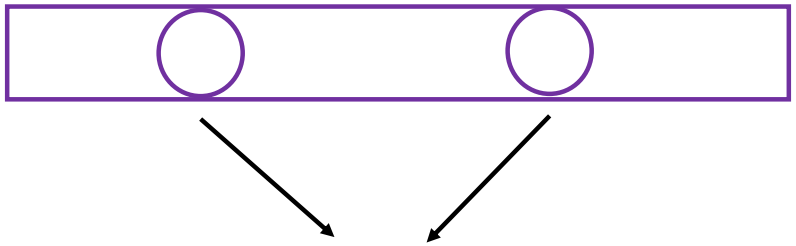
第10章 排序

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

10.3交换排序

基本思想：



两两比较，如果发生逆序则交换，直到所有记录都排好序为止。

常见的交换排序方法：

冒泡排序 $O(n^2)$

快速排序 $O(nlog_2n)$

10.3.1 冒泡排序

——基于简单交换思想

基本思想：每趟不断将记录两两比较，并按“前小后大”规则交换

例：6个记录	21	25	49	25*	16	08
第1趟结束：	21	25	25*	16	08	49
第2趟结束：	21	25	16	08	25*	49
第3趟结束：	21	16	08	25	25*	49
第4趟结束：	16	08	21	25	25*	49
第5趟结束：	08	16	21	25	25*	49

冒泡排序过程（升序）

初始：21，25，49，25*，16，08　n=6

第一趟

位置0，1进行比较 —— 判断 —— 不交换 —— 结果：	21	25	49	25*	16	08
位置1，2进行比较 —— 判断 —— 不交换 —— 结果：	21	25	49	25*	16	08
位置2，3进行比较 —— 判断 —— 交换 —— 结果：	21	25	25*	49	16	08
位置3，4进行比较 —— 判断 —— 交换 —— 结果：	21	25	25*	16	49	08
位置4，5进行比较 —— 判断 —— 交换 —— 结果：	21	25	25*	16	08	49

冒泡排序过程（升序）

第一趟结束后：21，25，25*，16，08，49

第二趟

位置0，1进行比较 —— 判断 —— 不交换 —— 结果：	21	25	25*	16	08	49
位置1，2进行比较 —— 判断 —— 不交换 —— 结果：	21	25	25*	16	08	49
位置2，3进行比较 —— 判断 —— 交换 —— 结果：	21	25	16	25*	08	49
位置3，4进行比较 —— 判断 —— 交换 —— 结果：	21	25	16	08	25*	49

冒泡排序过程（升序）

第2趟结束后：21，25，16，08，25*，49

继续下一趟，每一趟增加一个有序元素

第3趟结果：21，16，08，25，25*，49

第4趟结果：16，08，21，25，25*，49

第5趟结果：08，16，21，25，25*，49

总结：

n个记录，总共需要n-1趟

第m趟需要比较n-m次

冒泡排序算法


```
void bubble_sort (SqList &L){ //冒泡排序算法
    int m, i, j;     RedType x;       //交换时临时存储
    for ( m = 1; m <= n-1; m++) {//总共需要n-1趟
        for (j=1; j<=n-m; j++){
            if (L. r[j].key > L.r[j+1].key){  //发生逆序
                x = L.r[j]; L.r[j] = L.r[j+1]; L.r[j+1] = x;   //交换
            }
        }
}
```

10.3.1 冒泡排序

优点：每趟结束时，不仅能挤出一个最大值到最后面位置，还能同时部分理顺其他元素；

如何提高效率？

一旦某一趟比较时不出现记录交换，说明已排好序了，就可以结束本算法。

冒泡排序算法改进

例

49	38	38	38	13	13	13
38	49	49	13	27	27	27
97	76	13	27	30	30	30
76	13	27	30	38	38	38
13	27	30	49	49	49	49
27	30	76	76	76	76	76
30	97	97	97	97	97	97
	第一趟	第二趟	第三趟	第四趟	第五趟	第六趟

未发生交换，后面几趟可以省略

改进的冒泡排序算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

```
void bubble_sort (SqList &L){ //改进的冒泡排序算法
    int m, i, j, flag=1;     RedType x;       //flag作为是否有交换的标记
    for ( m = 1; m <= n-1 && flag==1; m++) {
        flag = 0;
        for (j=1; j<=n-m; j++){
            if (L. r[j].key > L.r[j+].key){  //发生逆序
                flag = 1;  //发生交换，flag置为1，若本趟没发生交换，flag保持为0
                x = L.r[j]; L.r[j] = L.r[j+1]; L.r[j+1] = x;  //交换
            }
        }
}
```

冒泡排序算法分析

时间复杂度

最好情况（正序）

比较次数：n-1

移动次数：0

最坏情况（逆序）

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

移动次数：$3\sum_{i=1}^{n-1}(n-i)=\frac{3}{2}(n^2-n)$

冒泡排序的算法评价

- 冒泡排序最好时间复杂度是O(n)
- 冒泡排序最坏时间复杂度为$O(n^2)$
- 冒泡排序平均时间复杂度为$O(n^2)$
- 冒泡排序算法中增加一个辅助空间temp,辅助空间为S(n)=O(1)
- 冒泡排序是稳定的

各种排序方法比较

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

类别	排序方法	时间复杂度			空间复杂度	稳定性
		最好情况	最坏情况	平均情况	辅助空间	
插入排序	直接插入排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
	希尔排序	O(n)	$O(n^2)$	$\sim O(n^{1.3})$	O(1)	不稳定
交换排序	冒泡排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定

10.3.2 快速排序

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

——— 改进的交换排序

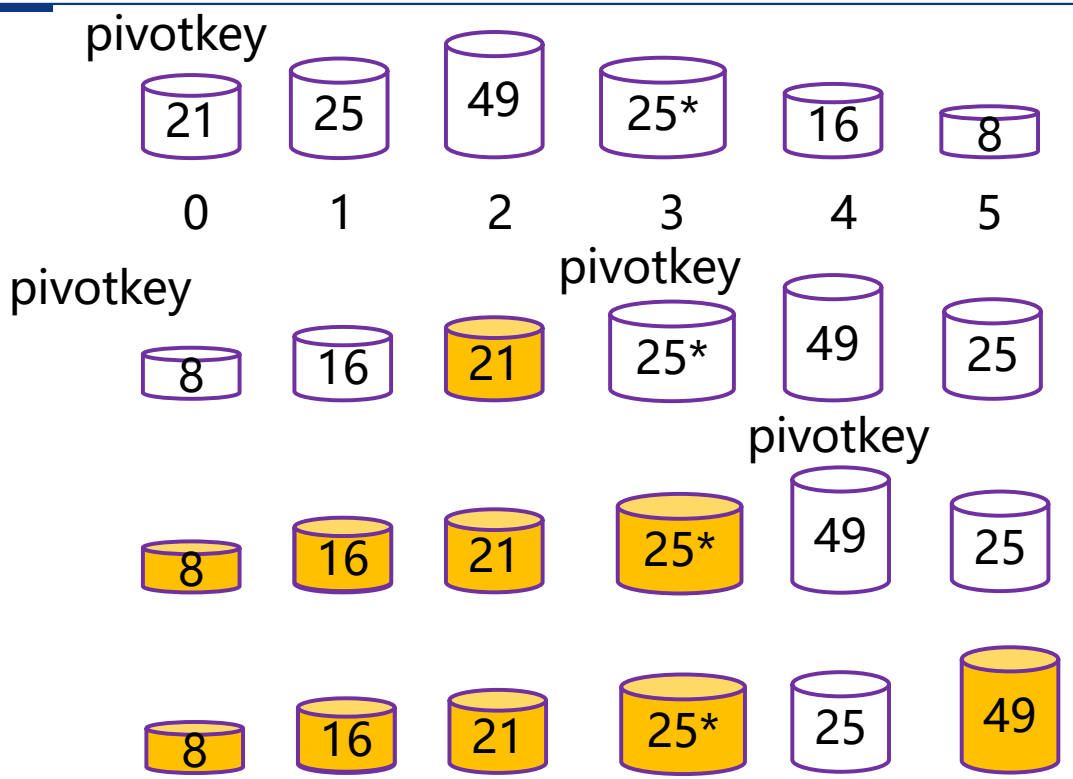
pivot:枢轴，中心点

基本思想：

- 任取一个元素（如：第一个）为中心
- 所有比它小的元素一律前放，比它大的元素一律后放，形成左右两个子表
- 对各子表重新选择中心元素并依此规则调整，直到每个子表的元素只剩一个

递归思想

快速排序演示


pivotkey
21
25
49
25*
16
8
0
1
2
3
4
5
pivotkey
pivotkey
8
16
21
25*
49
25
pivotkey
8
16
21
25*
49
25
8
16
21
25*
25
49

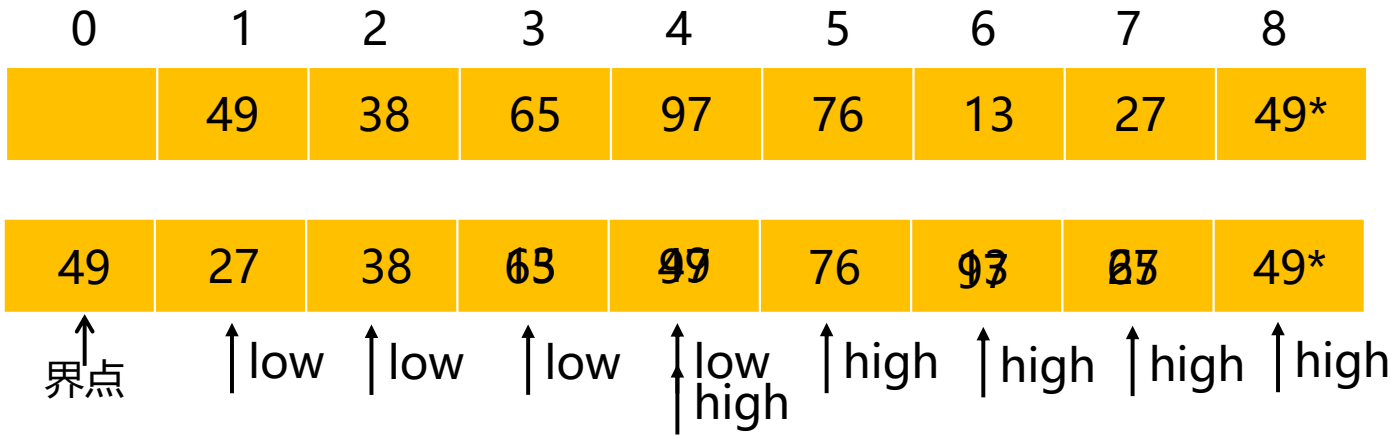
10.3.2 快速排序

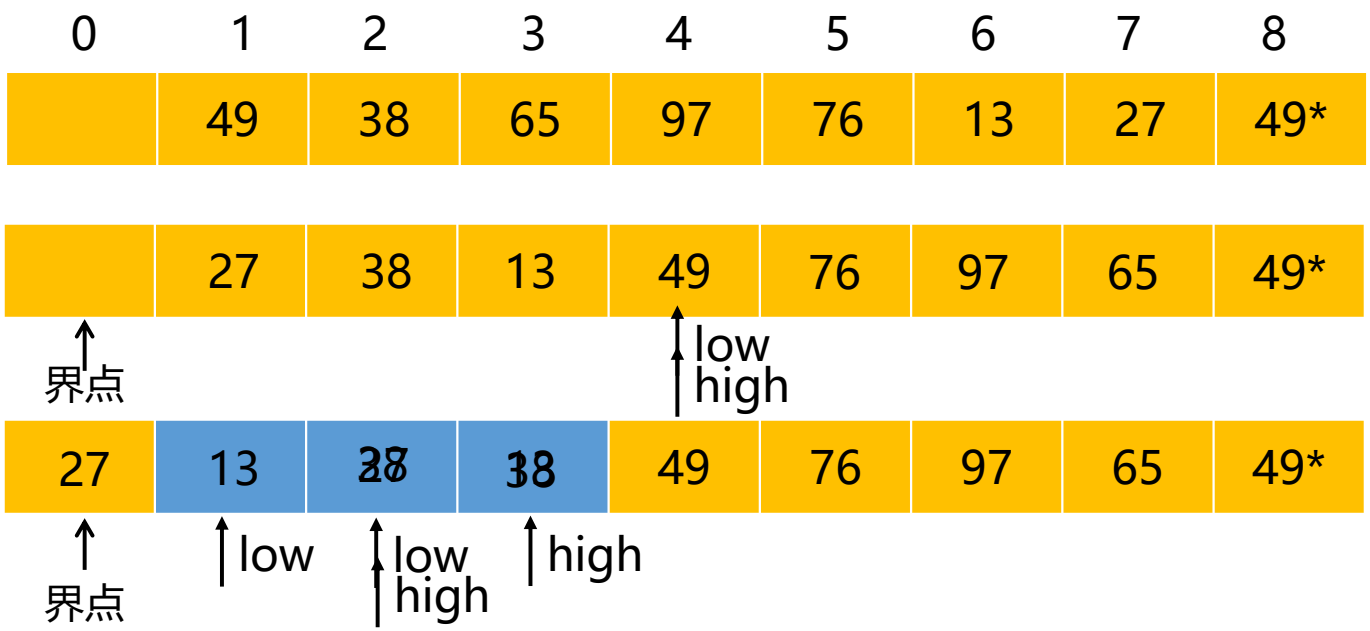
杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

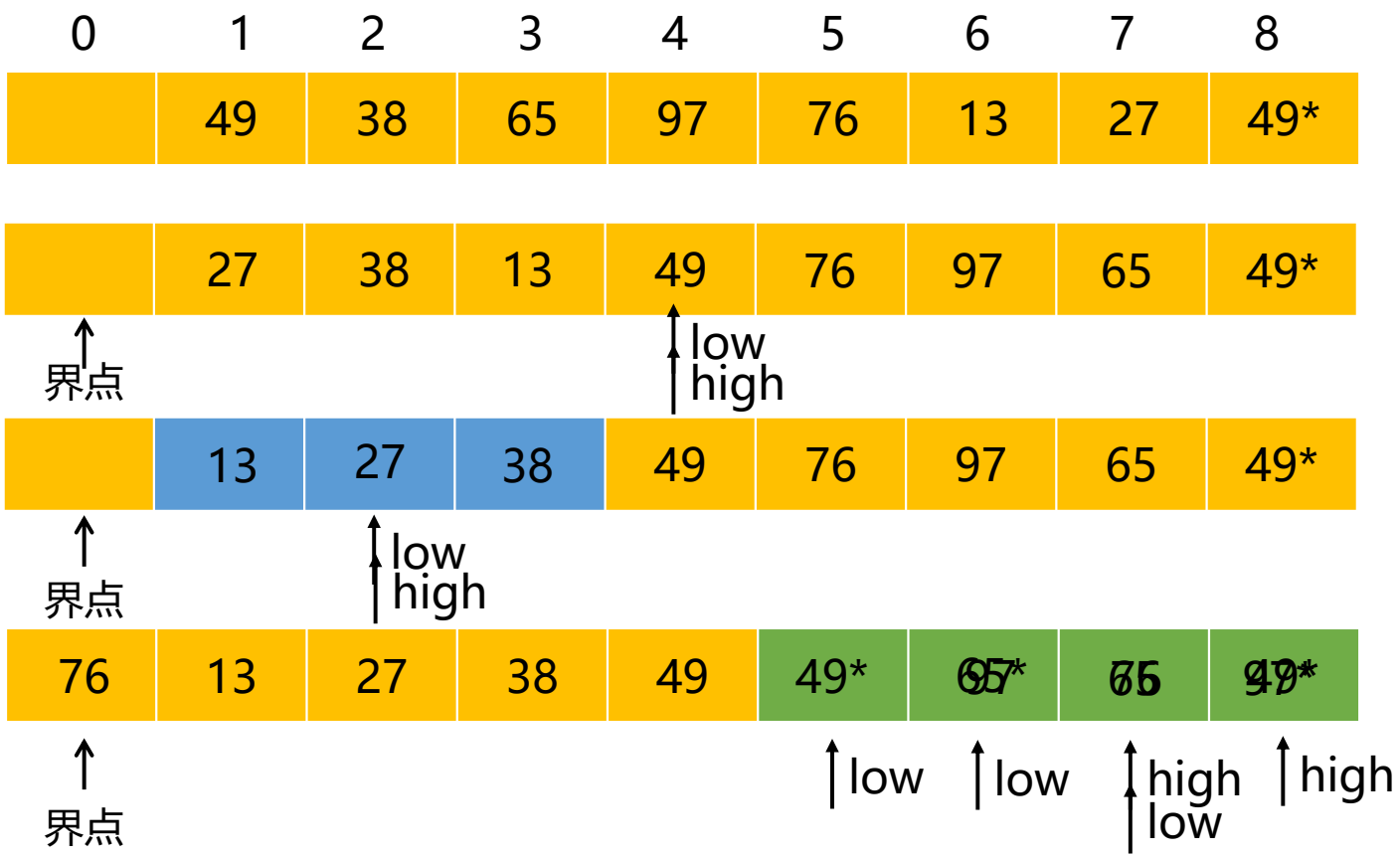
基本思想：通过一趟排序，将待排序记录分割成独立的两部分，其中一部分记录的关键字比另一部分记录的关键字小，则可分别对这两部分记录进行排序，以达到整个序列有序

具体实现：选定一个中间数作为参考，所有元素与之比较，小的调到其左边，大的调到其右边

（枢轴）中间数：可以是第一个数、最后一个数、最中间一个数、任选一个数等。


0
1
2
3
4
5
6
7
8
49
38
65
97
76
13
27
49*
49
27
38
49*
界点
low
low
low
low
high
high
high
high
high


0 1 2 3 4 5 6 7 8
49 38 65 97 76 13 27 49*
27 38 13 49 76 97 65 49*
界点
low
high
27 13 38 49 76 97 65 49*
界点
low
low
high
high


0 1 2 3 4 5 6 7 8
49 38 65 97 76 13 27 49*
27 38 13 49 76 97 65 49*
界点
low
high
13 27 38 49 76 97 65 49*
界点
low
high
76 13 27 38 49 49* 65* 65 49*
界点
low
low
high
low
high

10.3.2 快速排序

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

① 每一趟的子表的形成是采用从两头向中间交替式逼近法;

② 由于每趟中对各子表的操作都相似，可采用递归算法。

快速排序算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

```
void QuickSort (SqList &L){ //快速排序算法
    QSort (L, 1, L.length);
}
void QSort (SqList &L, int low, int high){
//对顺序表L中的子序列L.r[low..high]做快速排序
     if (low < high){                                  //长度大于1
         pivotloc = Partition(L, low, high);  //将L.r[low..high]一分为二
         QSort (L, low, pivotloc-1);      //对低子表递归排序，   pivotloc是枢轴位置
         QSort (L, pivotloc+1, high);   //对高子表递归排序
     }
}
```

快速排序算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

```
int Partition (SqList &L, int low, int high){
    L.r[0] = L.r[low]; pivotkey = L.r[low].key;
    while (low < high){                          //从表的两端交替地向中间扫描
        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;
}
```

快速排序算法分析

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

- 时间复杂度
  - 可以证明，平均时间是O(nlogn)
    - $T(n)=T_{pass}(n) + T(k-1) + T(n-k)$

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，在所有同数量级（O(nlogn)）的排序方法中，其平均性能最好。

快速排序算法分析

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

空间复杂度

快速排序不是原地排序

由于程序中使用了递归，需要递归调用栈的支持，而栈的长度取决于递归调用的深度。(即使不用递归，也需要用用户栈)

最好情况：每次左右都是均匀划分，空间复杂度为 O(logn)，

最坏情况：每次只能排除一个元素，要递归剩下n-1个元素，其空间复杂度为O(n)，

平均情况：空间复杂度也为O(logn)。

快速排序算法分析

稳定性

快速排序是一种不稳定的排序方法。

例如: 49, 38, 49*, 20, 97, 76

一次划分后: 20, 38, 49*, 49, 97, 76

思考

试对(90, 85, 79, 74, 68, 50, 46)进行快速排序的划分

- 你是否发现什么特殊情况?
- 再对(46, 50, 68, 74, 79, 85, 90)进行快速排序划分呢?

由于每次枢轴记录的关键字都是大于其它所有记录的关键字，致使一次划分之后得到的子序列(1)的长度为0，这时已经退化成为没有改进措施的冒泡排序。

- 快速排序不适于对原本有序或基本有序的记录序列进行排序。

快速排序算法分析

划分元素的选取是影响时间性能的关键

输入数据次序越乱，所选划分元素值的随机性越好，排序速度越快，快速排序不是自然排序方法。

改变划分元素的选取方法，至多只能改变算法平均情况的下的时间性能，无法改变最坏情况下的时间性能。即最坏情况下，快速排序的时间复杂性总是O(n^2)

各种排序方法比较

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

类别	排序方法	时间复杂度			空间复杂度	稳定性
		最好情况	最坏情况	平均情况	辅助空间	
插入排序	直接插入排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
	希尔排序	O(n)	$O(n^2)$	$\sim O(n^{1.3})$	O(1)	不稳定
交换排序	冒泡排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
	快速排序	O(nlogn)	$O(n^2)$	O(nlogn)	O(logn)	不稳定

第10章 排序

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

10.4.1 简单选择排序

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

基本思想：在待排序的数据中选出最大（小）的元素放在其最终的位置

基本操作:

1.首先通过n-1次关键字比较，从n个记录中找出关键字最小的记录，将它与第一个记录交换

2.再通过n-2次比较，从剩余的n-1个记录中找出关键字次小的记录，将它与第二个记录交换

3.重复上述操作，共进行n-1趟排序后，排序结束

10.4.1 简单选择排序

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

i=1

最小：08
交换21，08

i=2

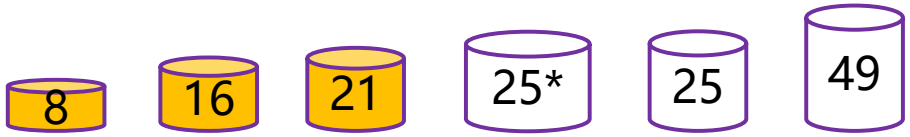
最小：16
交换25，16

i=3

最小：21
交换49，21

10.4.1 简单选择排序

i=4



最小：25*
无交换

i=5

最小：25
无交换

结果

10.4.1 简单选择排序算法

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

```
void SelectSort (SqList &L){

    for (i = 1; i < L.length; ++i ){
        k = i;
        for (j = i+1; j<= L.length; ++j)
                if (L.r[j].key < L.r[k].key)    k = j; //记录最小值位置
        if (k != i)
                {t = L.r[i]; L.r[i] = L.r[k]; L.r[k] = t;} //交换
     }
}
```

10.4.1 简单选择排序算法分析

时间复杂度

- 记录移动次数

  最好情况: 0

  最坏情况: 3(n-1)

- 比较次数:无论待排序列处于什么状态，选择排序所需进行的"比较"次数都相同 $\sum_{i=1}^{n-1}(n-i)=\frac{1}{2}(n^2-n)$

算法稳定性

- 简单选择排序是不稳定排序

例: 8 5 8* 7 9

第一次: 5 8 8* 7 9

第二次: 5 7 8* 8 9

各种排序方法比较

类别	排序方法	时间复杂度			空间复杂度	稳定性
		最好情况	最坏情况	平均情况	辅助空间	
插入排序	直接插入排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
	希尔排序	O(n)	$O(n^2)$	$\sim O(n^{1.3})$	O(1)	不稳定
交换排序	冒泡排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
	快速排序	O(nlogn)	$O(n^2)$	O(nlogn)	O(logn)	不稳定
选择排序	直接选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	不稳定

第10章 排序

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

堆排序

堆的定义：

若n个元素的序列{a1 a2 … an}满足

$$\begin{cases} a_i \le a_{2i} \\ a_i \le a_{2i+1} \end{cases} \quad \text{或} \quad \begin{cases} a_i \ge a_{2i} \\ a_i \ge a_{2i+1} \end{cases}$$

则分别称该序列{a1 a2 … an}为小根（顶）堆和大根（顶）堆

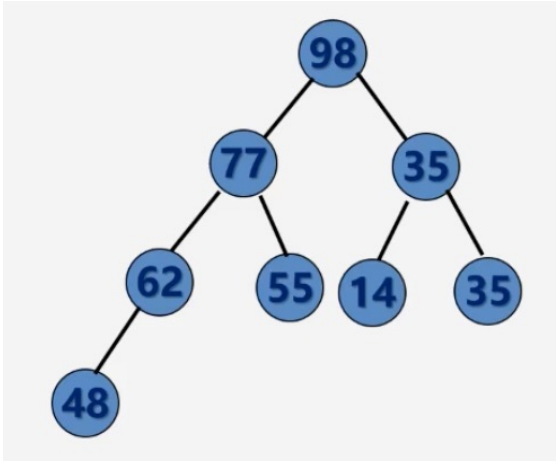
从堆的定义可以看出，堆实质是满足如下性质的完全二叉树：

二叉树中，任一非叶子结点均小于(大于)它的孩子结点

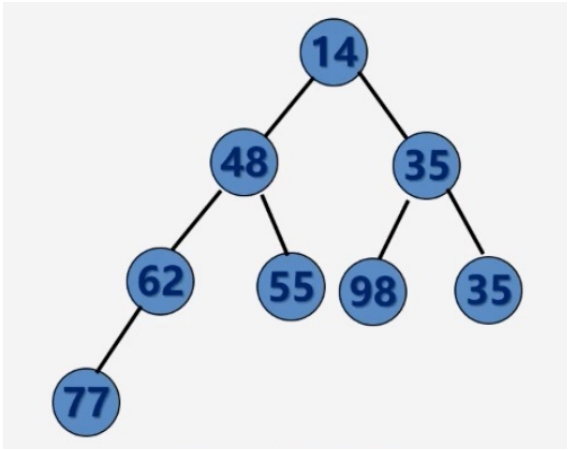
例：判断是否是堆?

{98 77 35 62 55 14 35 48}

{14 48 35 62 55 98 35 77}

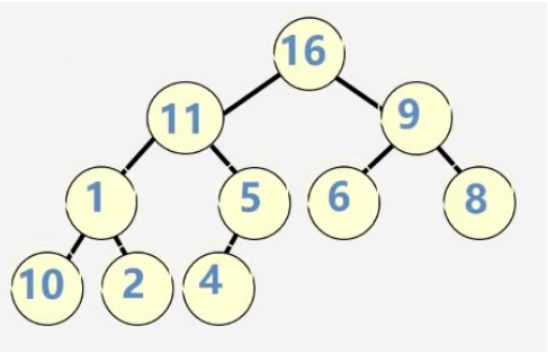


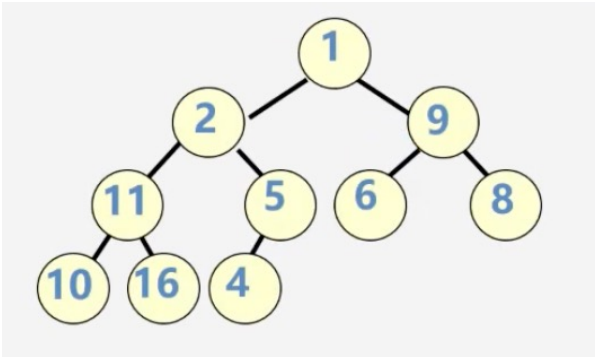
是一个大根堆



是一个小根堆

例：判断是否是堆?





堆排序

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

堆排序：

若在输出堆顶的最小值(最大值) 后，使得剩余n-1个元素的序列重又建成一一个堆， 则得到n个元素的次小值(次大值) ... 如此反复，便能得到一一个有序序列，这个过程称之为堆排序。

堆排序

实现堆排序需解决两个问题:

1、如何由一个无序序列建成一个堆?

2、如何在输出堆顶元素后，调整剩余元素为一个新的堆?

下面先讨论第二个问题:

堆的调整

如何在输出堆顶元素后，调整剩余元素为一个新的堆?

小根堆:

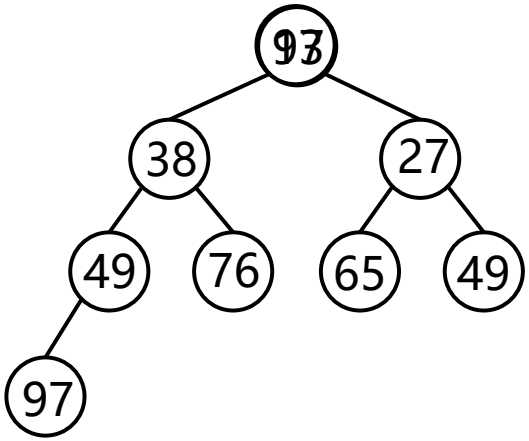
1.输出堆顶元素之后，以堆中最后一个元素替代之;

2.然后将根结点值与左、右子树的根结点值进行比较，并与其中小者进行交换;

3.重复上述操作，直至叶子结点，将得到新的堆，称这个从堆顶至叶子的调整过程为“筛选”。

那么大根堆的调整呢?

堆的调整



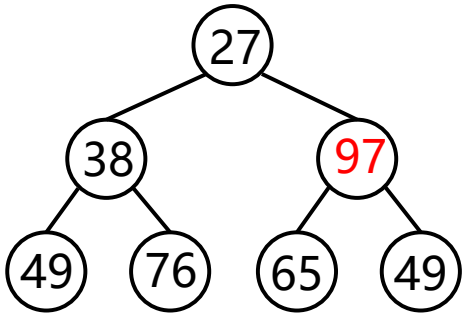
小根堆

13

输出根并以最后一个元素替代之：

比较其左右孩子值的大小，并与其中较小者交换

堆的调整



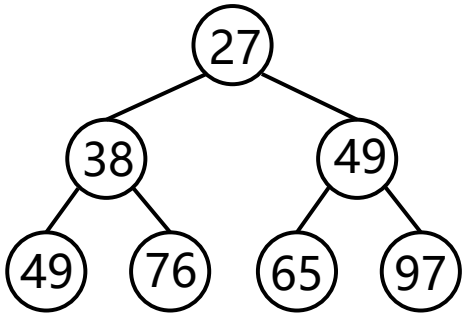
小根堆

输出根并以最后一个元素替代之：

比较其左右孩子值的大小，并与其中较小者交换

13

堆的调整



小根堆

输出根并以最后一个元素替代之：

比较其左右孩子值的大小，并与其中较小者交换

13

筛选过程的算法描述为：

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

```
void HeapAdjust (elem R[], int s, int m){
/* 已知R[s..m]中记录的关键字除R[s]之外均满足堆的定义，本函数调整R[s]
的关键字，使R[s..m]成为一个大根堆*/
    rc = R[s];
    for (j = 2*s; j <= m; j*=2 ){   //沿key较大的孩子节点向下筛选
        if (j<m && R[j]<R[j+1]).  ++j;    //j为key较大的记录的下标
        if (rc >= R[j])    break;
        R[s] = R[j];     s = j;  //rc应插入在位置s上
     }
     R[s] = rc;  //插入
}
```

堆的调整

可以看出:

对一个无序序列反复“筛选”就可以得到一个堆;

即：从一个无序序列建堆的过程就是一个反复“筛选”的过程。

那么：如何由一个无序序列建成一个堆?

堆的建立

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

显然；

单结点的二叉树是堆;

在完全二叉树中所有以叶子结点(序号i > n/2)为根的子树是堆。

这样，我们只需依次将以序号为n/2，n/2-1, … 1的结点为根的子树均调整为堆即可。

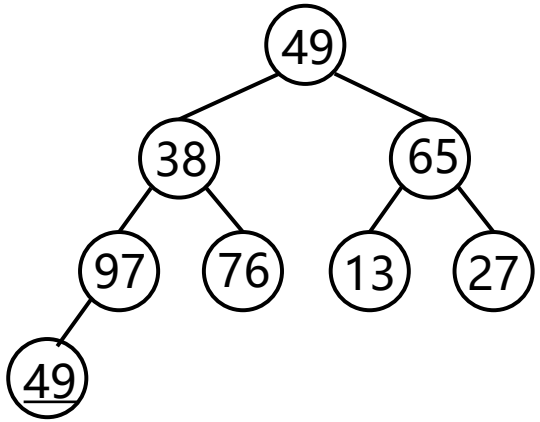
即：对应由n个元素组成的无序序列，“筛选” 只需从第n/2个元素开始。

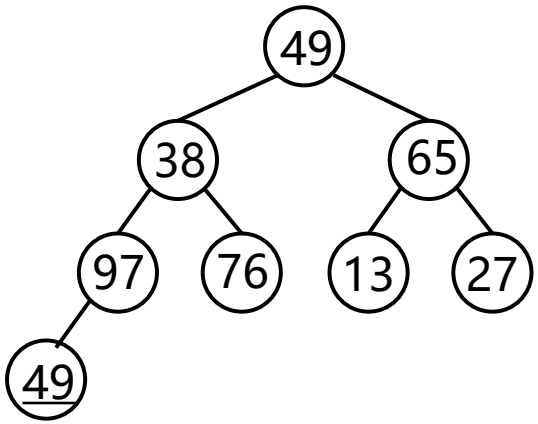
堆的建立

由于堆实质上是一个线形表，那么我们可以顺序存储一个堆。

下面以一个实例介绍建一个小根堆的过程。

例：有关键字为49，38，65，97，76，13，27，49的一组记录，将其按关键字调整为一个小根堆。

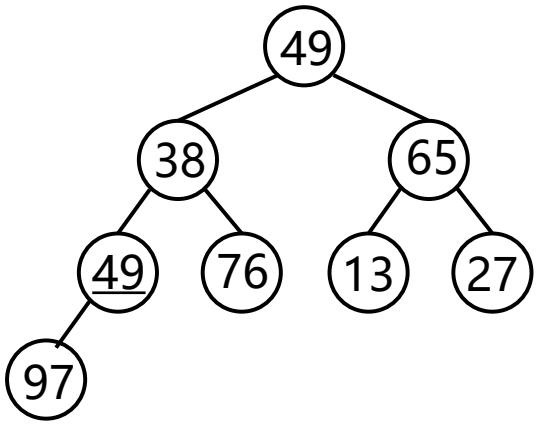




0	1	2	3	4	5	6	7	8
	49	38	65	97	76	13	27	49

从最后一个非叶子结点开始，以此向前调整：

①调整从第n/2个元素开始，将以该元素为根的二叉树调整为堆

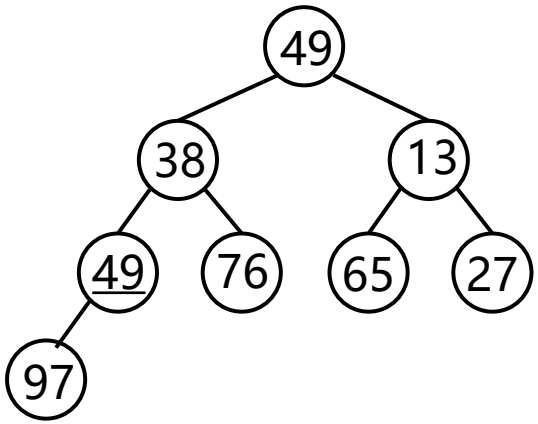


0	1	2	3	4	5	6	7	8
	49	38	65	49	76	13	27	97

从最后一个非叶子结点开始，以此向前调整：

①调整从第n/2个元素开始，将以该元素为根的二叉树调整为堆

②将以序号为n/2-1的结点为根的二叉树调整为堆;



0	1	2	3	4	5	6	7	8
	49	38	13	49	76	65	27	97

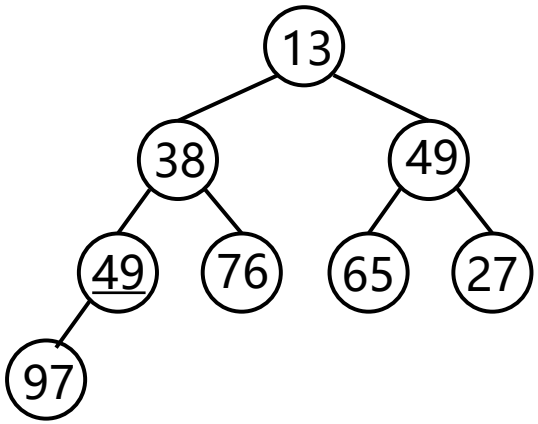
从最后一个非叶子结点开始，以此向前调整：

①调整从第n/2个元素开始，将以该元素为根的二叉树调整为堆

②将以序号为n/2-1的结点为根的二叉树调整为堆;

③再将以序号为n/2-2的结点为根的二叉树调整为堆;

④再将以序号为n/2-3的结点为根的二叉树调整为堆;



0	1	2	3	4	5	6	7	8
	13	38	49	49	76	65	27	97

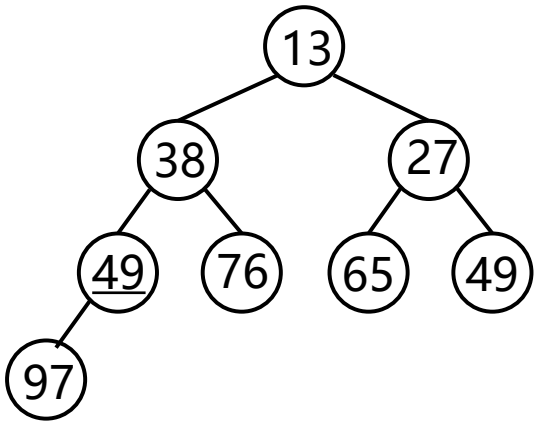
从最后一个非叶子结点开始，以此向前调整：

①调整从第n/2个元素开始，将以该元素为根的二叉树调整为堆

②将以序号为n/2-1的结点为根的二叉树调整为堆;

③再将以序号为n/2-2的结点为根的二叉树调整为堆;

④再将以序号为n/2-3的结点为根的二叉树调整为堆;



0	1	2	3	4	5	6	7	8
	13	38	27	49	76	65	49	97

从最后一个非叶子结点开始，以此向前调整：

①调整从第n/2个元素开始，将以该元素为根的二叉树调整为堆

②将以序号为n/2-1的结点为根的二叉树调整为堆;

③再将以序号为n/2-2的结点为根的二叉树调整为堆;

④再将以序号为n/2-3的结点为根的二叉树调整为堆;

堆的建立

将初始无序的R[1]到R[n]建成一个小根堆，可用以下语句实现：

```
for (i=n/2; i>=1; i--)
        HeapAdjust(R,i,n);
```

堆排序

由以上分析知:

若对一个无序序列建堆，然后输出根；重复该过程就可以由一个无序序列输出有序序列。

实质上，堆排序就是利用完全二叉树中双亲结点与孩子结点之间的内在关系来排序的。

堆排序

堆排序算法如下：

```
void HeapSort (elem R[]){
/* 对R[1]到R[n]进行堆排序*/
    int i;
    for (i = n/2; i >= 1; i-- )
          HeapAdjust(R, i, n);          //建初始堆；
    for (i = n; i >1; i-- ){.   //进行n-1趟排序
         Swap(R[1], R[i]);    //根与最后一个元素交换
         HeapAdjust(R, 1, i-1);   //将R[1]到R[i-1]重新调整为堆
     }
}
```

算法性能分析

- 初始堆化所需时间不超过O(n)

- 排序阶段(不含初始化堆)
  - 一次重新堆化所需时间不超过O(logn)
  - n-1次循环所需时间不超过O(nlogn)

Tw(n)=O(n)+ O(nlogn)= O(nlogn)

算法性能分析

- 堆排序的时间主要耗费在建初始堆和调整建新堆时进行的反复筛选上。堆排序在最坏情况下，其时间复杂度也为O($nlog_2n$)，这是堆排序的最大优点。无论待排序列中的记录是正序还是逆序排列，都不会使堆排序处于"最好"或"最坏"的状态。
- 另外，堆排序仅需一个记录大小供交换用的辅助存储空间。
- 然而，堆排序是一种不稳定的排序方法，它不适用于待排序记录个数n较少的情况，但对于n较大的文件还是很有效的。

各种排序方法比较

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

类别	排序方法	时间复杂度			空间复杂度	稳定性
		最好情况	最坏情况	平均情况	辅助空间	
插入排序	直接插入排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
	希尔排序	O(n)	$O(n^2)$	$\sim O(n^{1.3})$	O(1)	不稳定
交换排序	冒泡排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
	快速排序	O(nlogn)	$O(n^2)$	O(nlogn)	O(logn)	不稳定
选择排序	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

第10章 排序

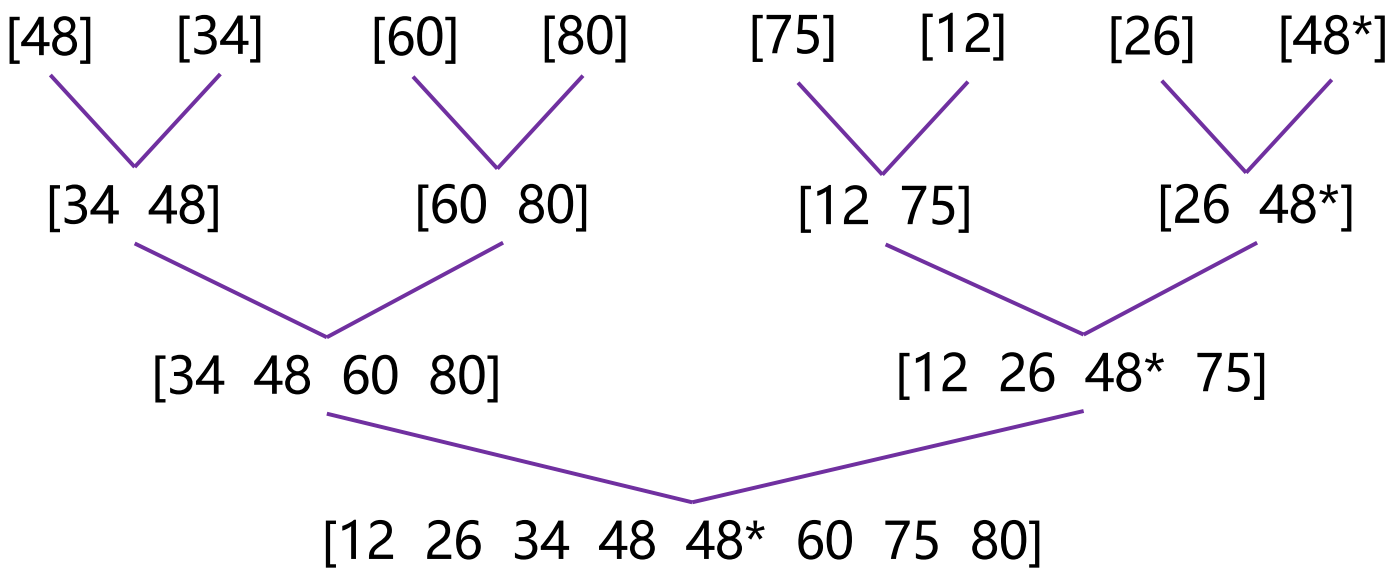
杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

10.5 归并排序

- 基本思想：将两个或两个以上的有序子序列"归并”为一个有序序列。
- 在内部排序中，通常采用的是2-路归并排序。
  - 即：将两个位置相邻的有序子序列R[l...m]和R[m+1...n]归并为一个有序序列R[l...n]

归并排序示例

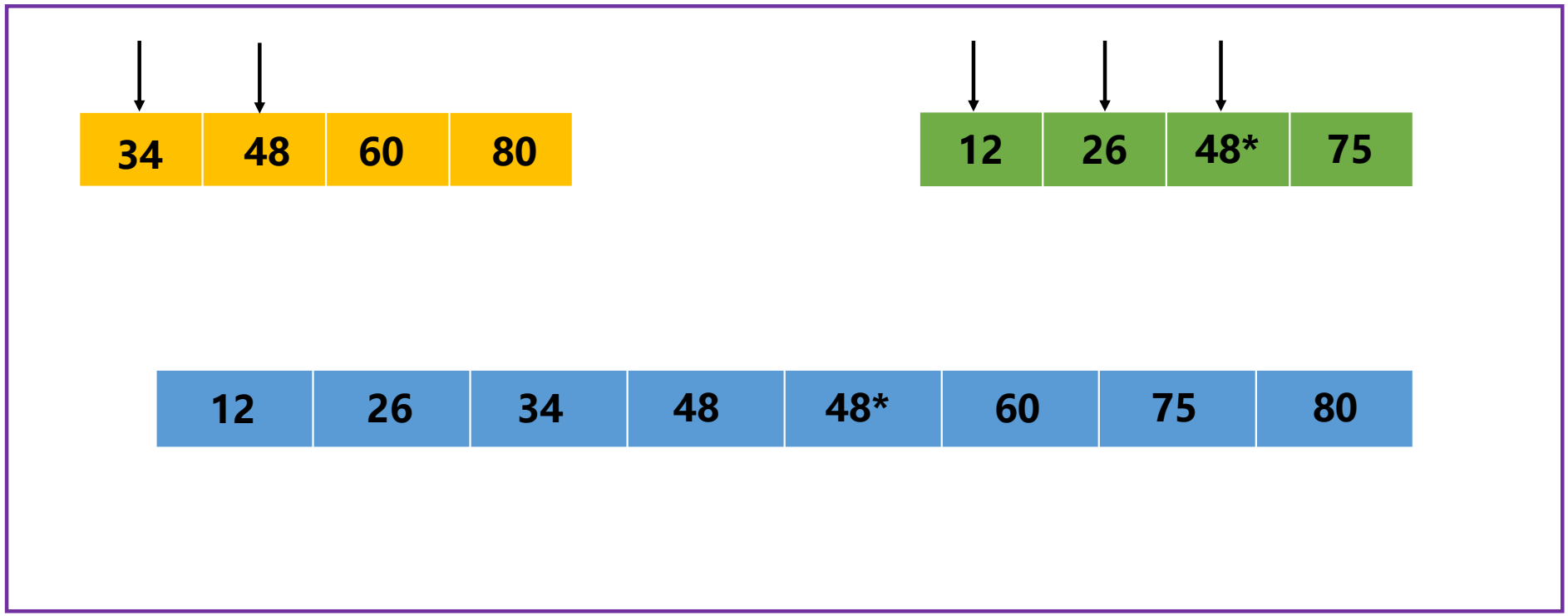
设初始关键字序列为：[48 34 60 80 75 12 26 48*]



整个归并排序仅需 $\lceil \log_2 n \rceil$ 趟

归并排序

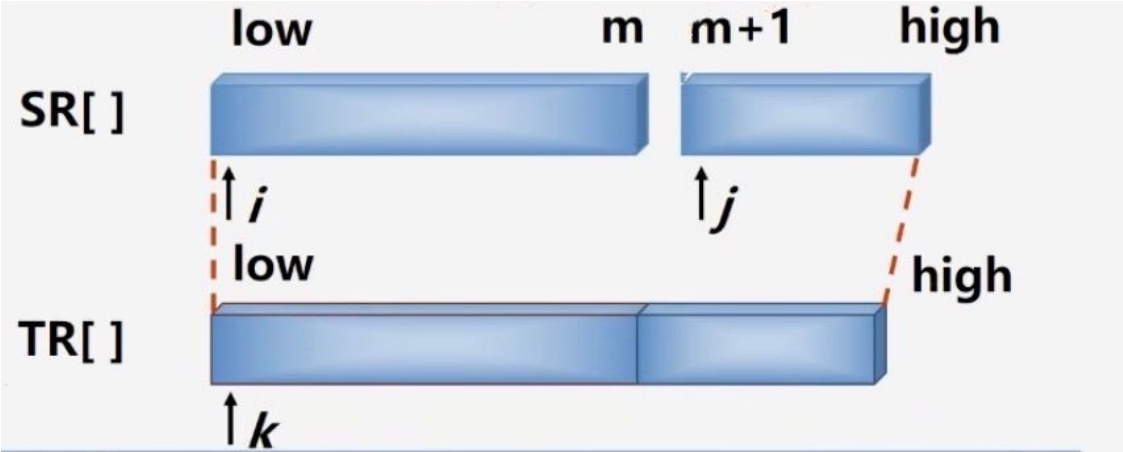
关键问题：如何将两个有序序列合成一个有序序列？



归并排序

设R[low]-R[mid]和R[mid+1]-R[high]为相邻,

归并成一个有序序列R1[low]-R1[high]。



若SR[i].key<=SR[j].key，则TR[k]=RS[i]; k++; i++;

否则，TR[k]=SR[j]; k++; j++;

归并排序算法分析

- 时间效率: $O(nlog_2n)$
- 空间效率: O(n)
  - 因为需要一个与原始序列同样大小的辅助序列(R1) 。这正是此算法的缺点。
- 稳定性：稳定

各种排序方法比较

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

类别	排序方法	时间复杂度			空间复杂度	稳定性
		最好情况	最坏情况	平均情况	辅助空间	
插入排序	直接插入排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
	希尔排序	O(n)	$O(n^2)$	$\sim O(n^{1.3})$	O(1)	不稳定
交换排序	冒泡排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
	快速排序	O(nlogn)	$O(n^2)$	O(nlogn)	O(logn)	不稳定
选择排序	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
归并排序		O(nlogn)	O(nlogn)	O(nlogn)	O(n)	稳定

第10章 排序

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

10.6 基数排序

基本思想：分配+收集

也叫桶排序或箱排序：设置若干个箱子，将关键字为k的记录放入第k个箱子，然后再按序号将非空的连接。

基数排序：数字是有范围的，均由0-9这十个数字组成，则只需设置十个箱子，相继按个、十、百...进行排序。

例：**(614，738，921，485，637，101，215，530，790，306)**

614	738	921	485	637	101	215	530	790	306

第一趟分配（按个位排）

e[0]	e[1]	e[2]	e[3]	e[4]	e[5]	e[6]	e[7]	e[8]	e[9]
530	921			614	485	306	637	738	
790	101				215				

第一趟收集

530	790	921	101	614	485	215	306	637	738

第一趟收集的结果：

530	790	921	101	614	485	215	306	637	738

第二趟分配（按十位排）

e[0]	e[1]	e[2]	e[3]	e[4]	e[5]	e[6]	e[7]	e[8]	e[9]
101	614	921	530					485	790
306	215		637						
			738						

第二趟收集

101	306	614	215	921	530	637	738	485	790

第二趟收集的结果：

101	306	614	215	921	530	637	738	485	790

第三趟分配（按百位排）

e[0]	e[1]	e[2]	e[3]	e[4]	e[5]	e[6]	e[7]	e[8]	e[9]
	101	215	306	485	530	614 637	738 790		921

排序结束！

第三趟收集

101	215	306	485	530	614	637	738	790	921

10.6 基数排序算法分析

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

时间效率: O(k*(n+m))

k：关键字个数，

m：关键字取值范围为m个值

空间效率：O(n+m)

稳定性：稳定

基数排序算法分析

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

例如: 10000个人按照生日排序

年 月 日 3个关键字

假设取值范围分别是: 1930～2018 1～12 1～31

O(n^2)≈10^8

O(nlogn)≈10^5

O(k*(n+m))≈(10000+89)+ (10000+12)+ (10000+31)

≈10^4

各种排序方法比较

类别	排序方法	时间复杂度			空间复杂度	稳定性
		最好情况	最坏情况	平均情况	辅助空间	
插入排序	直接插入排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
	希尔排序	O(n)	$O(n^2)$	$\sim O(n^{1.3})$	O(1)	不稳定
交换排序	冒泡排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定
	快速排序	O(nlogn)	$O(n^2)$	O(nlogn)	O(logn)	不稳定
选择排序	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
归并排序		O(nlogn)	O(nlogn)	O(nlogn)	O(n)	稳定
基数排序 k:待排元素的维数，m:基数的个数		O(n+m)	O(k*(n+m)	O(k*(n+m)	O(n+m)	稳定

第10章 排序

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

各种排序方法的综合比较

一、时间性能

1.按平均的时间性能来分，有三类排序方法:

- 时间复杂度为O(nlogn)的方法有:
  - 快速排序、堆排序和归并排序，其中以快速排序为最好;
- 时间复杂度为O(n^2)的有:
  - 直接插入排序、冒泡排序和简单选择排序，其中以直接插入为最好，特别是对那些对关键字近似有序的记录序列尤为如此;
- 时间复杂度为O(n)的排序方法只有：基数排序。

各种排序方法的综合比较

2.当待排记录序列按关键字顺序有序时，直接插入排序和冒泡排序能达到O(n)的时间复杂度；而对于快速排序而言，这是最不好的情况，此时的时间性能退化为O(n^2)，因此是应该尽量避免的情况。

3.简单选择排序、堆排序和归并排序的时间性能不随记录序列中关键字的分布而改变。

各种排序方法的综合比较

二、空间性能

指的是排序过程中所需的辅助空间大小

1.所有的简单排序方法(包括：直接插入、冒泡和简单选择)和堆排序的空间复杂度为O(1)

2.快速排序为O(logn)，为栈所需的辅助空间

3.归并排序空间复杂度为O(n)

4.基数排序空间复杂度为O(n+m)

链式基数排序稍微好一点，需附设队列首尾指针，则空间复杂度为O(rd)

各种排序方法的综合比较

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

三、排序方法的稳定性能

- 稳定的排序方法指的是，对于两个关键字相等的记录，它们在序列中的相对位置，在排序之前和经过排序之后，没有改变。
- 当对多关键字的记录序列进行LSD方法排序时，必须采用稳定的排序方法。
- 对于不稳定的排序方法，只要能举出一个实例说明即可。
- 快速排序、堆排序、简单选择排序是不稳定的排序方法。

各种排序方法的综合比较

四、关于“排序方法的时间复杂度的下限”

- 本章讨论的各种排序方法，除基数排序外，其它方法都是基于“比较关键字”进行排序的排序方法，可以证明，这类排序法可能达到的最快的时间复杂度为O(nlogn)。

  (基数排序不是基于“比较关键字”的排序方法，所以它不受这个限制)。
- 可以用一棵判定树来描述这类基于“比较关键字”进行排序的排序方法。

各种排序方法的综合比较

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

五、关于内部排序和外部排序的关系

- 外部排序基本上由两个相对独立的阶段组成：
  - 首先，按可用内存大小，将外存上含n个记录的文件分成若干长度为l的子文件或段，依次读入内存并利用有效的**内部排序方法**对它们进行排序，并将排序后得到的有序子文件重新写入外存，通常称这些有序子文件为归并段或顺串；（划分归并段的过程）
  - 然后，对这些归并段进行逐趟归并，使归并段（有序的子文件）逐渐由小至大，直至得到整个有序文件为止。（归并排序的过程）