第五章 排序

part 3: 分配排序 与 归并排序

张史梁
slzhang.jdl@pku.edu.cn

内容提要

- 排序的基本概念
- 插入排序
- 选择排序
- 交换排序
- 分配排序
- 归并排序

分配排序

- 例子：扑克牌非递减排序
  - 花色：梅花<方块<红心<黑桃
  - 面值：2<3<...<10<J<Q<K<A；
  - 规定：花色的地位高于面值
  - 排序后为：梅花2，...，梅花A，方块2，...，方块A，红心2，...，红心A，黑桃2，...，黑桃A 。

- 扑克牌排序有以下两种方法：
  ① 先将牌按花色分成4堆，然后将每堆按面值从小到大排序，最后按花色从小到大迭在一起。（每一堆分别进行排序）
  ② 先将牌按面值大小分成13堆，然后从小到大把它们收集起来；再按花色分成4堆，最后顺序地收集起来。（分配一收集一分配）

分配排序

- 多关键码排序条件分析
  - 假设有n个记录($R_0, R_1, \ldots, R_{n-1}$)
  - 每个记录R_i中含有d个关键码($k_i^0, k_i^1, \ldots, k_i^{d-1}$)
  - 则该n个记录对关键码($k^0, k^1, \ldots, k^{d-1}$)有序是指：

    文件中任意两个记录R_i和R_j($0 \le i \le j \le n-1$)满足有序关系

    $(k_i^0, k_i^1, \ldots, k_i^{d-1}) < (k_j^0, k_j^1, \ldots, k_j^{d-1})$
  - k^0称为最高位关键码，k^{d-1}称为最低位关键码。

分配排序

- 实现多关键码排序的方法：
  - 高位优先法
    - 先对 K^0 排序分堆，再就每堆对K^1排序分堆，直到K^{d-1}
  - 低位优先法
    - 先对K^{d-1}排序，再对K^{d-2}排序，直到K^0

分配排序的基本思想：
把排序码分解成若干部分，然后通过对各个部分排序码的分别排序，最终达到整个排序码的排序。

基数排序

- 把每个排序码看成是一个d元组：

  $K_i=(K_i^0, K_i^1,\ldots, K_i^{d-1})$

  - 其中每个K_i都是集合$\{C_0，C_1，\ldots，C_{r-1}\}$中的值
  - 即$C_0\le K_i^j\le C_{r-1}(0\le i\le n-1,0\le j\le d-1)$， 其中*r*称为基数。
- 排序时先按K_i^{d-1}从小到大将记录分配到*r*个堆中；然后依次收集，再按K_i^{d-2}分配到*r*个堆中...
- 如此反复，直到对K_i^0分配、收集，便得到最终排序序列

- 例如，数值675：$K_i=(6, 7, 5)$，$\{0,1,2, \ldots , 9\}$，基数为10

重要：理解什么是基数

基数排序 - 例子

- 数列{ 36, 5, 16, 98, 95, 47, 32, 36’, 48, 10 }，请用基数排序法排序。

(1)初始状态

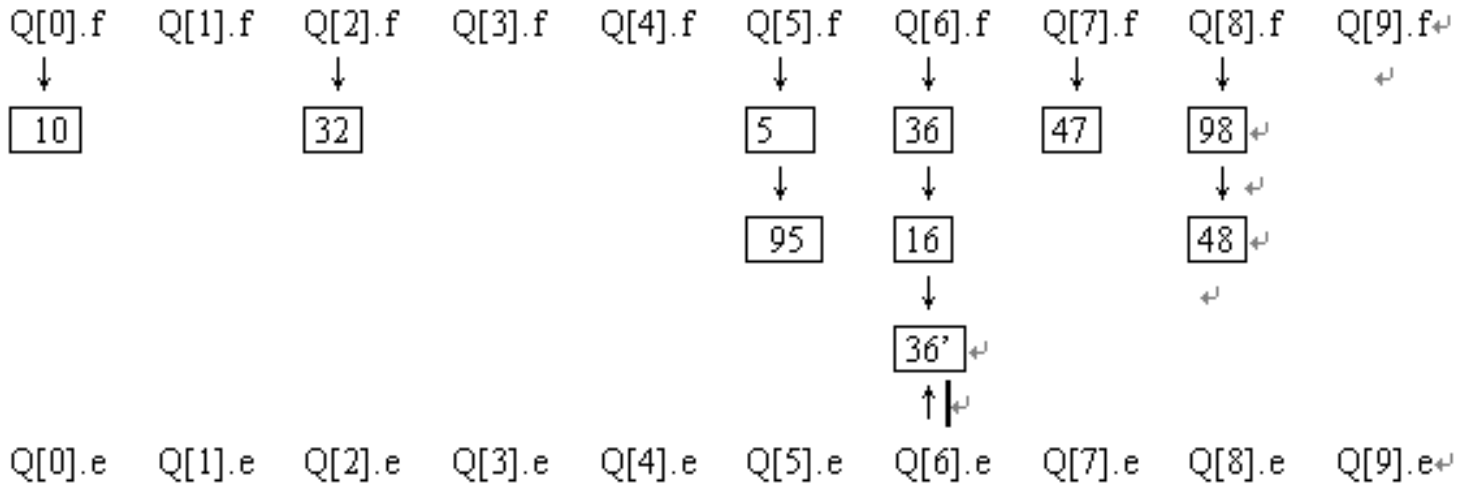
36 →5 →16 →98 →95 →47 →32 →36’ →48 →10

例子(续)

(1) 初始状态

36 →5 →16 →98 →95 →47 →32 →36’ →48 →10

(2) 第一趟分配后



(3)第一趟收集后

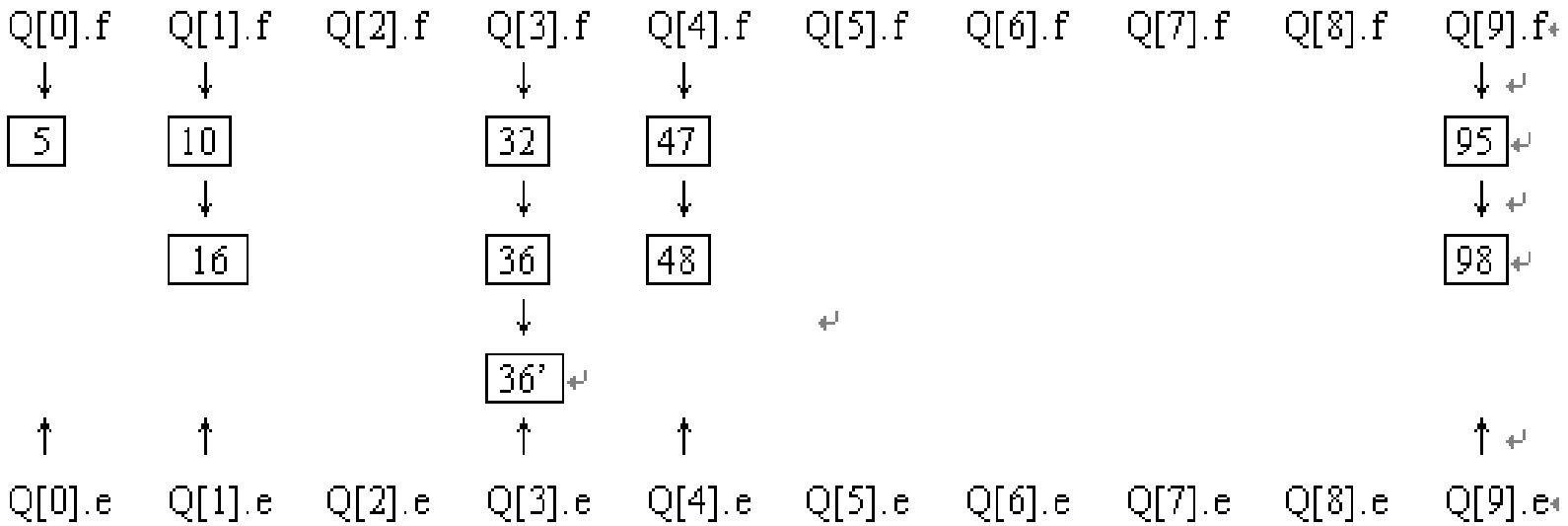
10 →32 →5 →95 →36 →16 →36’ →47 →98 →48

例子(续)

(3)第一趟收集后

10 →32 →5 →95 →36 →16 →36’ →47 →98 →48

(4) 第二趟分配后



(5)第二趟收集后

5 →10 →16 →32 →36 →36’ →47’ →48→95→98

归并排序-记录的数据结构

```
①  # define D 3     /* D为排序码的最大位数*/
②  # define R 10    /* R为基数*/
③  struct Node
④  {
⑤     KeyType key;       /* 排序码 */
⑥     DataType info;
⑦     Struct Node *next;
⑧   };
⑨  typedef struct Node * RadixList;
```

```
⑩  typedef struct
⑪  {
⑫       RadixList *f;
⑬       RadixList *e;
⑭   }Quene;
⑮  Queue queue[R];
```

使用单向链表存储待排序文件
使用R个队列来实现分配-收集，R是什么？

基数排序-实现

```
① void radixSort( RadixList *plist,int d, int r)   // d为排序码个数，r为基数
② {                                                 // plist为存储排序文件的单链表
③    int i,j,k;  RadixNode *p,*head;
④    head=(*plist)->next;
⑤    for(j=d-1;j>=0;j--)    /*进行d次分配和收集*/
⑥    {
⑦       p=head;
⑧       for (i=0;i<r;i++)   //首先清空队列
⑨       {
⑩          queue[i].f=NULL;      queue[i].e=NULL;
⑪       }
⑫       while(p!=NULL)     //从头向后扫描待排序链表，并分配
⑬       {
⑭         k=p->key[j];     //取第j个排序码
⑮         if(queue[k].f==NULL)    queue[k].f=p;     /*为队头*/
⑯         else    (queue[k].e)->next=p;  /*到第k队的队尾*/
⑰         queue[k].e=p;                          //p所指文件进入队列queue[k]
⑱          p=p->next;
⑲       }
```

基数排序-实现

```
   //分配完毕，开始收集
①        i=0;
②        while(queue[i].f==NULL) i++;  /* 第一个非空的队列*/
③        p=queue[i].e;
④        head=queue[i].f;    /* head为收集链表的头指针*/

⑤        for(i=0; i<r;i++)
⑥          if(queue[i].f!=NULL)
⑦           {
⑧             p->next=queue[i].f;
⑨             p=queue[i].e;
⑩           }                        /*收集非空队列*/

⑪         p->next=NULL;
⑫     }   //完成一次分配、收集

⑬     (*plist)->next=head;
⑭  }
```

基数排序算法性能分析

- 排序中没有记录的移动，只是对链表的扫描和指针的赋值，所以，时间耗费主要在修改指针上；
- 每趟排序中，清队列的时间为O(r)，将n个记录分配到队列的时间为O(n)，收集的时间为O(r)，因此，一趟排序的时间为O(r+n)；
- 共要进行d趟排序，因此，基数排序的时间复杂度T(n)=O(d*(r+n))
- 当n较大、d较小(排序码位数)，特别是记录的信息量较大时，基数排序非常有效。

基数排序算法性能分析

- 基数排序中，每个记录中增加了一个next字段，还增加了一个queue数组，故辅助空间为S(n)=O(n+r)
- 基数排序是稳定的。

内容提要

- 排序的基本概念
- 插入排序
- 选择排序
- 交换排序
- 分配排序
- 归并排序

归并排序：划分-归并

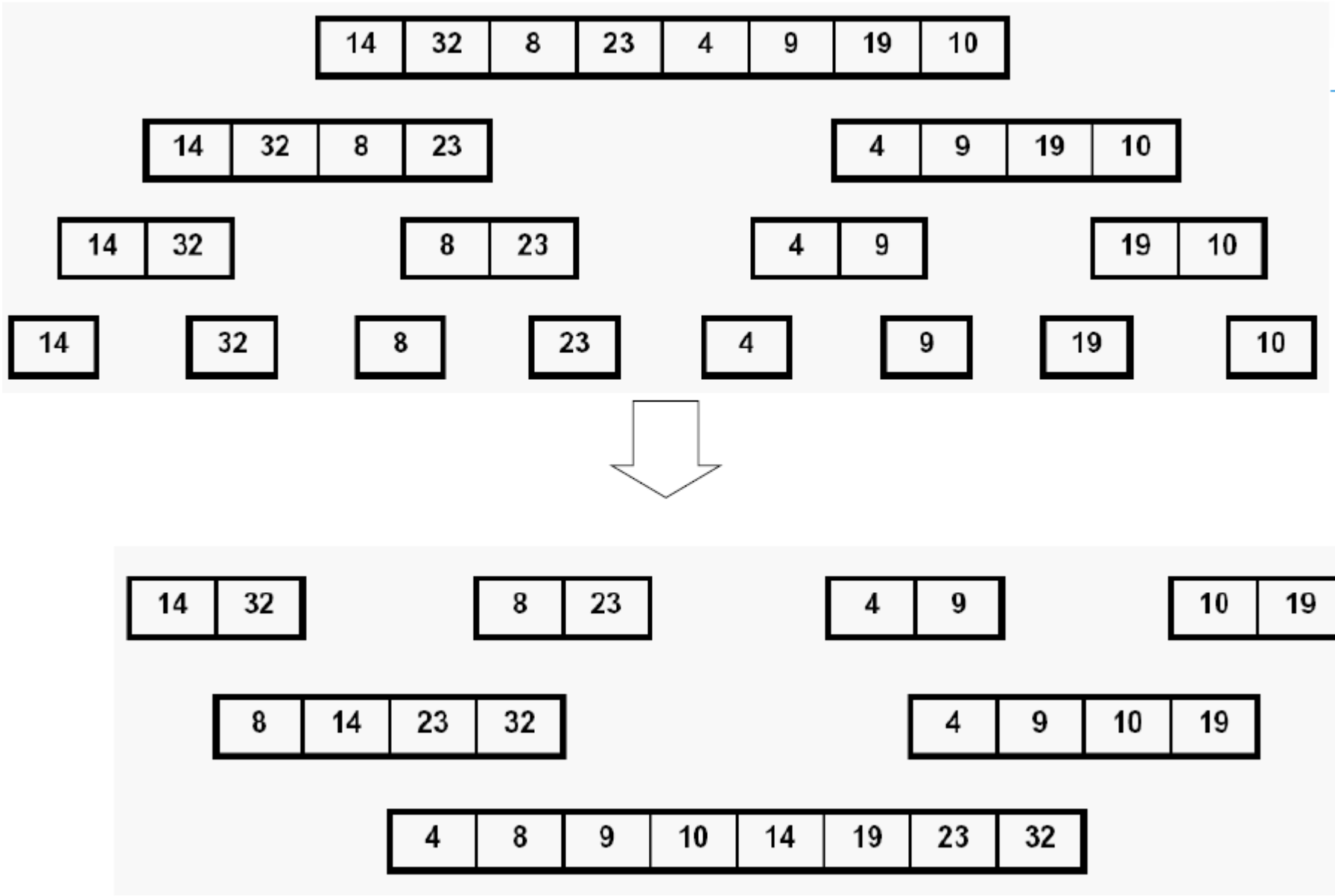
- 把待排序的文件分成若干个子文件，先将每个子文件内的记录排序；
- 再将已排序的子文件合并，得到完全排序的文件：
  - 合并时比较各子文件第一个记录的排序码，排序码最小的记录为排序后的第一个记录，取出该记录；
  - 继续比较各子文件的第一个记录，找出排序后的第二个记录；
  - 如此反复，经过一次扫描，得到排序结果。

分而治之：分治法

归并排序(续)

- 归并排序的一般处理：
  - 设文件中有n个记录，可以看成n个子文件，每个子文件中只包含一个记录;
  - 先将每两个子文件归并，得到n/2个部分排序的较大的子文件，每个子文件包含2个记录；
  - 再将子文件归并，如此反复，直到得到一个文件。

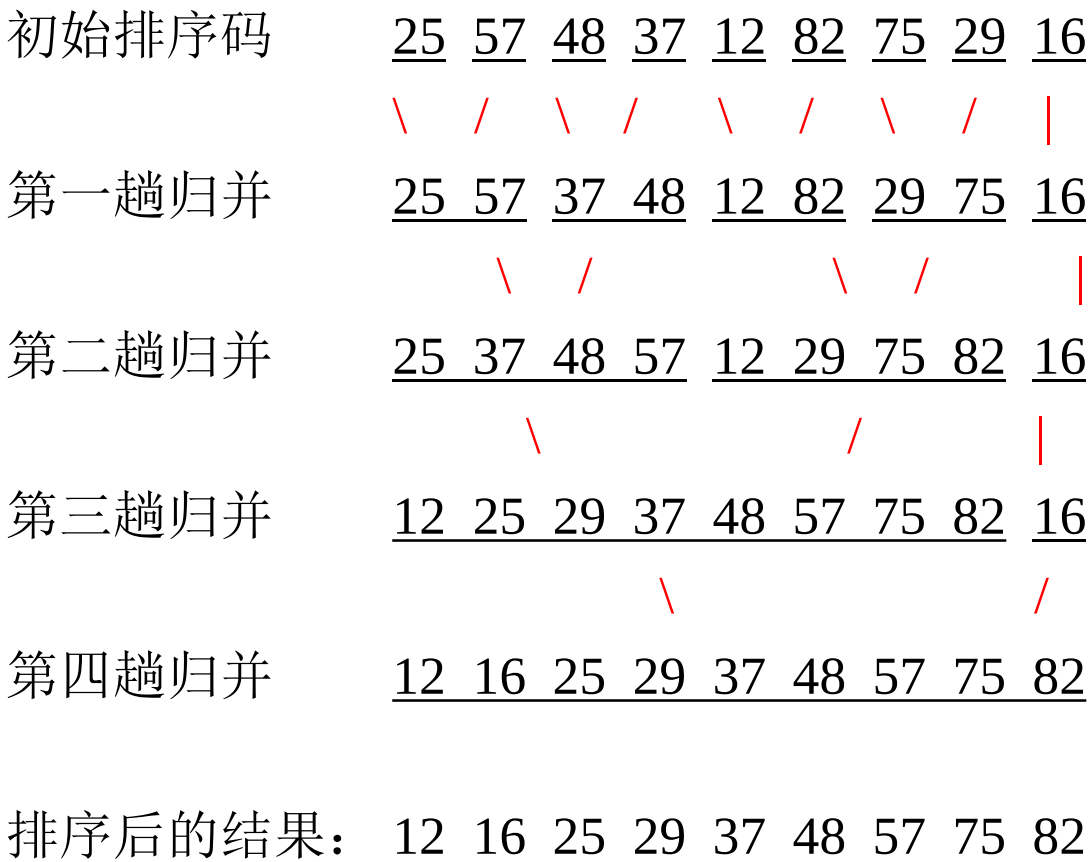
- 上述每步归并都是将两个子文件合成一个子文件，称为“二路归并排序”。类似地，还有“三路归并排序”或“多路归并排序”



bottom-up, 自底向上

二路归并排序-例子

初始序列{25, 57, 48, 37, 12, 82, 75, 29, 16}，用二路归并排序法排序



排序后的结果：12 16 25 29 37 48 57 75 82

两路归并算法

```
①  void merge(RecordNode r[ ],RecordNode r1[ ],int low,int m,int high)
②  {                  /*r[low]到r[m]和r[m+1]到r[high]是两个有序文件*/
③     int i,j,k;
④     i=low;j=m+1;k=low;
⑤     while((i<=m)&&(j<=high))
⑥     {              /*从两个有序文件中的第一个记录中选出小的记录*/
⑦        if(r[i].key<=r[j].key)        r1[k++]=r[i++];
⑧        else    r1[k++]=r[j++];
⑨     }
⑩     while(i<=m) r1[k++]=r[i++];       /*复制第一个文件的剩余记录*/
⑪     while(j<=high) r1[k++]=r[j++];    /*复制第二个文件的剩余记录*/
⑫  }
```

一趟归并算法

```
①  void mergePass(RecordNode r[] ,RecordNode r1[ ],int n, int length)
②  {
③     /*对r做一趟归并,结果放在r1中, length 为本趟归并的有序子文件的长度*/
④     int i,j;
⑤     i=0;
⑥     while(i+2*length-1<n)     /* 归并长度为length的两个子文件*/
⑦     {
⑧        merge(r,r1,i,i+length-1,i+2*length-1);
⑨        i+=2*length;
⑩     }
⑪     if(i+length-1<n-1)   /*剩下两个子文件,其中一个长度小于length */
⑫        merge(r,r1,i,i+length-1,n-1)
⑬     else
⑭        for(j=i;j<n;j++)     / *将最后一个子文件复制到数组r1中*/
⑮           r1[j]=r[j];
⑯  }
```

两路归并算法

```
① void mergeSort(SortObject  *pvector)   //
② {
③       RecordNode record[MAXNUM];
④       int length;
⑤       length=1;
⑥       while(length<pvector->n)
⑦       {
⑧          mergePass(pvector->record,record,pvector->n,length);
⑨                  /*一趟归并,结果存放在数组r1中*/
⑩          length*=2;
⑪          mergePass(record,pvector->record,pvector->n,length);
⑫                  /*一趟归并,结果存放在数组r中*/
⑬          length*=2;
⑭       }
⑮ }
```

算法评价

- 二路归并排序算法的时间复杂度$T(n)=O(n\log_2 n)$
  - 第i次归并以后，有序记录的长度为2^i。因此，具有n个记录的文件排序，必须做$\lceil \log_2 n \rceil$趟归并
  - 每一趟归并所花费的时间是O(n)
- 二路归并排序算法的辅助空间为$S(n)=O(n)$
  - 算法中增加了一个数组**record [ ]**
- 二路归并排序是稳定的

内容提要

- 排序的基本概念
- 插入排序
- 选择排序
- 交换排序
- 分配排序
- 归并排序
- 各种排序算法的比较

排序算法总结

- 五类、10个排序算法
  - 插入排序
  - 选择排序
  - 交换排序
  - 分配排序
  - 归并排序

1 直接插入排序
2 二分法插入排序
3 表插入排序
4 Shell排序
5 直接选择排序
6 堆排序
7 起泡排序
8 快速排序
9 基数排序
10 归并排序

各种排序方法的比较

- 排序算法之间的比较主要考虑以下几个方面：
  - 算法的时间复杂度
  - 算法的辅助空间
  - 排序的稳定性
  - 算法结构的复杂性
  - 参加排序的数据的规模
- 各种排序算法的时间复杂度与辅助空间及算法的稳定性如下表所示

各种排序算法的理论时间代价

算法	最大时间	平均时间	最小时间	辅助空间代价	稳定性
直接插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
二分插入排序	$O(n^2)$	$O(n^2)$	$O(nlogn)$	$O(1)$	稳定
Shell排序	$O(n^{1.3})$	$O(n^{1.3})$	$O(n^{1.3})$	$O(1)$	不稳定
直接选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(1)$	不稳定

各种排序算法的理论时间代价

算法	最大时间	平均时间	最小时间	辅助空间代价	稳定性
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	稳定
改进冒泡排序	$O(n^2)$	$O(n^2)$	O(1)	O(1)	稳定
快速排序	$O(n^2)$	O(nlogn)	O(nlogn)	O(logn)	不稳定
基数排序	O(d*(n+r))	O(d*(n+r))	O(d*(n+r))	O((n+r))	稳定
归并排序	O(nlogn)	O(nlogn)	O(nlogn)	O(n)	稳定

算法评价

- 各种排序方法各有优缺点，可适用于不同的场合，应当根据具体情况，选择合适的排序算法。
- 从数据的规模n来看，
  - 当数据规模n较小时，n^2和$n\log_2 n$的差别不大，则采用简单的排序方法比较合适
    - 如直接插入排序、直接选择排序、起泡排序等
  - 当数据规模n较大时，应选用速度快的排序算法
    - Shell排序$O(n^{1.3})$、堆排序$O(n\log_2 n)$、快速排序$O(n\log_2 n)$及归并排序$O(n\log_2 n)$的排序速度较快

算法评价(续)

- 从算法结构的简单性看，
  - 速度慢的排序算法比较简单、直接
  - Shell排序（插入排序）、快速排序（交换排序）、堆排序法（选择排序）、归并排序法可以看作是对某一种排序方法的改进，算法结构一般都比较复杂

算法评价(续)

- 从文件的初态来看，
  - 当文件的初态已基本有序时，可选择简单的排序方法，如直接插入排序、起泡排序等
  - 快速排序有可能出现最坏情况，则快速排序算法的时间复杂度为$O(n^2)$，且递归深度为n，即所需栈空间为O(n)；
  - 堆排序不会出现像快速排序那样的最坏情况，且堆排序所需的辅助空间比快速排序少；
    - 但这两种算法都是不稳定的，如果要求排序是稳定的，则可以选择归并排序方法

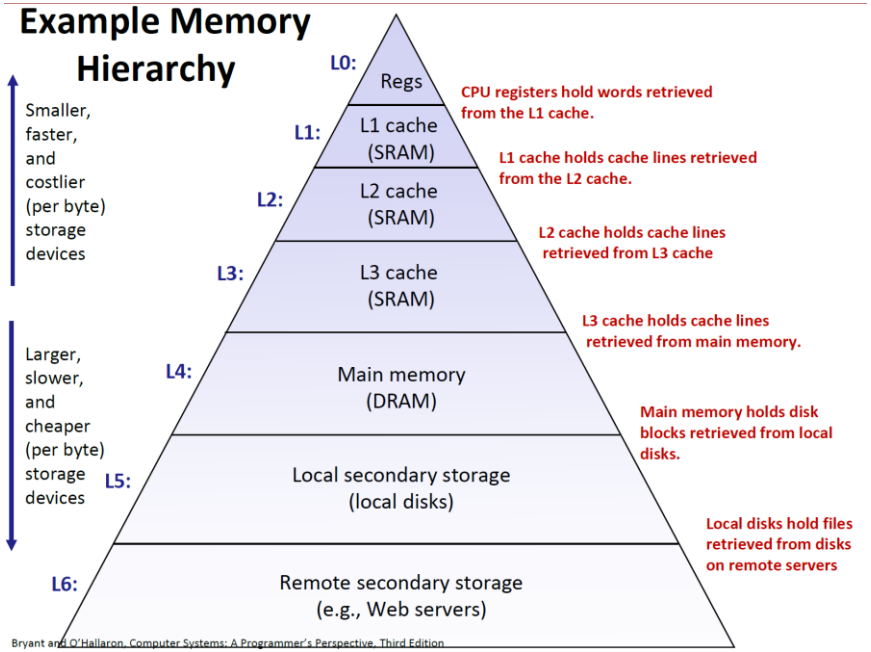
算法评价(续)

- 基数排序法所需的辅助空间较大，
  - 其时间复杂度可简化成O(dn)；当排序码的位数d较少时，可进一步简化成O(n)，能达到较快的速度。
  - 但是基数排序只适用于像字符串和整数这类有明显结构特征的排序码，当排序码的取值范围为某个无穷集合时，则无法使用
  - 因此，当n较大，记录的排序码位数较少且可以分解时，采用基数排序方法较好

- 归并排序法可以用于内排序，也可用于外排序。

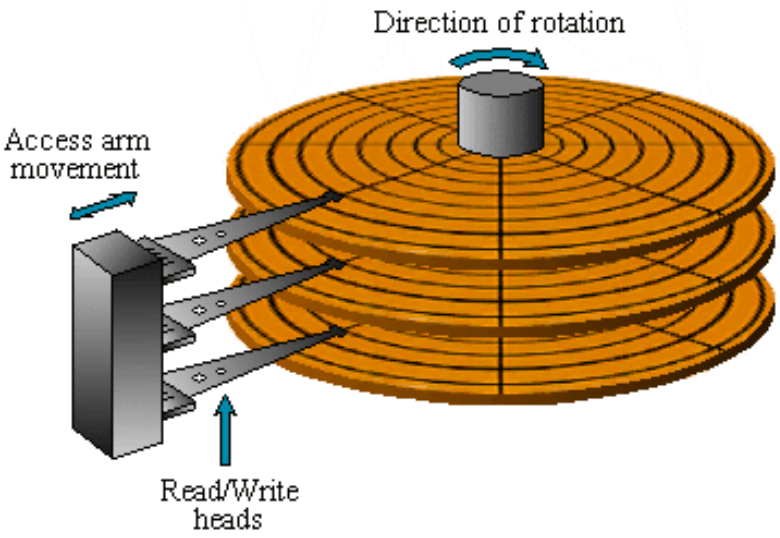
外排序（选学）

- 外排序（external sorting）
  - 外存设备上（文件）的排序技术
  - 由于文件很大，无法把整个文件的所有记录同时调入内存中进行排序，即无法进行内排序
  - 访问外存的速度比访问内存的速度慢一千倍以上



磁盘读取过程


磁盘盘片
传动轴
主轴
下方是轴承和电动机
永磁铁
空气过滤片
磁头停泊区
磁头
知乎 @吴先生


Direction of rotation
Access arm movement
Read/Write heads

外部排序需要解决的主要问题

- 减少读写磁盘的次数
- 批量读写记录
- 关键码排序

减少读写磁盘的次数

- 起泡排序需要扫描n趟记录
- 直接选择排序需要扫描n趟记录
- 归并排序需要扫描$\log_2 n$趟记录

批量读写记录

- 建立缓冲区
  - 每次从外存将记录批量读入输入缓冲区，排序算法逐个使用
  - 待输出的记录先写入输出缓冲区，缓冲区满了后一起输出

关键码排序

- 一般而言，每条记录都很大，而其关键码很小（只占几个字节）
  - 例如，学生管理条目可能存储几百个字节的信息，包括姓名、学号、地址、选课记录、成绩等。排序关键码可能只是只占几个字节的学号
- 方法１：读入所有的记录，在内存排序，再把排好序的这些记录写回磁盘，即每当需要处理的时候读取整个记录。I/O太多！
- 方法２：只读出关键码，以及其相应位置的指针。关键码排好序后，可以对原文件的记录重新排列，或是不排（诸如数据库，利用索引）

小结

- 五类、10个排序算法
  - 插入排序
  - 选择排序
  - 交换排序
  - 分配排序
  - 归并排序

1 直接插入排序
2 二分法插入排序
3 表插入排序
4 Shell排序
5 直接选择排序
6 堆排序
7 起泡排序
8 快速排序
9 基数排序
10 归并排序

总结

排序方法	平均时间复杂度	辅助空间	稳定性
直接插入排序	$O(n^2)$	O(1)	稳定
二分法直接排序	$O(n^2)$	O(1)	稳定
表插入排序	$O(n^2)$	O(n)	稳定
Shell排序	$O(n^{1.3})$	O(1)	不稳定
直接选择排序	$O(n^2)$	O(1)	不稳定
堆排序	$O(n\log_2 n)$	O(1)	不稳定
冒泡排序	$O(n^2)$	O(1)	稳定
快速排序	$O(n\log_2 n)$	$O(\log_2 n)$	不稳定
基数排序	O(d(n+r))	O(r+n)	稳定
归并排序	$O(n\log_2 n)$	O(n)	稳定

课堂练习

- 在以下排序方法中，关键字比较的次数与记录的初始排列次序无关的是（）

  A shell排序

  B 起泡排序

  C 直接插入排序

  D 直接选择排序

  E 二分插入排序

课堂练习

- 在内排序的过程中，通常需要对待排序的关键码集合进行多次扫描。采取不同排序方法会产生不同的排序中间结果。设要将序列{Q，H，C，Y，P，A，M，S，R，D，F，X}中的关键码按字母序的升序重新排列，则
  （A）是冒泡排序的一次扫描结果，
  （B）是初始步长为4的希尔排序的一次扫描结果，
  （C）是二路归并排序的一次扫描结果。
  （D）是以第一个元素为分界元素的快速排序的一次扫描结果
- 供选择的答案A~E：

  1：F，H，C，D，P，A，M，Q，R，S，Y，X

  2：P，A，C，S，Q，D，F，X，R，H，M，Y

  3：A，D，C，R，F，Q，M，S，Y，P，H，X

  4：H，C，Q，P，A，M，S，R，D，F，X，Y

  5：H，Q，C，Y，A，P，M，S，D，R，F，X