# 排序概念

* **排序：**是计算机程序设计中的一种重要操作，它的功能是将一个数据元素（或记录）的任意序列，重新排列成一个**按关键字有序**的序列。
  + - 注意：我们排序的**是记录**而非关键字，**关键字只是排序的依据**。但是我们为了讲课的方便，举例时可能只涉及到仅包含关键字的记录。
* **内部排序:** 指的是**待排序记录存放在计算机随机存储器中进行的排序过程。**而外部排序指的是当排序记录的数量很大，以致内存一次不能容纳全部记录，在排序过程中尚需对**外存**进行访问的排序过程。
* **排序的稳定性：**如果两记录Ri与Rj的关键字相同，在排序前Ri在Rj的前面，排序之后， Ri依然在Rj之前，我们称这种排序方法是**稳定的**，反之，是不稳定的。

## 排序表定义

#define MAXSIZE 20

typedef int KeyType;

typedef struct

{

KeyType key; // 关键字

InfoType otherinfo; // 其他信息

}RedType;

typedef struct

{

RedType r[MAXSIZE+1]; // r[0]闲置或用作哨兵单位

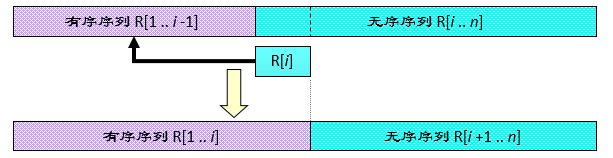
int length; // 记录长度

}SqList;

# 插入排序

## 直接插入排序

一趟直接插入排序的基本思想：

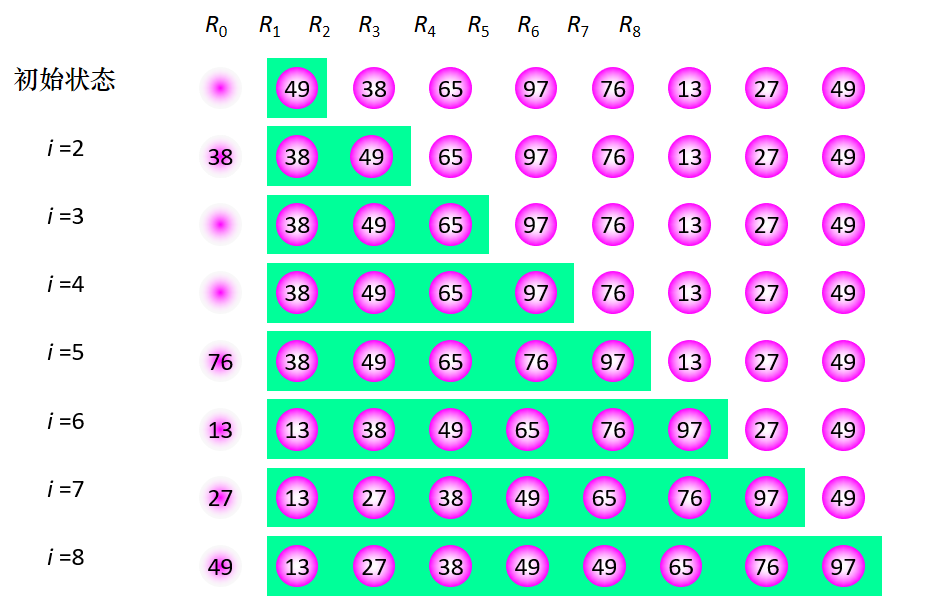


实现“一趟插入排序”可分三步进行：

1．在R[1 .. *i* -1] 中查找 R[*i*] 的插入位置，R[1 .. *j*].key ≤ R[*i*].key < R[ *j*+1 .. *i* -1].key；

2．将 R[ *j*+1 .. *i* -1] 中的所有记录均后移一个位置；

3．将 R[i] 插入（复制）到 R[ j+1] 的位置上。



注意细节：在65和97进行直接插入排序的时候，**没有和r[0]进行交换**

**排序过程：**先将序列中第 1 个记录看成是一个有序子序列，然后从第 2 个记录开始，逐个进行插入，直至整个序列有序。

void InsertSort(SqList &L)

{

// 对顺序表 L 作直接插入排序。

for (i = 2;i <= L.length;++ i)

{

if (L.r[i].key < L.r[i-1].key)

{

L.r[0] = L.r[i]; // 复制为监视哨

L.r[i] = L.r[i -1];

for (j = i - 2;L.r[0].key < L.r[j].key;--j)

L.r[j+1] = L.r[j]; // 记录后移

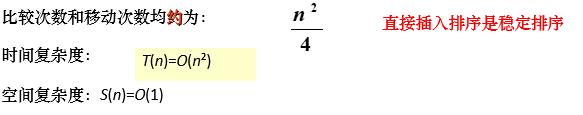
L.r[j+1] = L.r[0]; // 插入到正确位置

}

}

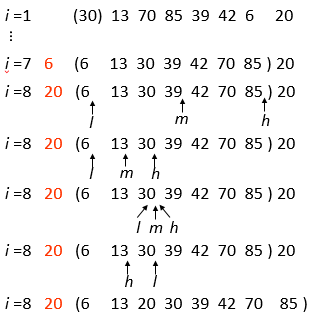
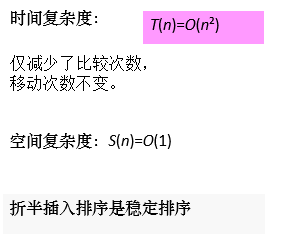
} // InsertSort

**算法分析：**



## 折半插入排序

用折半查找方法确定插入位置的排序。



void BInsertSort ( SqList &L ) {

for ( i = 2; i <= L.length; ++ i ) {

L.r[0] = L.r[i];

low = 1; high = i -1;

while (low <= high) {

m = (low+high)/2; // 折半

if (L.r[0].key < L.r[m].key)

high = m -1;

else low = m +1;

} //while

for ( j = i -1; j >= high +1; --j )

L.r[j +1] = L.r[j]; // 记录后移

L.r[high+1] = L.r[0]; // 插入

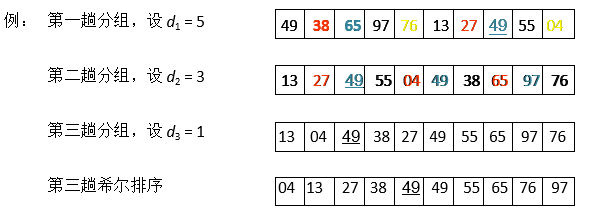
} //for

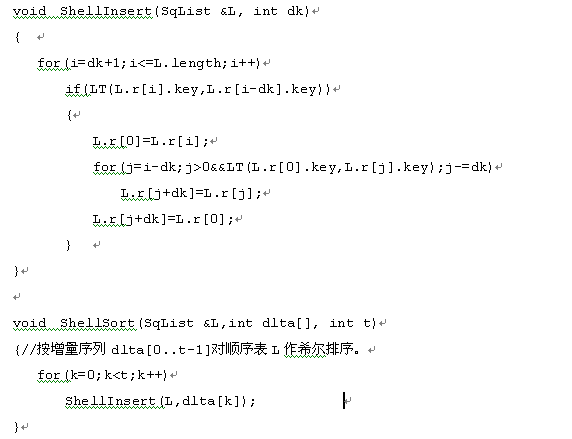
} // BinsertSort

## 希尔排序（缩小增量排序）

基本思想：对待排序列先作“宏观”调整，再作“微观”调整。

排序过程：先取一个正整数 *d*1 < *n*，把所有相隔 *d*1 的记录放在一组内，组内进行直接插入排序；然后取 *d*2 < *d*1，重复上述分组和排序操作；**直至 *di* = 1**，即所有记录放进一个组中排序为止。其中 ***di* 称为增量**（一般取值5 3 1）。



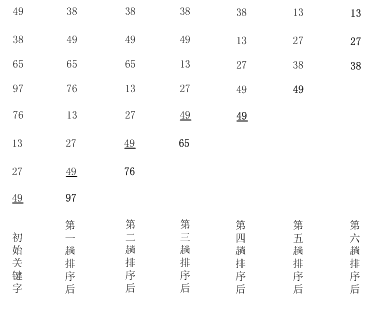


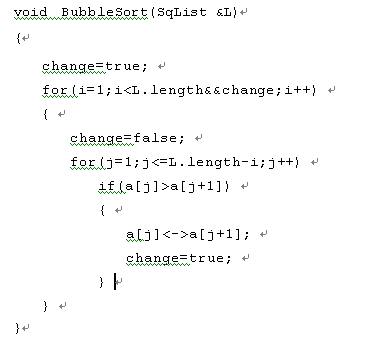
**希尔排序的时间复杂度约为：O(n1.3)**

**空间复杂度S(n) = O(1) 希尔排序是非稳定性排序**

# 交换排序

## 起泡排序





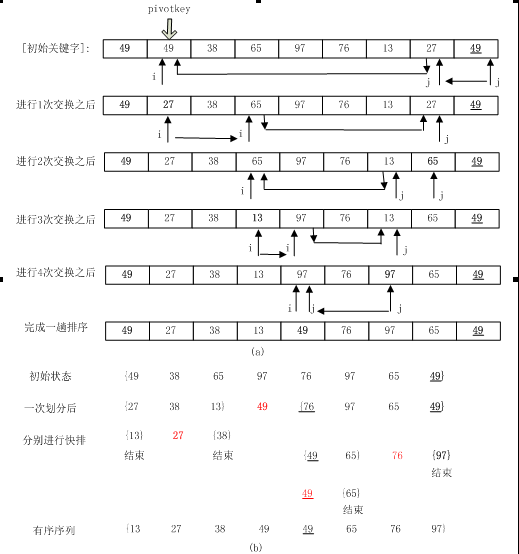
**起泡排序的时间复杂度约为：O(n2)**

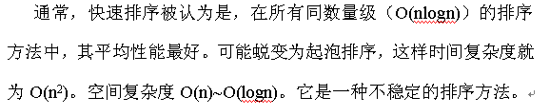
**空间复杂度S(n) = O(1) 起泡排序是稳定性排序**

## 快速排序

**基本思想：任选一个记录**，**（一般取第一个记录）**以它的关键字作为“枢轴”，凡关键字小于枢轴的记录均移至枢轴之前，凡关键字大于枢轴的记录均移至枢轴之后(枢轴要放在合适的位置上)。

附设两个指针 low 和 high，从 high 所指位置起**向前搜索找到第一个关键字小于枢轴的关键字的记录与枢轴记录交换**，然后**从 low所指位置起向后搜索找到第一个关键字大于枢轴的关键字的记录与枢轴记录交换**，重复这两步直至 low = high 为止。

****

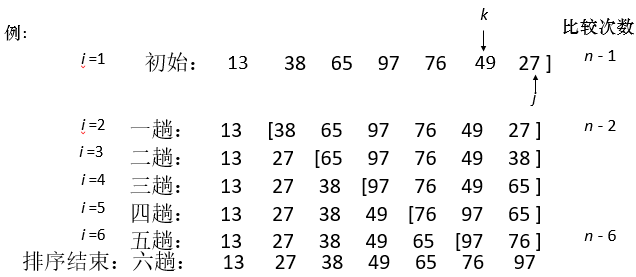


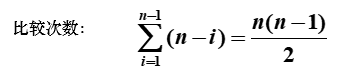
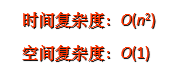
# 选择排序

## 简单选择排序

排序过程：

* 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。
* 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。
* 重复上述操作，共进行 *n* –1 趟排序后，排序结束。



移动次数： 正序：最小值为 0； 最大值为 3(*n*-1) 。

**最大值的情况：前 *n* – 1 个为正序，第 *n* 个记录的关键字最小。**

简单选择排序是一种**不稳定**的排序方式

## 堆排序

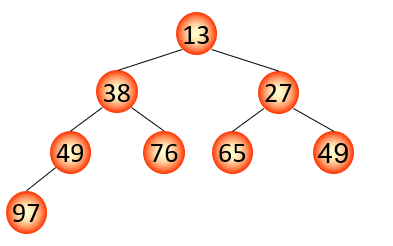
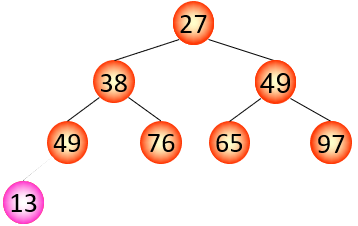
**堆排序：将无序序列建成一个堆**，得到关键字最小（大）的记录；输出堆顶的最小（大）值后，将剩余的 *n*-1 个元素重又建成一个堆，则可得到 *n* 个元素的次小值；如此重复执行，直到堆中只有一个记录为止，每个记录出堆的顺序就是一个有序序列，这个过程叫堆排序。

**堆排序需解决的两个问题：**

1、如何由一个无序序列建成一个堆？

2、在输出堆顶元素后，如何将剩余元素调整为一个新的堆？

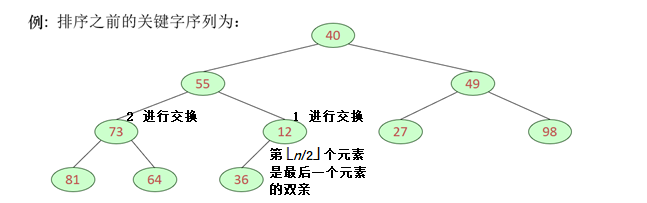
**输出堆顶元素之后，以堆中最后一个元素替代之**；**然后将根结点值与左、右子树的根结点值进行比较，并与其中小者进行交换；**重复上述操作，直至叶子结点，将得到新的堆，称这个从堆顶至叶子的调整过程为“筛选”。

对深度为 ***k*** 的堆，“筛选”所需进行的关键字比较的次数至多为 **2(*k*-1)**。

**堆排序—建堆**

从无序序列的第 **⎣*n*/2⎦** 个元素（即无序序列对应的完全二叉树的最后一个内部结点）起，至第一个元素止，进行反复筛选。



效率分析

因此，**堆排序的时间复杂度为*O*(*n*log*n*)**，与简单选择排序 *O*(*n*2) 相比时间效率提高了很多。

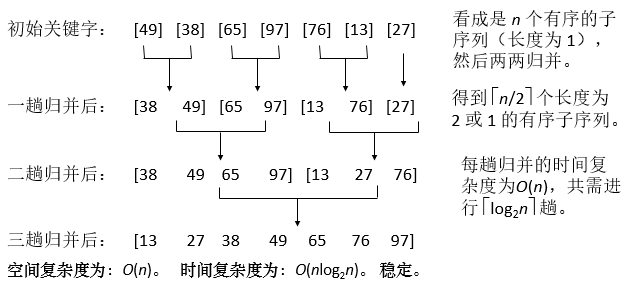
**空间复杂度：S(n) = O(1)**

堆排序是一种**速度快**且**省空间**的排序方法。是一种**不稳定**的排序方式

# 归并排序

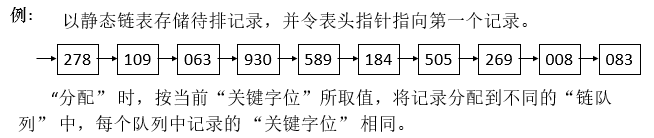
归并：将两个或两个以上的有序表组合成一个新的有序表。

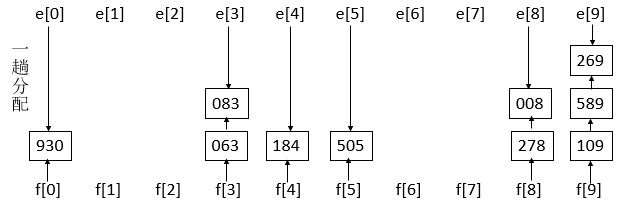
在内部排序中，通常采用的是 **2-路归并排序**。即：将两个位置相邻的记录有序子序列归并为一个记录有序的序列。

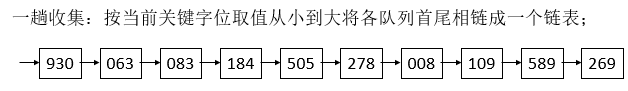


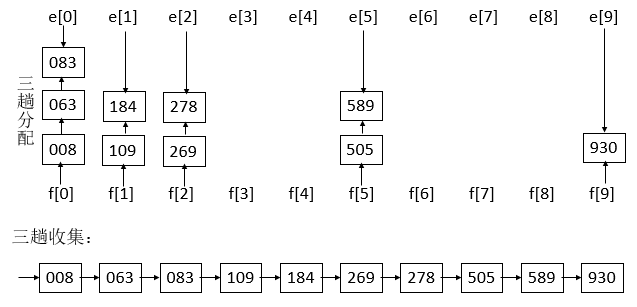
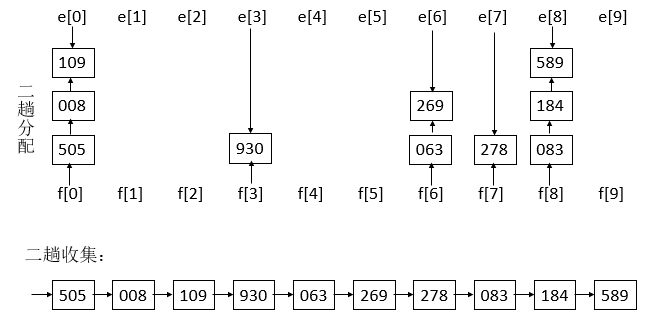
# 基数排序

* 最高位优先法（MSD）必须将序列逐层分割成若干子序列， 然后对各子序列分别排序。
* 最低位优先法（LSD）不必分成若干子序列，只要按照低位优先排就行 。(排序方法有限制)









***T*(*n*)=*O*(*d*(*n*+*rd*))**

**空间复杂度：*S*(*n*)=2*rd* 个队列指针 + *n* 个指针域空间**

**基数排序是一种稳定的排序方式**

# 各种排序方法比较



## 排序总结

**排序的时间复杂度，空间复杂度，稳定性**

|  |  |  |  |
| --- | --- | --- | --- |
| 排序方法 | 时间复杂度 | 空间复杂度 | 稳定性 |
| 直接插入排序 | **O（n2）** | **O（1）** | **稳定** |
| 折半插入排序 | **O（n2）** | **O（1）** | **稳定** |
| 希尔排序 | **O（n1.3）** | **O（1）** | **不稳定** |
| 起泡排序 | **O（n2）** | **O（1）** | **稳定** |
| 快速排序 | **O（nlogn-n2）** | **O（1）** | **不稳定** |
| 简单选择排序 | **O（n2）** | **O（1）** | **不稳定** |
| 堆排序 | **O（nlogn）** | **O（1）** | **不稳定** |
| 归并排序 | **O（nlog2n）** | **O（n）** | **稳定** |
| 基数排序 |  |  | **稳定** |