



6.1 Introduction 概述



Chapter 6 排序

- The algorithm we describe will all be interchangeable.
- Sorting 排序: A data sequence will be sorted according to the key. 将一组杂乱无章的数据排列成一个按关键字有序的序列。
- 数据表 (datalist): 它是待排序数据对象的有限集合。
- 关键字 (key): 通常数据对象有多个属性域，即多个数据成员组成，其中有一个属性域可用来区分对象，作为排序依据。该域即为关键字。每个数据表用哪个属性域作为关键字，要视具体的应用需要而定。即使是同一个表，在解决不同问题的场合也可能取不同的域做关键字。

- Stability 排序算法的稳定性: 如果在对象序列中有两个对象 $r_{[i]}$ 和 $r_{[j]}$ ，它们的关键字 $k_{[i]} == k_{[j]}$ ，且在排序之前，对象 $r_{[i]}$ 排在 $r_{[j]}$ 前面。如果在排序之后，对象 $r_{[i]}$ 仍在对象 $r_{[j]}$ 的前面，则称这个排序方法是稳定的，否则称这个排序方法是不稳定的。

10, 9, 8, 7, 6, 5, 4
 12, 4, 22, 9, 12*, 33, 2 33, 22, 12, 12*, 9, 4, 2
 5, 8, 10, 6, 7, 9, 4
 33, 22, 12*, 12, 9, 4, 2
 5, 8, 6, 10, 7, 9, 4

- 内排序与外排序: 内排序是指在排序期间数据对象全部存放在内存的排序；外排序是指在排序期间全部对象个数太多，不能同时存放在内存，必须根据排序过程的要求，不断在内、外存之间移动的排序。
- 排序的时间开销: 排序的时间开销是衡量算法好坏的最重要的标志。排序的时间开销可用算法执行中的数据比较次数与数据移动次数来衡量。各节给出算法运行时间代价的大略估算一般都按平均情况进行估算。对于那些受对象关键字序列初始排列及对象个数影响较大的，需要按最好情况和最坏情况进行估算。

- 衡量排序方法的标准
 - 排序时所需要的平均比较次数
 - 排序时所需要的平均移动
 - 排序时所需要的平均辅助存储空间
 - 排序的稳定性

6.2 插入排序 (Insertion Sort)

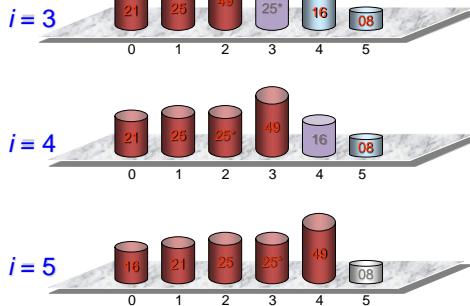
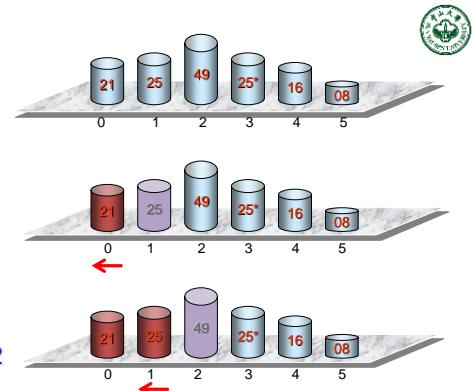
插入排序的基本方法是：每一步将一个待排序的对象，按其关键字大小，插入到前面已经排好序的一组对象的适当位置上（完成一趟排序），直到对象全部插入为止。



直接插入排序 (Insertion Sort)

- 直接插入排序的基本思想是：当插入第 i ($i \geq 1$) 个对象时，前面的 $v[0], v[1], \dots, v[i-1]$ 已经排好序。这时，用 $v[i]$ 的关键字与 $v[i-1], v[i-2], \dots$ 的关键字顺序进行比较，找到插入位置即将 $v[i]$ 插入，原来位置上之后的所有对象依次向后顺移。

各趟
排序
结果



完成

直接插入排序的算法

```
void InsertSort(Sqlist &L)
{ for (int i=2;i<=L.length;++i)
    if (LT(L.r[i].key,L.r[i-1].key))
        { L.r[0]=L.r[i]; // L.r[0]为监视哨
          for (int j=i-1; LT(L.r[0].key,L.r[j].key); --j)
              L.r[j+1]=L.r[j];
              L.r[j+1]=L.r[0];
        }
}
```



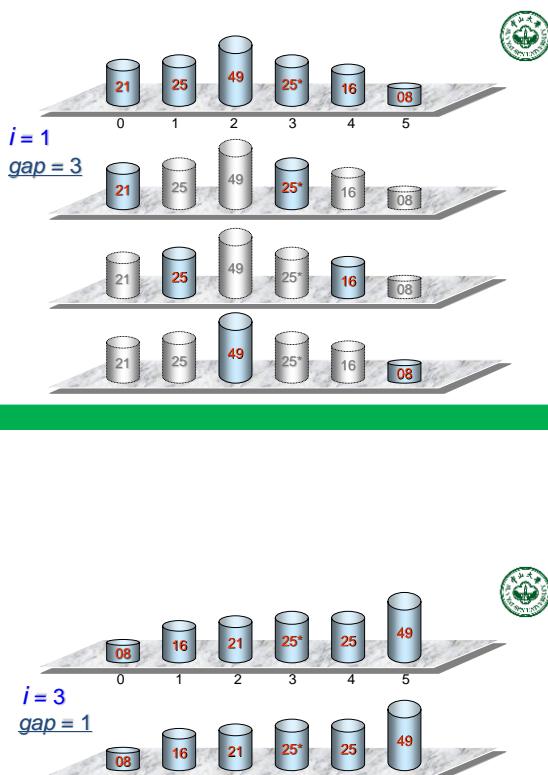
算法分析

- 若设待排序的对象个数为 $L.length=n$ ，则该算法的主程序执行 $n-1$ 趟。
- 关键字比较次数和对象移动次数与对象关键字的初始排列有关。
- 最好情况下，排序前对象已经按关键字大小从小到大有序，每趟只需与前面的有序对象序列的最后一个对象的关键字比较 1 次，移动 2 次对象，总的关键字比较次数为 $n-1$ ，对象移动次数为 $2(n-1)$ 。





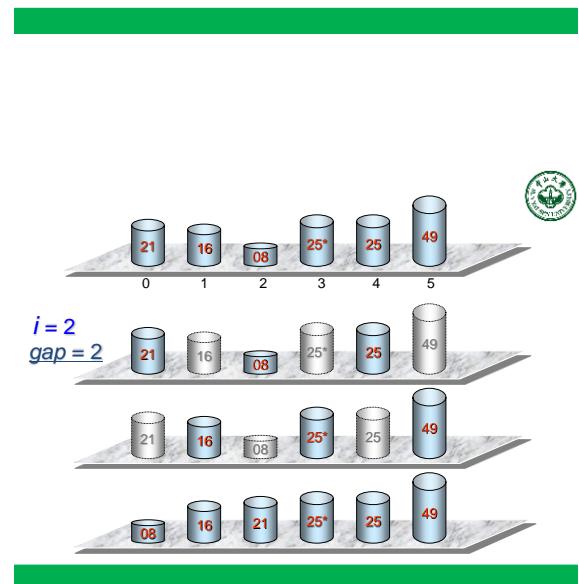
- 若待排序对象序列中出现各种可能排列的概率相同，则可取上述最好情况和最坏情况的平均情况。在平均情况下的关键字比较次数和对象移动次数约为 $n^2/4$ 。因此，直接插入排序的时间复杂度为 $O(n^2)$ 。
- 直接插入排序是一种稳定的排序方法。



- 开始时 gap （间隔值）的值较大，子序列中的对象较少，排序速度较快；随着排序进展， gap 值逐渐变小，子序列中对象个数逐渐变多，由于前面工作的基础，大多数对象已基本有序，所以排序速度仍然很快。

6.3 希尔排序 (Shell Sort)

- 希尔排序方法又称为“缩小增量”排序。
- 该方法的基本思想是：先将整个待排对象序列按照一定间隔分割成为若干子序列，分别进行直接插入排序，然后缩小间隔，对整个对象序列重复以上的划分子序列和分别排序工作，直到最后间隔为1，此时整个对象序列已“基本有序”，进行最后一次直接插入排序。



希尔排序的算法

```
void ShellSort(SqList &L, int dlt[], int t){
    for (int k=0;k<t;++k){
        ShellInsert(L,dlt[k]);
    }
}
```

说明： $dlt[3]=\{5, 3, 1\}$

//一趟希尔排序,按间隔 dk 划分子序列

```
void ShellInsert(SqList &L, int gap){
    for (int i=gap+1;i<=L.length;++i){
        if (LT(L.r[i].key,L.r[i-gap].key)){
            L.r[0]=L.r[i];
            for (int j=i-gap;j>0 &&
                LT(L.r[0].key,L.r[j].key); j-=gap)
                L.r[j+gap]=L.r[j];
            L.r[j+gap]=L.r[0];
        }
    }
}
```



算法分析

- 对特定的待排序对象序列，可以准确地估算关键字的比较次数和对象移动次数。
- 但想要弄清关键字比较次数和对象移动次数与增量选择之间的依赖关系，并给出完整的数学分析，还没有人能够做到。
- gap 的取法有多种。最初shell提出取 $gap = \lfloor n/2 \rfloor$, $gap = \lfloor gap/2 \rfloor$ ，直到 $gap = 1$ 。后来Knuth提出取 $gap = \lfloor gap/3 \rfloor + 1$ 。还有人提出都取奇数为好，也有人提出各 gap 互质为好。



- Knuth利用大量的实验统计资料得出，当 n 很大时，关键字平均比较次数和对象平均移动次数大约在 $n^{1.25}$ 到 $1.6n^{1.25}$ 的范围内。这是在利用直接插入排序作为子序列排序方法的情况下得到的。

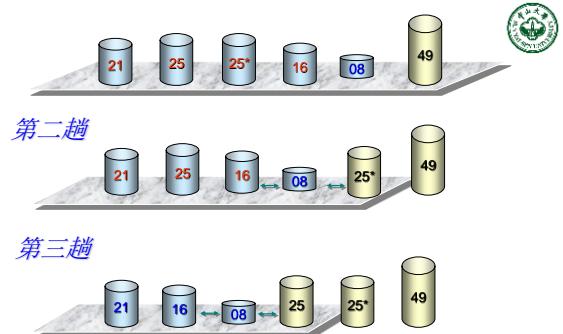
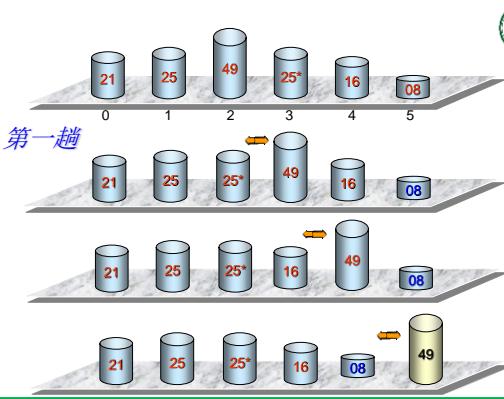


6.4 交换排序 (Exchange Sort)

交换排序的基本思想是两两比较待排序对象的关键字，如果发生逆序(即排列顺序与排序后的次序正好相反)，则交换之，直到所有对象都排好序为止。

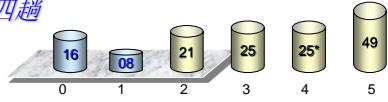
起泡排序 (Bubble Sort)

- 起泡排序的基本方法是：设待排序对象序列中的对象个数为 n ，最多作 $n-1$ 趟排序。在第 i 趟中顺次两两比较 $r[i-1].Key$ 和 $r[j].Key$, $j = i, i+1, \dots, n-i-1$ 。如果发生逆序，则交换 $r[i-1]$ 和 $r[j]$ 。





第四趟



起泡排序的算法

```
void BubbleSort(SqList &L)
{for (int i=L.length, change=TRUE;i>1 && change; --i)
 { change = FALSE;
  for (int j=1; j<i; ++j)
   if (LT(L.r[j+1].key,L.r[j].key))
    { ElemType temp=L.r[j];
     L.r[j]=L.r[j+1];
     L.r[j+1]=temp;
     change = TRUE;
    }
 }
```

第五趟



特 点

至少比较1次，至多比较 $n-1$ 次；
一次确定位置；
可实现部分排序；
稳定排序。



6.5 快速排序 (Quick Sort)

- 快速排序方法的基本思想是任取待排序对象序列中的某个对象(例如取第一个对象)作为**枢轴(pivot)**，按照该对象的关键字大小，将整个对象序列划分为左右两个子序列：
 - 左侧子序列中所有对象的关键字都小于或等于枢轴对象的关键字
 - 右侧子序列中所有对象的关键字都大于枢轴对象的关键字
- 枢轴对象则排在这两个子序列中间(这也是该对象最终应安放的位置)。

$k_1, k_2, \dots, k_s, pivot, k'_1, k'_2, \dots, k'_t$



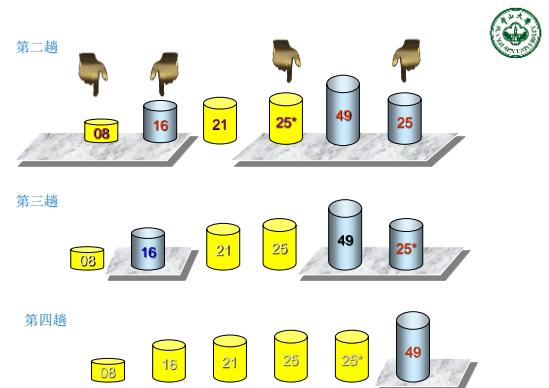
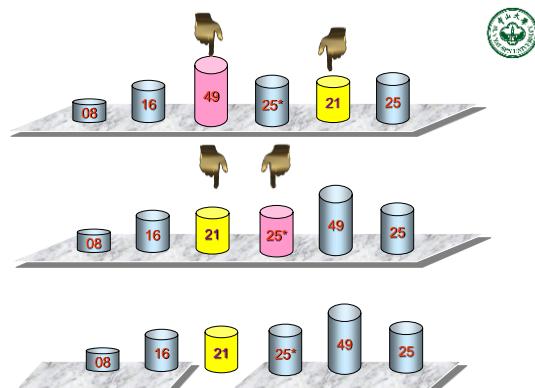
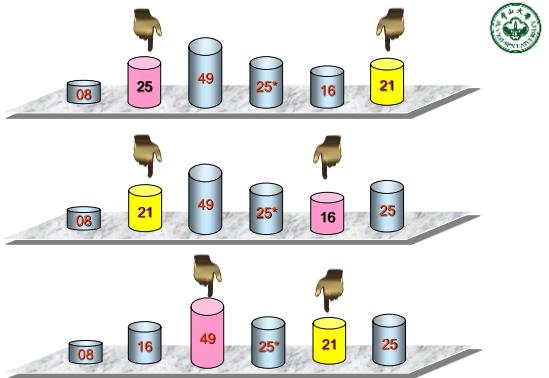
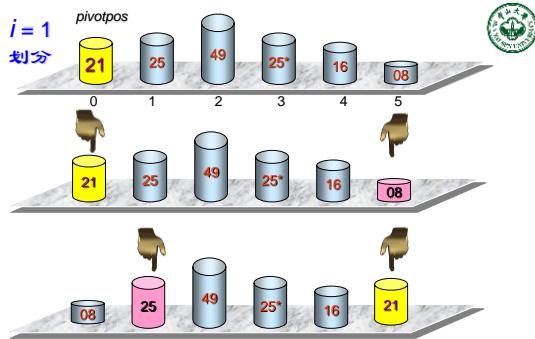
- 然后分别对这两个子序列重复施行上述方法，直到所有的对象都排在相应位置上为止。
- 算法描述

```
QuickSort ( List )
  if ( List 的长度大于 1 ) {
    将序列 List 划分为两个子序列
    LeftList 和 RightList;
    QuickSort ( LeftList );
    QuickSort ( RightList );
    将两个子序列 LeftList 和 RightList
    合并为一个序列 List;
  }
```

枢纽的确定方法

$k_1, k_2, \dots, k_s, pivot, k'_1, k'_2, \dots, k'_t, K, k''_1, k''_2, \dots, k''_p$





快速排序的算法

```

void QSort(SqList &L, int low, int high){
    if (low < high){
        int pivotloc = Partition(L,low,high);
        QSort(L,low, pivotloc-1);
        PrintST(L);
        QSort(L,pivotloc+1, high);
        PrintST(L); }
}
void QuickSort(SqList &L){
    QSort(L,1,L.length);
}

```

```

int Partition(SqList &L, int low, int high)
{ L.r[0] = L.r[low];
  int pivotkey = L.r[low].key;
  while (low < high)
  { while (low<high && L.r[high].key >= pivotkey) --high;
    L.r[low] = L.r[high];
    while (low<high && L.r[low].key <= pivotkey) ++low;
    L.r[high] = L.r[low];
  }
  L.r[low]=L.r[0];
  return low;
}

```



算法分析

- 从快速排序算法的递归树可知，快速排序的趟数取决于递归树的深度。
- 如果每次划分对一个对象定位后，该对象的左侧子序列与右侧子序列的长度相同，则下一步将是对两个长度减半的子序列进行排序，这是最理想的情况。
- 在 n 个元素的序列中，对一个对象定位所需时间为 $O(n)$ 。若设 $t(n)$ 是对 n 个元素的序列进行排序所需的时间，而且每次对一个对象正确定位后，正好把序列划分为长度相等的两个子序列，此时，总的计算时间为：



$$\begin{aligned}
 T(n) &\leq cn + 2 t(n/2) \quad // c \text{ 是一个常数} \\
 &\leq Cn + 2 (cn/2 + 2t(n/4)) = 2cn + 4t(n/4) \\
 &\leq 2cn + 4 (cn/4 + 2t(n/8)) = 3cn + 8t(n/8) \\
 &\dots\dots \\
 &\leq Cn \log_2 n + nt(1) = O(n \log_2 n)
 \end{aligned}$$

- 可以证明，函数 `quicksort` 的平均计算时间也是 $O(n \log_2 n)$ 。实验结果表明：就平均计算时间而言，快速排序是我们所讨论的所有内排序方法中最好的一个。

- 在最坏的情况下，即待排序对象序列已经按其关键字从小到大排好序的情况下，其递归树成为单支树，每次划分只得到一个比上一次少一个对象的子序列。这样，必须经过 $n-1$ 趟才能把所有对象定位，而且第 i 趟需要经过 $n-i$ 次关键字比较才能找到第 i 个对象的安放位置，总的关键字比较次数将达到

$$\sum_{i=1}^{n-1} (n-i) = \frac{1}{2}n(n-1) \approx \frac{n^2}{2}$$



- 快速排序是递归的，需要有一个栈存放每层递归调用时的指针和参数。
- 最大递归调用层数与递归树的深度一致，理想情况为 $\lceil \log_2(n+1) \rceil$ 。因此，要求存储开销为 $O(\log_2 n)$ 。



- 其排序速度退化到简单排序的水平，比直接插入排序还慢。占用附加存储(即栈)将达到 $O(n)$ 。
- 若能更合理地选择基准对象，使得每次划分所得的两个子序列中的对象个数尽可能地接近，可以加速排序速度，但是由于对象的初始排列次序是随机的，这个要求很难办到。
- 有一种改进办法：取每个待排序对象序列的第一个对象、最后一个对象和位置接近正中的3个对象，取其关键字居中者作为基准对象。





6.6 选择排序(Selection Sort)



- 快速排序是一种不稳定的排序方法。
- 对于 n 较大的平均情况而言，快速排序是“快速”的，但是当 n 很小时，这种排序方法往往比其它简单排序方法还要慢。

选择排序的基本思想是：每一趟（例如第 i 趟， $i = 1, \dots, n-1$ ）在后面的 $n-i+1$ 个待排序对象中选出关键字最小的对象，作为有序对象序列的第 i 个对象。待到第 $n-1$ 趟作完，待排序对象只剩下 1 个，就不用再选了。

简单选择排序 (Simple Selection Sort)

- 基本步骤为： i 从 1 开始，直到 $n-1$ ，进行 $n-1$ 趟排序，第 i 趟的排序过程为：在一组对象 $r[i] \sim r[n]$ ($i=1, 2, \dots, n-1$) 中选择具有最小关键字的对象；并和第 i 个对象进行交换；

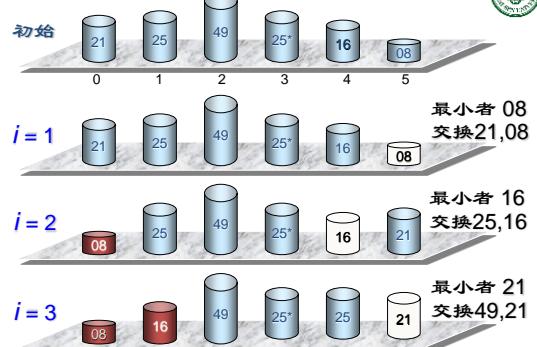


简单选择排序的算法

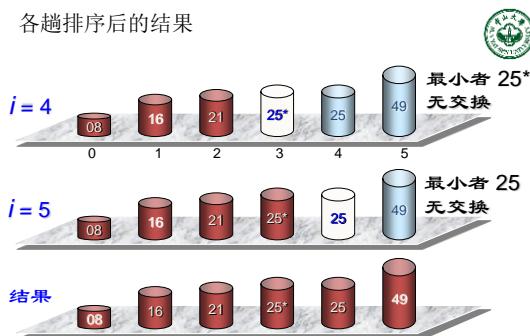
```
void SelectSort(Sqlist &L)
{ for (int i=1; i<L.length; ++i)
    { int k=i;
        for (int j=i+1; j<=L.length; ++j)
            if (L.r[k].key > L.r[j].key) k=j;
        if (i!=k) { ElemenType temp=L.r[i];
                    L.r[i]=L.r[k];
                    L.r[k]=temp;
                }
    }
}
```



各趟排序后的结果



各趟排序后的结果



算法分析

- 直接选择排序的关键字比较次数 KCN 与对象的初始排列无关。第 i 趟选择具有最小关键字对象所需的比较次数总是 $n-i-1$ 次，此处假定整个待排序对象序列有 n 个对象。因此，总的关键字比较次数为

$$KCN = \sum_{i=0}^{n-2} (n-i-1) = \frac{n(n-1)}{2}$$



- 对象的移动次数与对象序列的初始排列有关。当这组对象的初始状态是按其关键字从小到大有序的时候，对象的移动次数 $RMN = 0$ ，达到最少。
- 最坏情况是每一趟都要进行交换，总的对像移动次数为 $RMN = 3(n-1)$ 。**
- 直接选择排序是一种**不稳定的**排序方法。

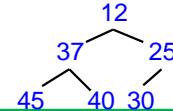


6.7 堆排序 (Heap Sort)

- 堆：在一个关键字序列 $(k_1 < k_2 < \dots < k_n)$ 中，如果满足： $k_i <= k_{2i}$ 且 $k_i <= k_{2i+1}$, $1 \leq i \leq n/2$ 则该序列称为堆。

- 堆的特点：堆顶为序列中最大（或最小）值。

12,37,25,45,40,30



6.8 归并排序 (Merge Sort)

- 利用堆的概念，可以很容易地实现选择排序的思路。堆排序分为两个步骤：第一步，根据初始输入数据，利用堆的调整算法形成初始堆，输出堆顶元素。第二步，重新调整剩余元素使之成为堆。重复以上操作。



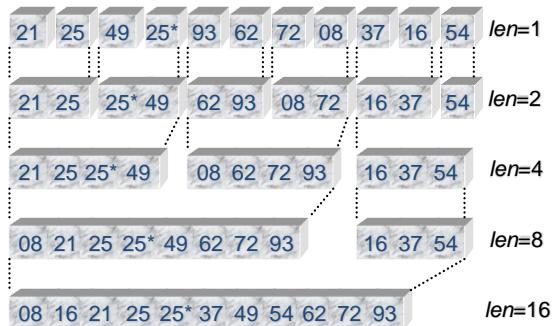
- 归并，是将两个或两个以上的有序表合并成一个新的有序表。
- 对象序列 *initList* 中有两个有序表 $V[1] \dots V[m]$ 和 $V[m+1] \dots V[n]$ 。它们可归并成一个有序表，存于另一对象序列 *mergedList* 的 $V[1] \dots V[n]$ 中。
- 这种归并方法称为**2路归并** (2-way merging)。
- 其基本思想是：设两个有序表A和B的对象个数(表长)分别为 al 和 bl ，变量 i 和 j 分别是表A和表B的当前检测指针。设表C是归并后的新有序表，变量 k 是它的当前存放指针。

归并排序算法



- 迭代的归并排序算法就是利用两路归并过程进行排序的。其基本思想是：
- 假设初始对象序列有 n 个对象，首先把它看成是 n 个长度为 1 的有序子序列 (归并项)，先做两两归并，得到 $\lceil n/2 \rceil$ 个长度为 2 的归并项 (如果 n 为奇数，则最后一个有序子序列的长度为 1)；再做两两归并，...，如此重复，最后得到一个长度为 n 的有序序列。

归并排序算法



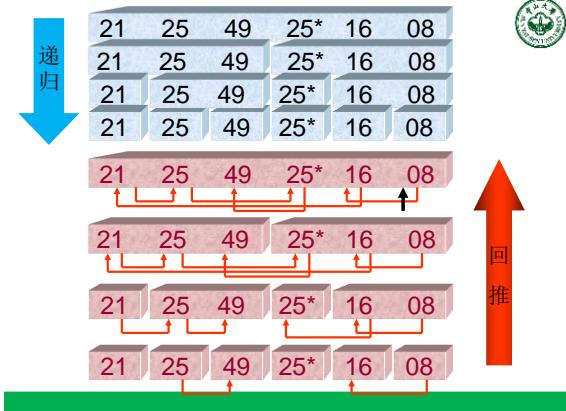


两路归并算法

```
void Merge(ElemType SR[],ElemType TR[],int i,
int m,int n){
    for (int j=m+1,k=i;j<=m && j<=n; ++k){
        if (LQ(SR[i].key ,SR[j].key))
            TR[k] = SR[i++];
        else TR[k] = SR[j++];
    }
}
```

```
if (i<=m)
for (int n1=k,n2=i;n1<=n &&
     n2<=m;n1++,n2++)
    TR[n1]=SR[n2];
if (j<=n)
for (int n1=k,n2=j;n1<=n &&
     n2<=n;n1++,n2++)
    TR[n1]=SR[n2];
}
```

```
void MSort(ElemType SR[ ],ElemType TR1[],int s,int t){
    ElemType TR2[MAXSIZE];
    if (s==t) TR1[s]=SR[s];
    else {
        int m=(s+t)/2;
        MSort(SR,TR2,s,m);
        MSort(SR,TR2,m+1,t);
        Merge(TR2,TR1,s,m,t);
    }
}
void MergeSort(SqList &L){
    MSort(L.r,L.r,1,L.length);
}
```



算法分析

- 在归并排序算法中，**递归深度为 $O(\log_2 n)$** ，对象关键字的比较次数为 **$O(n \log_2 n)$** 。算法总的时间复杂度为 **$O(n \log_2 n)$** 。
- 归并排序占用附加存储较多，需要另外一个与原待排序对象数组同样大小的辅助数组。这是这个算法的缺点。
- 归并排序是一个稳定的排序方法。

6.9 基数排序 (Radix Sort)

- 基数排序是采用“**分配**”与“**收集**”的办法，用对多关键字进行排序的思想实现对单关键字进行排序的方法。

多关键字排序



- 以扑克牌排序为例。每张扑克牌有两个“关键字”：花色和面值。其有序关系为：
 - ◆ 花色： $\clubsuit < \diamond < \heartsuit < \spadesuit$
 - ◆ 面值： $2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J < Q < K < A$
- 如果我们把所有扑克牌排成以下次序：
 $\clubsuit 2, \dots, \clubsuit A, \diamond 2, \dots, \diamond A, \heartsuit 2, \dots, \heartsuit A, \spadesuit 2, \dots, \spadesuit A$

- 这就是**多关键字排序**。排序后形成的有序序列叫做词典有序序列。
- 对于上例两关键字的排序，可以先按花色排序，之后再按面值排序；也可以先按面值排序，再按花色排序。
- 一般情况下，假定有一个 n 个对象的序列 $\{V_0, V_1, \dots, V_{n-1}\}$ ，且每个对象 V_i 中含有 d 个关键字 $(K_i^1, K_i^2, \dots, K_i^d)$
- 如果对于序列中任意两个对象 V_i 和 V_j ($0 \leq i < j \leq n-1$) 都满足：

$$(K_i^1, K_i^2, \dots, K_i^d) < (K_j^1, K_j^2, \dots, K_j^d)$$

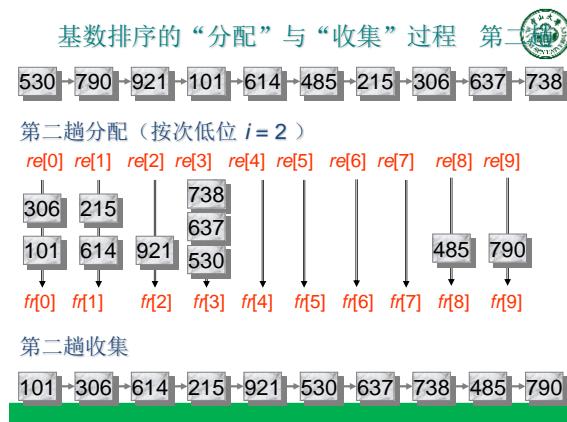
- 则称序列对关键字 (K^1, K^2, \dots, K^d) 有序。其中，
 K^1 称为最高位关键字， K^d 称为最低位关键字。
- 如果关键字是由多个数据项组成的数据项组，则依据它进行排序时就需要利用多关键字排序。
- 实现多关键字排序有两种常用的方法
 - 最高位优先MSD (Most Significant Digit first)
 - 最低位优先LSD (Least Significant Digit first)



- 最高位优先法**通常是一个递归的过程：
 - 先根据最高位关键字 K^1 排序，得到若干对象组，对象组中每个对象都有相同关键字 K^1 。
 - 再分别对每组中对象根据关键字 K^2 进行排序，按 K^2 值的不同，再分成若干个更小的子组，每个子组中的对象具有相同的 K^1 和 K^2 值。
 - 依此重复，直到对关键字 K^d 完成排序为止。
 - 最后，把所有子组中的对象依次连接起来，就得到一个有序的对象序列。

- 最低位优先法**首先依据最低位关键字 K^d 对所有对象进行一趟排序，再依据次低位关键字 K^{d-1} 对上一趟排序的结果再排序，依次重复，直到依据关键字 K^1 最后一趟排序完成，就可以得到一个有序的序列。使用这种排序方法对每一个关键字进行排序时，不需要再分组，而是整个对象组都参加排序。





算法分析

- 若每个关键字有 d 位，需要重复执行 d 趟“分配”与“收集”。每趟对 n 个对象进行“分配”，对 $radix$ 个队列进行“收集”。总时间复杂度为 $O(d(n+radix))$ 。
- 若基数 $radix$ 相同，对于对象个数较多而关键位数较少的情况，使用链式基数排序较好。
- 基数排序需要增加 $n+radix$ 个附加链接指针。
- 基数排序是稳定的排序方法。

小结

- 排序的基本概念**
 - 排序的基本概念
 - 关键字、初始关键字排列
 - 关键字比较次数、数据移动次数
 - 稳定性
 - 附加存储
- 插入排序**
 - 直接插入排序和折半插入排序的算法
 - 排序的性能分析

- 当待排序的关键字序列已经基本有序时，用直接插入排序最快
- 选择排序**
 - 直接选择排序和堆排序的算法
 - 选择排序的性能分析
- 用直接选择排序在一个待排序区间中选出最小的数据时，与区间第一个数据对调，不是顺次后移。这导致方法不稳定。

- 在堆排序中将待排序的数据组织成完全二叉树的顺序存储。
- 交换排序**
 - 用事例表明起泡排序和快速排序的过程
 - 起泡排序的算法
- 快速排序是一个递归的排序法
- 当待排序关键字序列已经基本有序时，快速排序显著变慢。



各种排序方法的比较



■ 二路归并排序

- ◆ 用事例表明二路归并排序的过程
- ◆ 二路归并排序的非递归算法
- ◆ 该算法的性能分析
- 归并排序可以递归执行
- 归并排序需要较多的附加存储。
- 归并排序对待排序关键字的初始排列不敏感，故排序速度较稳定。

排序方法	比较次数		移动次数		稳定性	附加存储	
	最好	最差	最好	最差		最好	最差
直接插入排序	n	n^2	0	n^2	✓	1	
起泡排序	n	n^2	0	n^2	✓	1	
快速排序	$n \log n$	n^2	$n \log n$	n^2	✗	$\log n$	n^2
简单选择排序	n^2		0	n	✗	1	
堆排序	$n \log n$		$n \log n$		✗	1	
归并排序	$n \log n$		$n \log n$		✓	n	



6.10 文件管理

- ◆ 内部排序过程中不涉及数据的内、外存交换，待排序的记录全部存放在内存中；
- ◆ 若待排序的文件很大，就无法将整个文件的所有记录同时调入内存进行排序；
- ◆ 外部排序的实现，主要是依靠数据的内、外存交换和“内部归并”。
- ◆ 外部排序基本上包括相对独立的两个阶段：初始归并段的形成；多路归并。
- ◆ 外部排序主要研究的技术问题是：
 - 如何进行多路归并以减少文件的归并遍数；
 - 如何运用内存的缓冲区使I/O和CPU尽可能并行工作；
 - 根据外存的特点选择较好的产生初始归并段的方法。



6.10.1 文件的基本概念



1、文件的基本概念

- (1) **数据项(Item或field)**：数据文件中最小的基本单位，反映实体某一方面的特征—属性的数据表示。
- (2) **记录(Record)**：一个实体的所有数据项的集合。用来标识一个记录的数据项集合(一个或多个)称为关键字项(Key)，关键字项的值称为关键字；能唯一标识一个记录的关键字称**为主关键字(Primary Key)**，其它的关键字称为**次关键字(Secondary Key)**。

文件存储在外存上，通常是以块(I/O读写的基本单位，称为**物理记录**)存取。

(3) **文件(File)**：大量性质相同的数据记录的集合。文件的所有记录是按某种排列顺序呈现在用户面前，这种排列顺序可以是按记录的关键字，也可以是按记录进入文件的先后等。则记录之间形成一种线性结构(逻辑上的)，称为文件的**逻辑结构**；文件在外存上的组织方式称为文件的**物理结构**。

基本的物理结构有：**顺序结构**，**链接结构**，**索引结构**。



2、文件的分类

- (1) 按记录类型, 可分为操作系统文件和数据库文件:
- ① 操作系统文件(流式文件): 连续的字符序列(串)的集合;
 - ② 数据库文件: 有特定结构(所有记录的结构都相同)的数据记录的集合。
- (2) 按记录长度, 可分为定长记录文件和不定长记录文件:
- ① 定长记录文件: 文件中每个记录都有固定的数据项组成, 每个数据项的长度都是固定的;
 - ② 不定长记录文件: 与定长记录文件相反。

3、文件的有关操作

文件是由大量记录组成的线性表, 因此, 对文件的操作主要是针对记录的, 通常有: 记录的检索、插入、删除、修改和排序, 其中检索是最基本的操作。

(1) 检索记录

根据用户的要求从文件中查找相应的记录。

- ① 查找下一个记录: 找当前记录的下一个逻辑记录;
- ② 查找第k个记录: 给出记录的逻辑序号, 根据该序号查找相应的记录;
- ③ 按关键字查找: 给出指定的关键字值, 查找关键字值相同或满足条件的记录。

对数据库文件, 有以下四种按关键字查找的方式:

- ◆ 简单匹配: 查找关键字的值与给定的值相等的记录;
- ◆ 区域匹配: 查找关键字的值在某个区域范围内的记录;
- ◆ 函数匹配: 给出关键字的某个函数, 查找符合条件的记录;
- ◆ 组合条件匹配: 给出用布尔表达式表示的多个条件组合, 查找符合条件的记录。

(2) 插入记录

将给定的记录插入到文件的指定位置。插入是首先要确定插入点的位置(检索记录), 然后才能插入。

(3) 删除记录

从文件中删除给定的记录。记录的删除有两种情况:

- ① 在文件中删除第k个记录;
- ② 在文件中删除符合条件的记录。

(4) 修改记录

对符合条件的记录, 更改某些属性值。修改时首先要检索到所要修改的记录, 然后才能修改。

(5) 记录排序

根据指定的关键字, 对文件中的记录按关键字值的大小以非递减或非递增的方式重新排列(或存储)。

6.10.2 文件的组织方式



文件的组织方式指的是文件的物理结构。

1、顺序文件

记录按其在文件中的逻辑顺序依次进入存储介质。在顺序文件中, 记录的逻辑顺序和存储顺序是一致的。

- (1) 根据记录是否按关键字排序: 可分为排序顺序文件和一般顺序文件;
- (2) 根据逻辑上相邻的记录的物理位置关系: 可分为连续顺序文件和链接顺序文件。

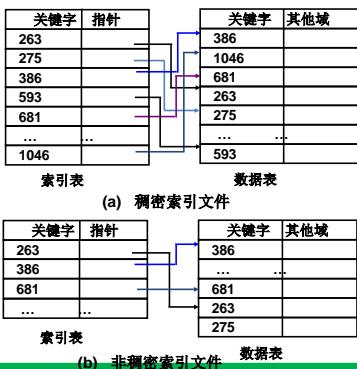
顺序文件类似于线性表的顺序存储结构, 比较简单, 适合于顺序存取的外存介质, 但不适合随机处理。

2、索引文件

索引技术是组织大型数据库的一种重要技术, 索引是记录和记录存储地址之间的对照表。索引结构(称为索引文件)由索引表和数据表两部分, 如图所示。



索引结构的基本形式



◆ **数据表**: 存储实际的数据记录;

◆ **索引表**: 存储记录的**关键字**和**记录(存储)地址**之间的对照表, 每个元素称为一个**索引项**。

如果数据文件中的每一个记录都有一个索引项, 这种索引称为**稠密索引**, 否则, 称为**非稠密索引**。

对于非稠密索引, 通常将文件记录划分为若干块, **块内**记录可以**无序**, 但**块间必须有序**。若块内记录是有序的, 称为索引顺序文件, 否则称为索引非顺序文件。对于索引非顺序文件, 只需对每一块建立一个索引项。



对于**稠密索引**, 可以根据索引项直接查找到记录的位置。

若在索引表中采用**顺序查找**, 查找时间复杂度为 $O(n)$; 若采用**折半查找**, 查找时间复杂度为 $O(\log_2 n)$ 。

对于**稠密索引**, 索引项数目与数据表中记录数相同, 当索引表很大时, 检索记录需多次访问外存。

对于**非稠密索引**, 查找的基本思想是:

首先根据索引找到记录所在块, 再将该块读入到内存, 然后再在块内顺序查找。

平均查找长度由两部分组成: 块地址的平均查找长度 L_b , 块内记录的平均查找长度 L_w , 即 $ASL_{bs} = L_b + L_w$

若将长度为 n 的文件分为 b 块, 每块内有 s 个记录, 则 $b=n/s$ 。设每块的查找概率为 $1/b$, 块内每个的记录查找概率为 $1/s$, 则采用顺序查找方法时有:



3、散列文件



散列文件(直接存取文件): 利用散列存储方式组织的文件。类似散列表, 即根据文件中记录关键字的特点, 设计一个散列函数和冲突处理方法, 将记录散列到存储介质上。

在散列文件中, 磁盘上的记录是成组存放的, 若干个记录组成一个存储单位, 称为**桶(Bucket)**, 同一个桶中的记录都是同义词(关键字的角度)。

设一个桶中能存放 m 个记录, 当桶中已有 m 个同义词的记录时, 要存放第 $m+1$ 个同义词就“溢出”。冲突处理方法一般是**拉链法**。

$$ASL_{bs} = L_b + L_w = (b+1)/2 + (s+1)/2 = (n/s+s)+1$$

显然, 当 $s=n^{1/2}$ 时, ASL_{bs} 的值达到最小;

若在索引表中采用折半查找方法时有:

$$ASL_{bs} = L_b + L_w = \log_2(n/s+1) + s/2$$

如果文件中记录数很庞大, 对非稠密索引而言, 索引也很大, 可以将索引表再分块, 建立**索引的索引**, 形成树形结构的多级索引, 如后面将要介绍的**ISAM文件**和**VSAM文件**。

检索记录时，先根据给定值求出散列桶地址，将基桶的记录读入内存进行顺序查找，若找到关键字等于给定值的记录，则查找成功；否则，依次读入各溢出桶中的记录继续进行查找。

在散列文件中删除记录，是对记录加删除标记。

散列文件的特点

(1) 优点

- ◆ 文件随机存取，记录不需进行排序；
- ◆ 插入、删除方便，存取速度快；
- ◆ 不需要索引区，节省存储空间。

(2) 缺点

- ◆ 不能进行顺序存取，只能按关键字随机存取；
- ◆ 检索方式仅限于简单查询。