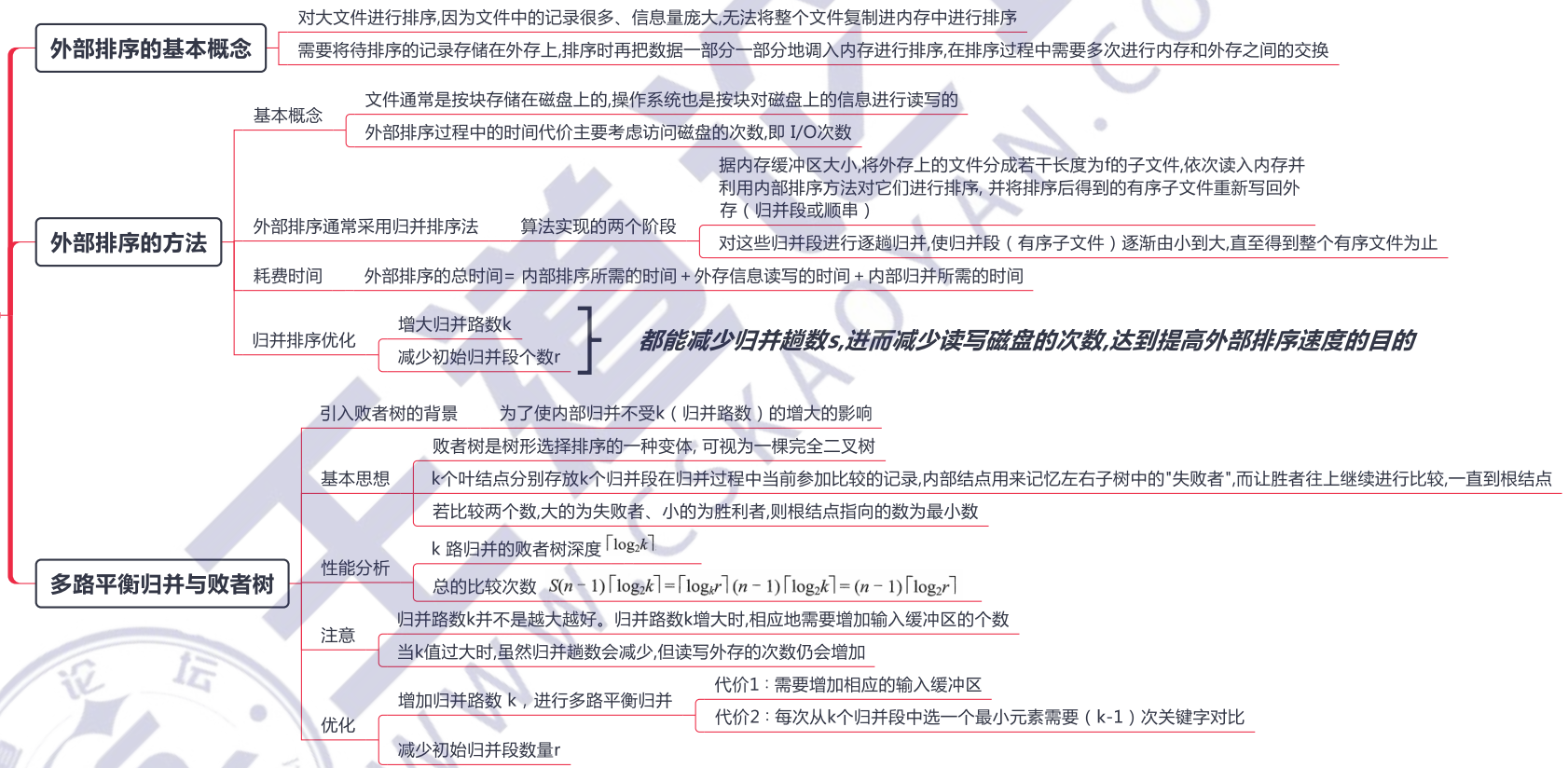
算法种类	时间复杂度			空间复杂度	是否稳定
	最好情况	平均情况	最坏情况		
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	是
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	是
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	否
希尔排序				$O(1)$	否
快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(\log_2 n)$	否
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	否
2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

排序算法小结

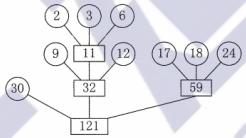
- 若n较小,可采用直接插入排序或简单选择排序
- 当记录本身信息量较大时,用简单选择排序较好
- 若文件的初始状态已按关键字基本有序,则选用直接插入或冒泡排序为宜
- 快速排序被认为是目前基于比较的内部排序方法中最好的方法　　待排序的关键字随机分布时,快速排序的平均时间最短
- 若n较大,则应采用时间复杂度为 $O(n\log_2 n)$ 的排序方法:快速排序、堆排序或归并排序
- 要求排序稳定且时间复杂度为 $O(n\log_2 n)$ 则可选用归并排序
- 若n很大,记录的关键字位数较少且可以分解时,采用基数排序较好
- 当记录本身信息量较大时,为避免耗费大量时间移动记录,可用链表作为存储结构

8.7 外部排序（上）

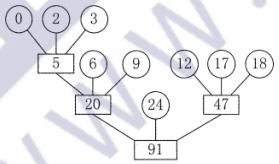


8.7 外部排序（下）

- **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）,直至在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
- **最佳归并树**
  - 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
  - 算法优化
    - 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
    - 示意图



  - 算法修正
    - 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
    - 示意图



    - 需要修正的条件
      - 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 $n_k=(n_0-1)/(k-1)$
      - $(n_0-1)\%(k-1)=0$ 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