BUPT

第八章 排序

杨震　计算机学院

8.1 基本概念与术语

排序是根据记录关键字的值的递增(递减)的关系将文件记录的次序重新排列。

[定义]

设有含n个记录的序列 {R1, R2,..., Rn},

对应的关键字序列为 {K1, K2,..., Kn},

一个置换 p1,p2,...,pn

使记录序列按关键字有序 {Rp1, Rp2,..., Rpn},

满足 Kp1≤ Kp2≤... ≤ Kpn (正序)

或 Kp1≥ Kp2 ≥...≥ Kpn

排序的分类

[1. 根据排序时文件记录的存放位置]

内部排序：排序过程中将全部记录放在内存中处理。

外部排序：排序过程中需在内外存之间交换信息。

[2. 根据排序前后相同关键字记录的相对次序]

稳定排序：设文件中任意两个记录的关键字值相同，即Ki=Kj(i≠j)，若排序之前记录Ri领先于记录Rj，排序后这种关系不变(对所有输入实例而言)。 看相对位置

不稳定排序：只要有一个实例使排序算法不满足稳定性要求。

[3. 根据文件的存储结构划分排序的种类]

连续顺序文件排序

链表排序 简单实现

地址排序: 待排记录顺序存储，排序时只对辅助表(关键字+指针)的表目进行物理重排。

[4. 根据排序的方法]

插入排序　　交换排序

选择排序　　归并排序

基数排序

[5. 根据排序算法所需的辅助空间]

就地排序： O(1)　非就地排序： O(n)或与n有关

评价排序算法的主要标准

[时间开销]

考察算法的两个基本操作的次数：

- 比较关键字
- 移动记录

算法时间还与输入实例的初始状态有关时，分情况：

- 最好
- 最坏
- 平均

[空间开销]

所需的辅助空间

约定

顺序方式存储时，结构如下

```
# define MAXSIZE  最大记录个数
typedef int KeyType;

typedef struct {
  KeyType  key;
  InfoType otherinfo;
} RedType;

typedef struct {
  RedType   r [MAXSIZE + 1 ] ; // r[0] 空或作哨兵
  int              length;
} SqList;
```

直接插入排序（增量法）

稳定排序 复杂度与待排情况有关 链式方式记录

[示例] { R(0) R(-4) R(8) R(1) R(-4) R(-6) } n=6

关键字

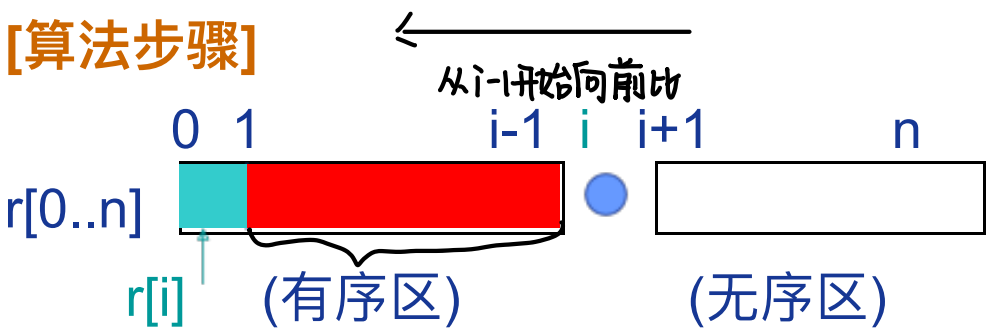
假设第1个有序 从后插在0前 无序 n-1趟排序

i=1	[0]	-4	8	1	-4	-6
i=2	[-4	0]	8	1	-4	-6
i=3	[-4	0	8]	1	-4	-6
i=4	[-4	0	1	8]	-4	-6
i=5	[-4	-4	0	1	8]	-6
i=6	[-6	-4	-4	0	1	8]

稳定的排序算法

[算法思想] 每次使有序区增加一个记录

[算法步骤]



设计程序更简洁 先放到r[0] r[0]与它们相比

循环(n-1)次，初值 i=2， r[0]用作哨兵。(缓存)

1) 把第i个记录取出保存在r[0]中，j=i-1

2)若r[0]< r[j]，则r[j]后移一位，j=j-1，转2)；

否则r[0]放在r[j+1]处，i=i+1，转1)

要和有序区记录相比 比较次数可能不同

比1号还小 { 相等，稳定算法 (后插入在后) 比所有记录都小 比完再和0号单元比 相等 插入后面

最好 2次 移到0.再移回去

最坏 i+1次 (i-1)+1+1 { 3 / n+1

i取值 2~n

[算法描述]

```
void InsertSort ( SqList &L )
{ for ( i = 2;  i <= L.length ; ++i )
      if  ( LT( L.r[i].key,  L.r[i-1].key ) )
        {  L.r[0].key = L.r[i].key;
           for ( 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];
        }
}
```

比较最好 1 O(n)

最坏 i (还有和0号单元相比)

$O(n^2)$

[哨兵/监视哨的作用]

简化边界条件的测试，提高算法时间效率。

[性能分析]

- 最好情况(原始数据按**正序**即非递减序排列)

  Cmin=n-1　Mmin= 2(n-1)

- 最坏情况(原始数据按**逆序**即非递增序排列)

  Cmax=(n+2)(n-1)/2　　Mmax=(n+4)(n-1)/2

- 随机情况

  Cavg=(Cmin+ Cmax)/2≈n2/4　　Mavg ≈n2/4

- 时间复杂度O(n2)

  辅助空间复杂度O(1)

8.3 交换排序

起泡排序(冒泡排序)

相邻比大小换位置（3次记录的移动）
（1次记录的比较）

[算法思想]

将两个相邻记录的关键字进行比较，若为逆序则交换两者位置，小者往上浮，大者往下沉。

最后n的位置上一定是最大值

[算法步骤]

记录1和2、2和3、……、(n-1)和n的关键字比较(交换)；

记录1和2、2和3、……、(n-2)和(n-1)的关键字比较(交换)；

……

终止条件：直到某一趟不出现交换操作为止。

[示例]

<初态>	<第一趟>	<第二趟>	<第三趟>	<第四趟>
0	-4	-4	-6	**-6**
-4	0	-6	-4	-4
8	-6	-4	-4	**-4**
-6	-4	0	**0**	**0**
-4	1	**1**	**1**	**1**
1	**8**	**8**	**8**	**8**
sorted	F	F	F	T

稳定排序

起泡一趟未全部排好序，∴仍需对前(n-1)个进行起泡

没有交换记录的操作，∴结束

[性能分析]

- 最好情况(原始数据按正序即非递减序排列) O(n)

  Cmin=n-1　Mmin=0

- 最坏情况(原始数据按逆序即非递增序排列) $O(n^2)$

  Cmax=n(n-1)/2　Mmax=3n(n-1)/2

  (n-1)趟才结束　移动次数是比较次数的3倍　只要比较都会交换位置

- 时间复杂度O(n2)

  辅助空间复杂度O(1)

  只在交换位置时需要辅助空间

[算法的改进]

- 每趟排序中，记录最后一次发生交换的位置 (往后的记录不用参加下一趟排序)
- 双向交替起泡，上□下，最轻升顶，上□下，最重沉底

思考：双向起泡算法如何实现?

```
void BubbleSort2(int a[],int n) //相邻两趟向相反方向起泡的
    冒泡排序算法
{ change=1;low=0;high=n-1; //冒泡的上下界
      while(low<high && change)
      { change=0;              //设不发生交换
        for(i=low;i<high;i++)  //从上向下起泡
          if(a[i]>a[i+1]){a[i]<-->a[i+1];change=1;} //有交换，修
    改标志change
        high--; //修改上界
        for(i=high;i>low;i--) //从下向上起泡
          if(a[i]<a[i-1]){a[i]<-->a[i-1];change=1;}
        low++; //修改下界
      }//while
}
```

思考：如以双链表存储记录，双向起泡算法如何实现?

快速排序(分划交换排序/分治法)

递归原理

[分治算法原理]

1)分解：将原问题分解为若干子问题

2)求解：递归地解各子问题，若子问题的规模足够小，则直接求解

3)组合：将各子问题的解组合成原问题的解

答案

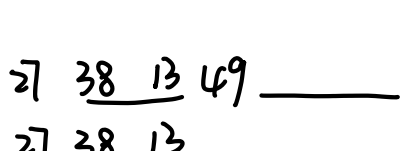
[快速排序算法思想]

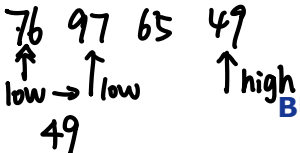
默认

指定**枢轴/支点/基准记录**r[p](通常为第一个记录)，通过一趟排序将其放在正确的位置上，它把待排记录分割为独立的两部分，使得

左边记录的关键字 ≤ **r[p].key** ≤ 右边记录的关键字

对左右两部分记录序列重复上述过程，依次类推，直到子序列中只剩下一个记录或不含记录为止。(可以用递归方法实现)





[示例]

低指针low　　高指针high

<初态>　{ (49)　38　65　97　76　13　27　49 }

x=49 low　　high

27　38　65　97　76　13　( )　49

从high指针位置往前找，与基准指针比较。high--

low　　high

27　38　( )　97　76　13　65　49

low 放到low当前位置　high

27　38　13　97　76　( )　65　49

low　　high　97放到high当前位置

27　38　13　( )　76　97　65　49

low　　high

<一趟结束> {27　38　13}　49　{76　97　65　49 }

最后在此放入基准记录。

(low =high)

```
Void QuickSort ( SqList &L )
{   QSort ( L, 1, L.length ); }  // QuickSort
    // 对顺序表 L 进行快速排序
```

```
Void QSort ( SqList &L,  int low,  int  high )
{  if  ( low < high )
   {  pivotloc = Partition(L, low, high ) ;
      Qsort (L, low, pivotloc-1) ;
      Qsort (L, pivotloc+1, high )
    }
}  // QSort
```

```
int Partition ( SqList &L,  int low,  int  high )
{  L.r[0] = L.r[low]; 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;
}  // Partition
```

[性能分析]

最坏情况(原始数据正/逆序排列)

Cmax=n(n-1)/2　　Mmax ≤ Cmax　　**O(n2)**

最好情况：每次划分的结果是基准的左、右两个无序子区间的长度大致相等

基准:min { 左 / n-1 右　划分(n-1)趟，每一趟基准记录与其他记录做比较，共(n-1)次

Cmin ≤ O(nlgn)　　Mmin ≤ Cmin　　**O(nlog2n)**

平均时间性能　Tavg(n)=kn ln(n)

k：某个常数；　n：待排序序列中记录个数

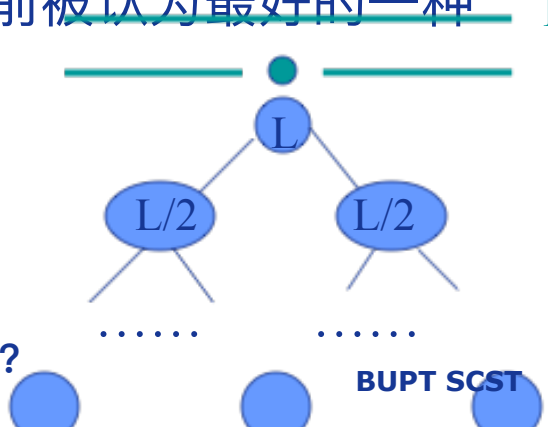
就平均时间而言，快速排序是目前被认为最好的一种内部排序方法。

每一次划分

二叉树高度为N.　递归高度：4

辅助空间复杂度　最好情况 O(log2 n)　恰好二分

最坏情况 O(n)



快速排序的最大递归深度是多少？最小递归深度是多少？

[算法的改进]

合理选择枢轴记录可改善性能。例如，

- 三者取中 比大小取中间值
- 随机产生

[算法的稳定性]

是非稳定排序

反例 [2，2，1]：{1 2 } **2** { }

思考：分别用单链表、双向链表存储记录，快排算法如何实现？？

简单选择排序(直接选择排序)

[算法步骤]

第1趟：从n个记录中选关键字最小的记录，与第1个记录交换；

n-1次比较

第2趟：从剩余的n-1个记录中选关键字最小的记录，与第2个记录交换；

n-2次

……

第i趟：从剩余的n-i+1个记录中选关键字最小的记录，与第i个记录交换；

…… 一定会进行(n-1)趟

直到第n-1趟执行完为止。

[算法描述]

$\sum_{i=1}^{n-1}(n-i)(n-1)$ 比较次数 $O(n^2)$

移动次数 $O(n)$

在任何情况下复杂度都是 $O(n^2)$

```
void SelectSort ( SqList &L )
{   for ( i =1; i < L.length; ++i )
   { j = SelectMinKey ( L, i );
     if ( i != j ) r[i] <=> r[j];
    }
} // SelectSort

int SelectMinKey (SqList  L; int i)
{
     k = i;
     for  (j = i + 1;j<=n;j--)
          if (r [ j ] .key < r [ k ].key) k = j;
     return k;
}
```

[示例] （n=8）

<初态>	49	38	65	97	76	49	13	27
<第1趟>	13	38	65	97	76	49	49	27
<第2趟>	13	27	65	97	76	49	49	38
<第3趟>	13	27	38	97	76	49	49	65
<第4趟>	13	27	38	49	76	97	49	65
<第5趟>	13	27	38	49	49	97	76	65
<第6趟>	13	27	38	49	49	65	76	97
<第7趟>	13	27	38	49	49	65	76	97

（每趟排序使有序区增加一个记录）

不稳定排序

[性能分析]

- 总的比较次数与记录排列的初始状态无关

  C=(n-1)+(n-2)+...+2+1=n(n-1)/2

- 移动次数

  初始记录逆序时：Mmax = 3(n-1)

  初始记录正序时：Mmin = 0

- 平均时间复杂度O(n2)

  辅助空间复杂度O(1)

8.5 归并排序

[归并的概念]

指将两个或两个以上的同序序列归并成一个序列的操作。

1 两路归并排序

[算法思想]

第1趟：将待排序列R[1..n]看作n个长度为1的有序子序列，两两归并，得到⌈n/2⌉个长度为2的有序子序列(或最后一个子序列长度为1)；

第2趟：将上述⌈n/2⌉个有序子序列两两归并；

……

直到合并成一个序列为止。

[示例]　(n=7)

<初态>　(49)　(38)　(65)　(97)　(76)　(49)　(13)

<第1趟>　(38　49)　(65　97)　(49　76)　(13)

<第2趟>　(38　49　65　97)　(13　49　76)

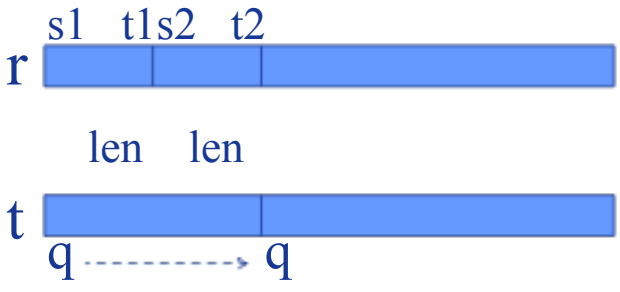
<第3趟>　(13　38　49　49　65　76　97)

[性能分析]

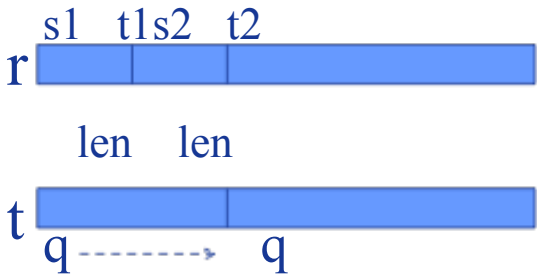
- 任何情况时间复杂度O(nlog2n)
- 空间复杂度O(n)
- 很少用于内部排序

[算法描述]

```
void  mergesort ( SqList &r ) {
    len = 1;
    While  (len < n)
    {      s1 = 1;     q = 1;
           While  (s1 + len <= n)
           {    s2= s1 +len;          t1 = s2 - 1;          t2 = s2 + len -1;}
                If  (t2 > n) t2 = n;
                merge ( r, t, s1, t1, s2, t2, q);          s1 = q;
           }
           for  (i = 1; i <= q – 1) r [ i ] = t [ i ];
           len = len*2;
    }
```



```
void  merge (SqList r, &t ; int s1, t1, s2, t2, int &q) {
   while  (( s1 <= t1 )  &&  ( s2 <= t2 ) )
    {      If   (r [ s1 ] <= r  [ s2 ])
                 {   t [ q ] = r [ s1 ];    s1 = s1 + 1;   }
           else
                 {  t [ q ] = r [ s2 ];    s2 = s2 + 1;   }
           q = q + 1;
    }
    while   (s1 <= t1}
    {  t [ q ] = r [ s1 ];    s1 = s1 + 1;    q = q + 1;   }
    while  ( s2 <= t2)
    {    t [ q ] = r [ s2 ];    s2 = s2 + 1;    q = q + 1   }
}
```



8.7 各种内部排序算法的比较

排序方法	最好时间	平均时间	最坏时间	辅助空间	稳定性
直接插入	O(n)	O(n2)	O(n2)	O(1)	
希尔		O(n1.3)		O(1)	
冒泡	O(n)	O(n2)	O(n2)	O(1)	
快速	O(nlog2n)	O(nlog2n)	O(n2)	O(log2n)	
简单选择	O(n2)	O(n2)	O(n2)	O(1)	
堆	O(nlog2n)	O(nlog2n)	O(nlog2n)	O(1)	

n较大

归并

- 按平均时间排序方法分为四类O(n2)、O(nlgn)、O(n1+□)、O(n)
- 快速排序是目前基于比较的内部排序中最好的方法
- 关键字随机分布时，快速排序的平均时间最短
- 当n较小时如(n<50)，可采用直接插入或简单选择排序，前者是稳定排序，但后者通常记录移动次数少于前者
- 当n较大时，应采用时间复杂度为O(nlgn)的排序方法(主要为快速排序和堆排序)
- 当n较大时，为避免顺序存储时大量移动记录的时间开销，可考虑用链表作为存储结构（如插入排序、归并排序）

- 文件初态基本按正序排列时，应选用直接插入、冒泡或随机的快速排序
- 选择排序方法应综合考虑各种因素

讨论：假设有 n 个值不同的元素存于顺序结构中，要求不经排序选出前k (k<<n) 个最小元素，问哪些方法可用，哪些方法比较次数最少?

- 选择排序或冒泡排序：k趟（数据比较次数约为kn次）；
- 快速排序：每次仅对第一个子序列划分，直至子序列长度小于等于k；长度不足k，则再对其后的子序列划分出补足的长度即可（平均对数据比较2n次）
- 堆排序：先建小根堆，k-1次堆调整（数据比较次数约为4n+(k-1)log2n）

内部排序作业

1.若待排序列用带头结点的单链表存储，试给出简单选择排序算法。

```
typedef   struct   node {
int     data;
node   *next;
} node,   *pointer;
void selectsort(pointer h) //h为头指针
```

2.请编写出算法，借助于快速排序的算法思想，在一组无序的记录中查找给定关键字值等于k的记录是记录序列中第几大的数。设此组记录存于数组r[low..high]中，若查找成功，则返回该记录是r数组中第几大的数，否则返回0。

```
typedef  struct  record{
keytype      key;
etype  otherinfo;
}RecType;
int Index (RecType  r[], int  low, int  high,  int   k)
```

3.以关键码序列(503, 087, 512, 061, 908, 170, 897, 275, 653, 426)为例，手工执行以下排序算法，写出每一趟排序结束时的关键码状态：

(1) 直接插入排序

(2)冒泡排序

(3) 简单选择排序

(4)快速排序(第一个记录为基准记录)

(5) 归并排序

作业->应用题