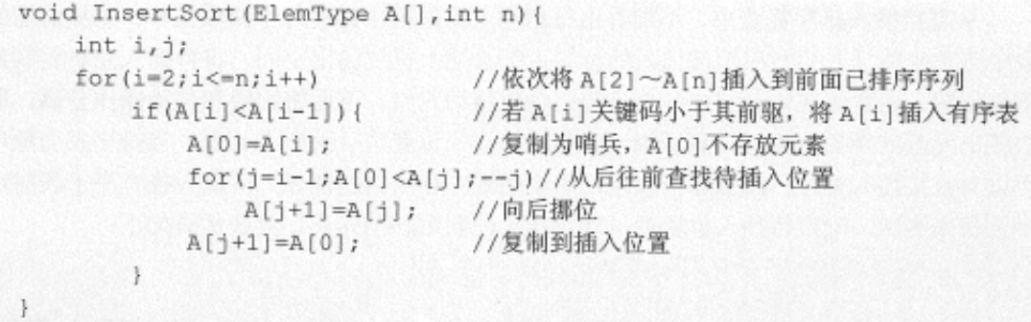
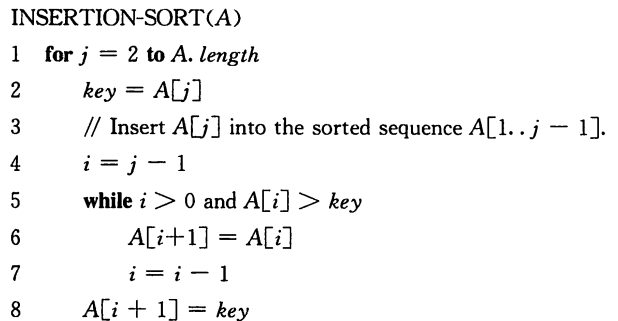
1. 排序
2. 排序的基本概念
3. 排序：见算法导论
4. 算法的稳定性：关键字相同的元素在排序之后相对位置不变；
5. 内部排序和外部排序：①内部排序是指排序期间元素全部放在内存中的排序（关注算法的时空复杂度）；②外部排序是指无法全部同时在内存中，排序时需要换入换出（还要使读写磁盘的次数最少）；一般情况下，内部排序算法在执行过程中都要进行两种操作，比较和移动（复制？）。
6. 插入排序
7. 直接插入排序



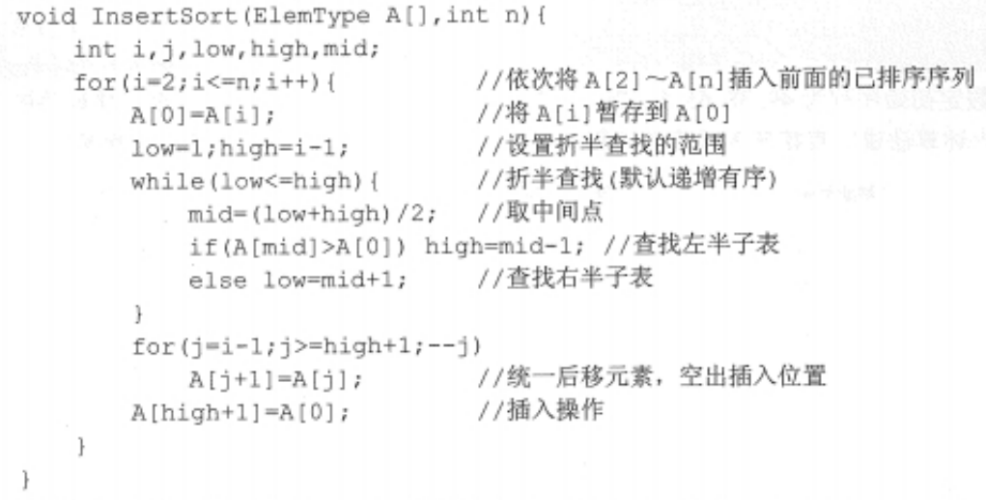
①操作步骤：查找出L(i)在L[1...i-1]中的插入位置k；将L[k...i-1]中所有元素依次后移一个位置；将L(i)复制到K(k)；

②空间复杂度：采用就地排序，仅使用常数个辅助单元，空间复杂度为；

③时间复杂度：插入元素共进行n-1趟，每趟分为比较关键字和移动元素；最好情况下为顺序排列，每插入一个元素只需比较1次（左/右）而不需要移动元素（右）/需要移动3次（左），时间复杂度为；最坏情况下为逆序排列，总比较次数为（右），总的移动次数为（右）；

④稳定性：每次总是从后向前比较再移动，因此相同元素相对位置不会改变，稳定；

⑤适用性：顺序存储、链式存储（从前向后）



1. 折半插入排序

①操作步骤：直接插入排序中，比较和移动是

一起的，而折半插入排序则把比较和移动分离，

先查找元素待插入位置，再统一移动元素；

②空间复杂度：；

③时间复杂度：折半插入排序减少了比较元素

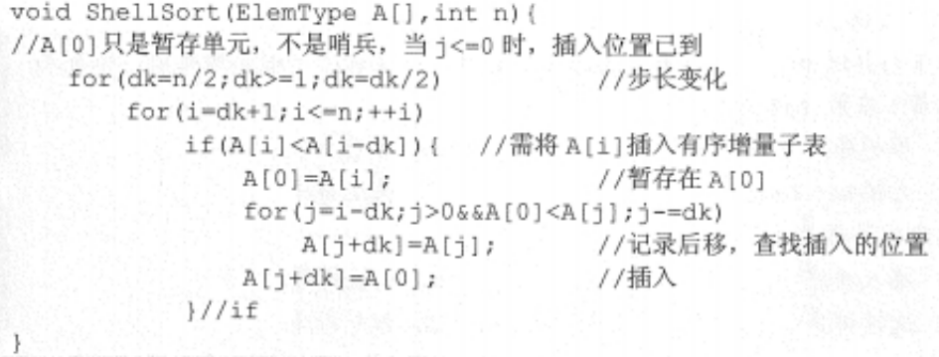
的次数，约为比较次数仅取决于

输入规模n;元素移动次数仍为；因此折

半插入时间复杂度为；

④稳定性：一直到low>high时才停止折半查

找，应将[low,i-1]内元素全部右移，并将A[0]复制到low所指位置；当mid所指元素等于当前元素时，应继续令low=mid+1查找右部元素，以保证稳定性；最终将当前元素插入到low所指的位置（即high+1）；

⑤适用性：只适用于顺序存储，不适用于链表；

1. 希尔排序（缩小增量排序）

①操作步骤：直接插入排序算法适合基本有序的排

序表和数据量不大的排序表；先将待排序表分割成

L[i,i+d,i+2d,...,i+kd]的特殊子表，即把相隔某个

增量的记录组成一个子表；对各个子表进行直接插

入排序；缩小增量d，重复上述过程直到d=1为

止；程序中对各个子表交替排序；

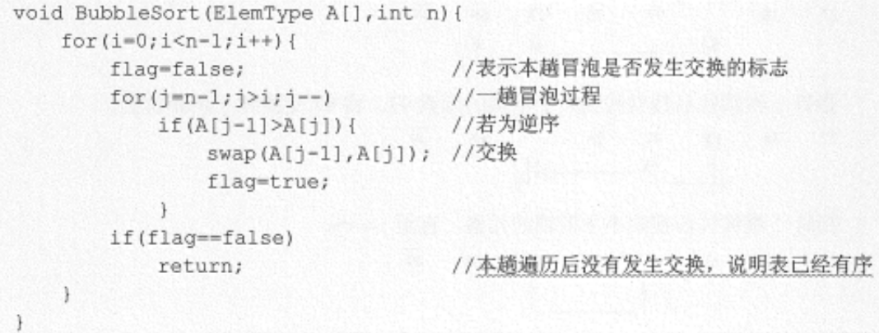
②空间复杂度：；

③时间复杂度：；最坏情况下时间复杂度为；

④稳定性：当相同关键字被划分到不同子表时可能会改变相对次序，因此不稳定；

⑤适用性：只适用于顺序存储，不适用于链表；

1. 交换排序
2. 冒泡排序



①操作步骤：从后向前（或相反）两两比较相邻元素的值，若为逆序（即A[i-1]>A[i]），则交换，直到序列比较完成；称为第一趟冒泡，结果是将最小的元素交换到第一个位置；没有交换发生则说明有序；

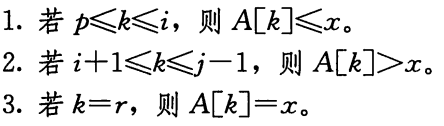
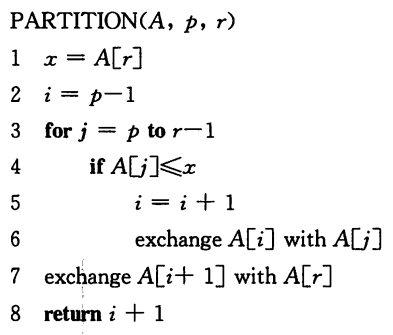
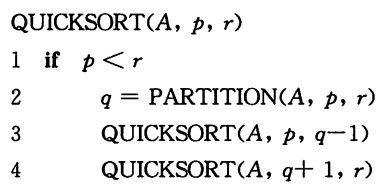
②空间复杂度：；

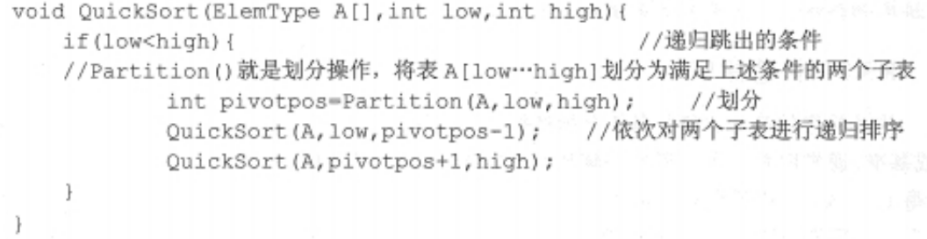
③时间复杂度：当初始序列有序时，比较次序为n-1，移动次数为0，最好情况时间复杂度为；最坏情况下为逆序排列，需要进行n-1趟排序，第i趟排序要进行n-i次关键字比较和交换，每次交换移动3次元素，最坏情况下时间复杂度为；

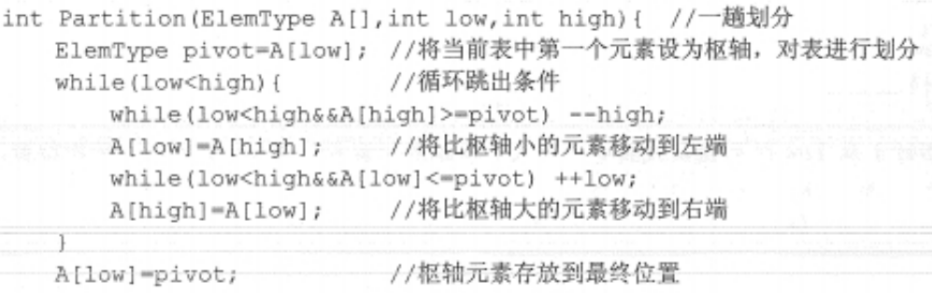
④稳定性：当i>j且A[i]=A[j]时不会交换，因此稳定；

⑤适用性：适用于链表；

1. 快速排序





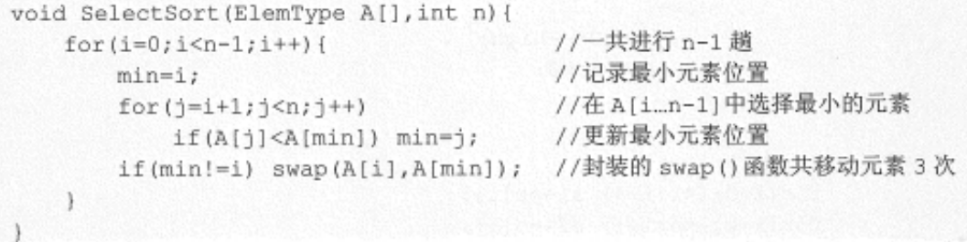


①操作步骤：在待排序表L[1...n]中任取一个元素pivot作为枢轴，通过一趟排序将待排序表划分为独立的两部分L[1...k-1]和L[k+1...n]，使得L[1...k-1]中的所有元素小于pivot，L[k+1...n]中的所有元素大于等于pivot，则pivot放在最终位置L(k)上，这个过程称为一趟快速排序；一趟排序和一次划分不等价，一趟排序可能确定多个元素的位置，即进行多次划分；

②空间复杂度：快速排序是递归的，需要借助一个递归工作栈来保存每层递归调用的必要信息，最好情况下为，最坏情况下要进行n-1次递归调用，栈深为；平均情况下，栈的深度为；

③时间复杂度：快速排序的最坏情况发生在两个区域分别包含n-1个元素和0个元素时，对应于初始排序表基本有序或基本逆序时，最坏情况的时间复杂度为；最理想情况发生在平衡地划分，时间复杂度为；快速排序是所有内部排序算法中平均性能最优的排序算法；

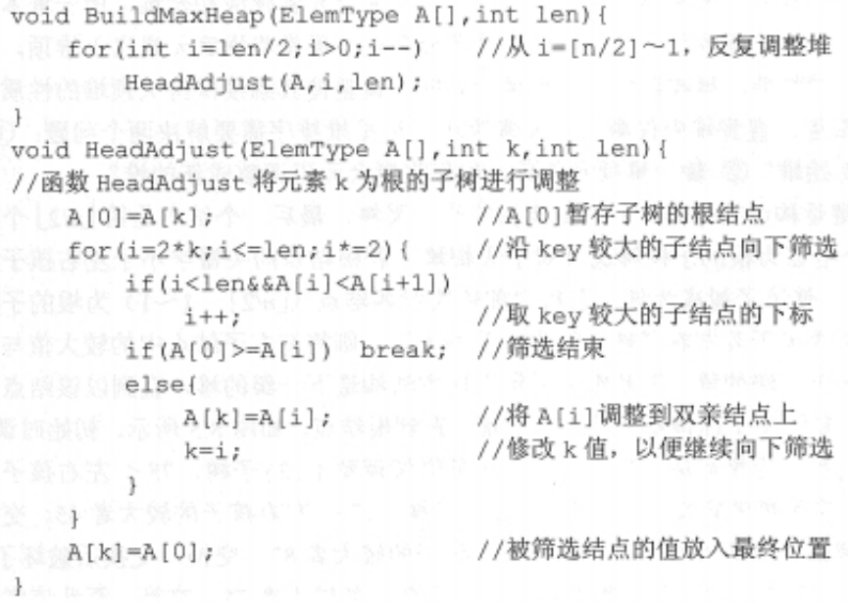
④稳定性：若右端区间有两个关键字相同，且均小于基准值，交换到左区间后相对位置会发生变化，因此不稳定；

1. 选择排序
2. 简单选择排序

①操作步骤：假设排序表为L[1...n]，第i趟排序即从L[i...n]中选择关键字最小的元素与L(i)交换，每趟排序可以确定一个元素的最终位置；

②空间复杂度：；

③时间复杂度：元素移动次数少，最多为3(n-1)次，最好为0次；元素比较次数与初始状态无关，始终是n(n-1)/2次，因此时间复杂度是；

④稳定性：在第i趟找到最小元素后，和第i个元素交换，可能导致第i个元素与其含有相同关键字元素的相对位置发生改变，因此不稳定。

⑤适用性：适用于链表；

1. 堆排序

◎定义：且

称为大根堆；且

称为小根堆

；大根堆（大顶堆）最大元素放在根节点；

①操作步骤：首先将存放在L[1...n]中的n个元

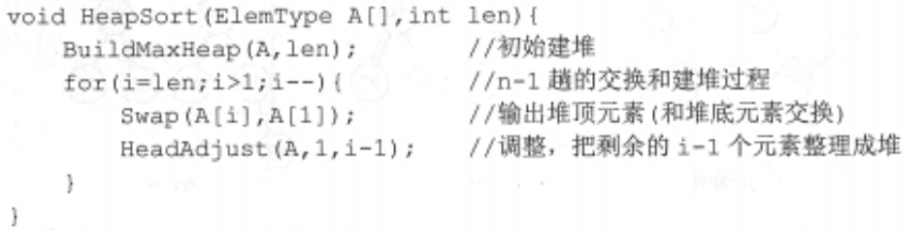
素建成初始堆，堆顶元素就是最大值；输出堆顶

元素后，将堆底元素送往堆顶，此时根节点不满

足性质，将堆顶元素向下调整保持性质，再次输

出堆顶元素；如此反复直到堆中仅剩一个元素为

止；从后向前检查所有非终端结点

是否满足根>左、右，若不满足，将当前结点与更

大的一个孩子结点更换；若元素互换破坏了下一级

的堆，则采用相同方法继续向下调整；

②空间复杂度：；

③时间复杂度：建堆时间为；一个结点下坠

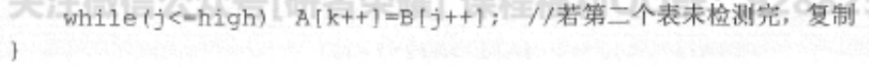
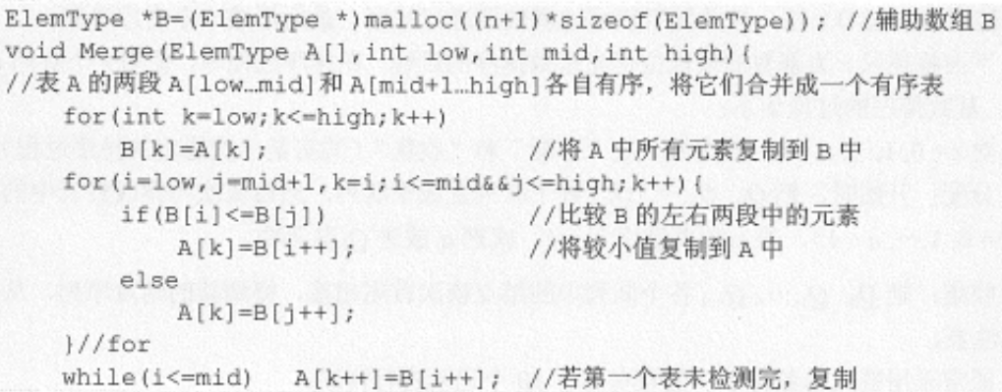
一层对比2次，树高h结点在第i层，需下坠h-i

次，即对比2(h-i)次；有n个结点的完全二叉树，树高；关键字对比次数不超过，建堆时间复杂度为；每趟最多下坠h-1层，因此每趟不超过，共进行h-1趟，因此时间复杂度为；

④稳定性：不稳定；

⑤在堆中插入删除元素：如果有两个孩子结点，则下坠1层需要对比2次，如果只有1个孩子，则需要对比1次；插入新元素放在堆底，将插入元素与父结点对比，（小根堆）如果小于父结点则交换，直到无法继续上升为止；删除元素后，用堆底元素代替被删除元素，然后让该元素与（小根堆）较小的孩子比较，不断下坠；

1. 归并排序和基数排序



1. 归并排序

①操作步骤：假定待排序表含有n个记

录，可将其视为n个有序子表，每个字

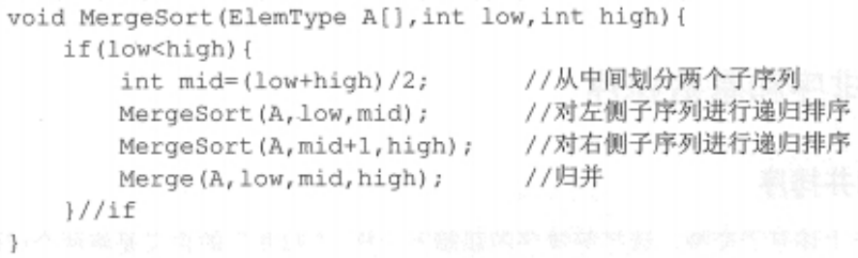
表长度为1；两两归并，得到个长度为

2或1的有序表；如此重复，直到合并

为一个长度为n的有序表为止；m路

归并，每选出一个元素需要对比关键字

m-1次；

②空间复杂度：辅助空间为n个单元，

空间复杂度为；

③时间复杂度：每趟归并的时间复杂

度为；共需进行趟归并

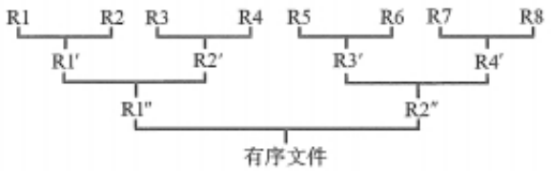
，时间复杂度为；

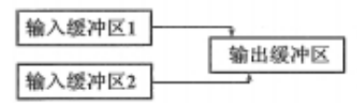
④稳定性：2路归并是稳定的；

1. 基数排序

①操作步骤：假设长度为n的线性表中每个结点aj的关键字由d元组（）组成。其中，r称为基数，为最主位关键字，为最次位关键字；初始化设置r个空队列；按照各个关键字位权重递增次序（个十百），对d个关键字作分配和收集；分配：顺序扫描各个元素，若当前处理关键字位=x，就将元素插入Qx队尾；把各个队列中的结点依次出队并链接；第一趟收集结束，按个位递减排列；第二趟收集结束，按十位递减排列，十位相同按个位递减排列；

②空间复杂度：一趟排序需要的辅助存储空间位r，空间复杂度为；

③时间复杂度：基数排序需要进行d趟分配和收集，一趟分配需要，一趟收集需要，所以基数排序时间复杂度为，它与序列的初始状态无关；

④稳定性：按位排序时必须稳定，所以基数排序稳定；

1. 外部排序
2. k路归并

①外部排序原理：归并排序最少只需要3个内存块即可对任意大的文件排序；首先构造初始归并段，因为归并排序要求各个子序列有序，因此每次读两块内容，内部排序后写回磁盘；第一趟归并，将两个有序归并段归并为一个，缓冲区2空了就要用归并段2的下一块补上......；

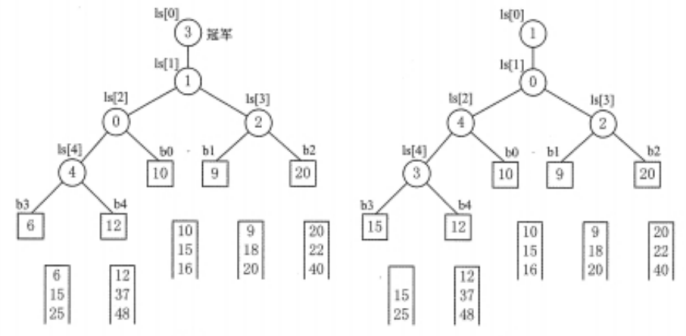
②时间复杂度：外部排序时间开销=读写外存时间+内部排序时间+内部归并时间；R1...R8为8次内部排序得到的8个初始归并段，初始归并和每一趟归并读写磁盘各16次，二路归并读写磁盘次数为32+32\*3=128次；

③多路归并优化：采用多路归并可以减少归并趟数，从而减少I/O次数，对r个初始归并段，做k路归并，归并树可用k叉树表示，若树高为h，则归并趟数=h-1=；故k越大,r越小归并趟数越少；缺点：k路归并需要开辟k个缓冲区内存开销增大，每挑选一个关键字需要对比k-1次内部排序开销增大； 减少初始归并段的数量，如果能增加初始归并段的长度，就可以减少归并段的数量；

④k路平衡归并：最多只能有k个段归并为1个；每一趟归并中，若有m个段参与归并，则一趟处理后得到个归并段；

1. 多路平衡归并与败者树

①败者树解决问题：多路平衡归并排序中，从k个归并段选出一个最小/大元素需要对比关键字k-1次，构造败者树可以使关键字对比次数减少到；

②败者树：败者树是一颗完全二叉树多一个结点；k个叶子结点对应k个归并段参加比较的元素，非页子结点用来记录左右子树中的失败者，胜者继续比较直到根节点之上的结点；对于k路归并，第一次构造败者树需要对比关键字k-1次；有败者树后，选出最小元素只需要对比关键字次；

③败者树的构建：如图

1. 置换-选择排序

◎置换-选择排序解决问题：让初始归并段长度增加，减少初始归并段数；

假设初始待排序文件为FI，初始归并段文件为输出文件FO，内存工

作区为WA，FO与WA的初始状态为空，并假设内存工作去WA的

容量可容纳w个记录，则置换-选择排序的操作的过程为：

①从FI输入w个记录到工作区WA。

②从WA中选出其中关键字最小的记录，记为MINIMAX记录。

③将MINIMAX记录输出到FO中去。

④若FI不为空，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。

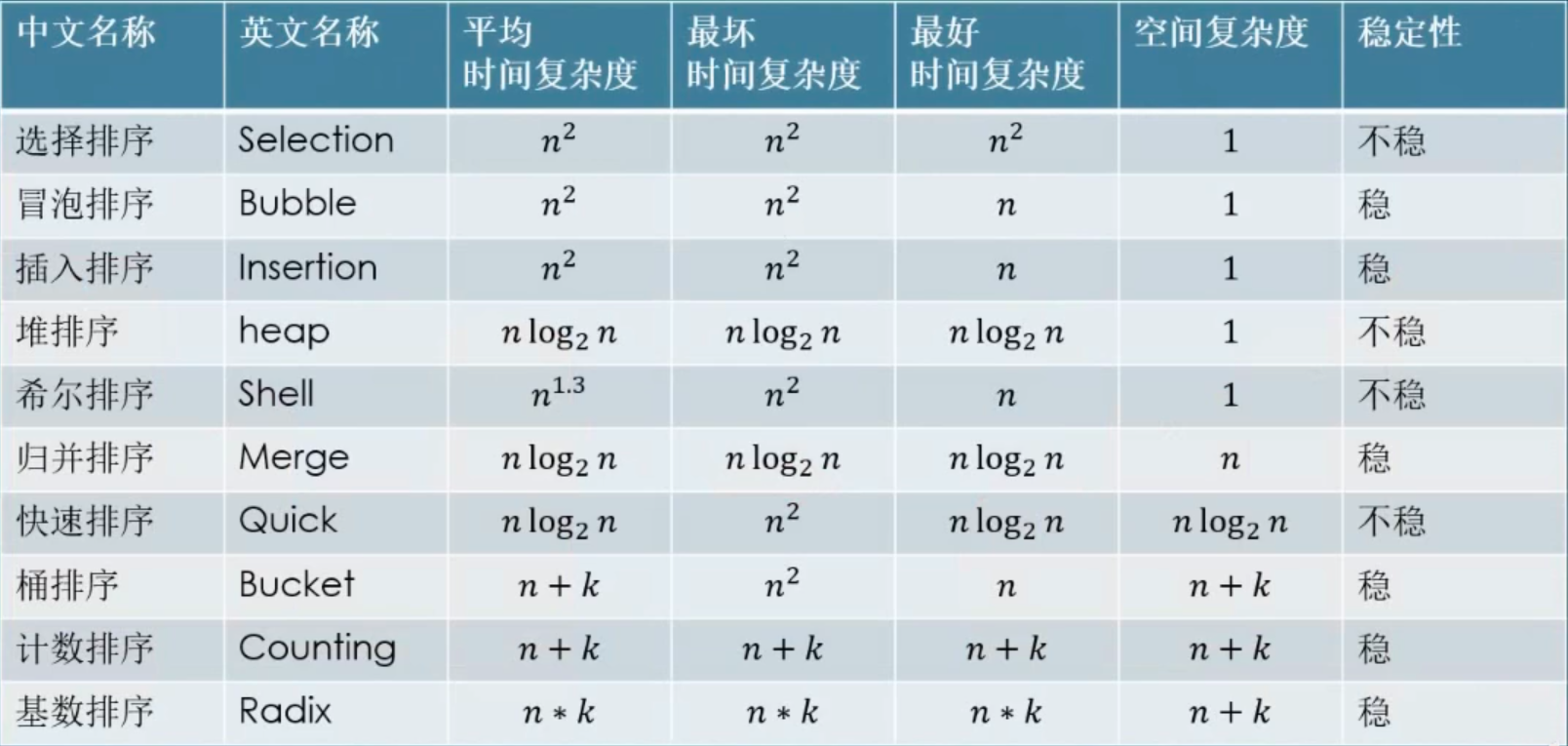
⑥重复③～⑤，直至WA中选不出新的MINIMAX记录为止，由此得到一个初始归并段，输出一个归并段的结束标记到FO中去。

⑦重复②～⑥，直至WA为空。由此得到全部归并段。

1. 最佳归并树

①最佳归并树解决问题：经过置换选择排序后得到长度不等的初始归并段，最佳归并树使I/O次数最少；每个初始归并段视作叶子结点，归并段长度作为结点权值，则磁盘I/O次数=归并树的带权路径长度WPL\*2；可以运用哈夫曼树思想，让记录数最少的初始归并段最先归并，就可以建立总的I/O次数最少的最佳归并树；

②非严格k叉树：哈夫曼树中的最佳归并树应该是严格k叉树，即树中只有度为3或0的结点；若初始归并段不足以构成严格k叉树时，需添加长度为0的虚段；

③如何判断添加虚段的数目：设度为0的结点有个，度为k的结点有个，则对严格k叉树有，由此可得；若，则说明这个叶结点正好构造k叉归并树；若，则说明对于这个叶结点，其中有u个多余，应当再加上k-u-1个空归并段；