# 数据结构与算法基础

## 数组与矩阵

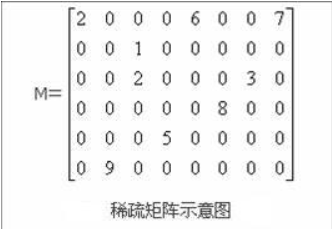
### 数组

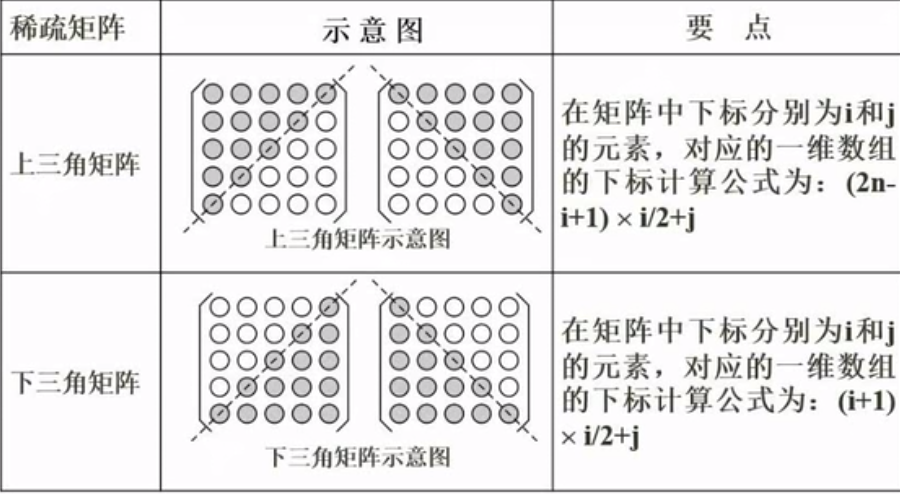
|  |  |
| --- | --- |
| 数组类型 | 存储地址计算 |
| 一维数组a[n] | a[i]的存储地址为：a+i\*len |
| 二维数组a[m][n] | a[i][j]的存储地址（按行存储）为：a + (i\*n+j)\*len  a[i][j]的存储地址（按列存储）为：a + (j\*m+i)\*len |

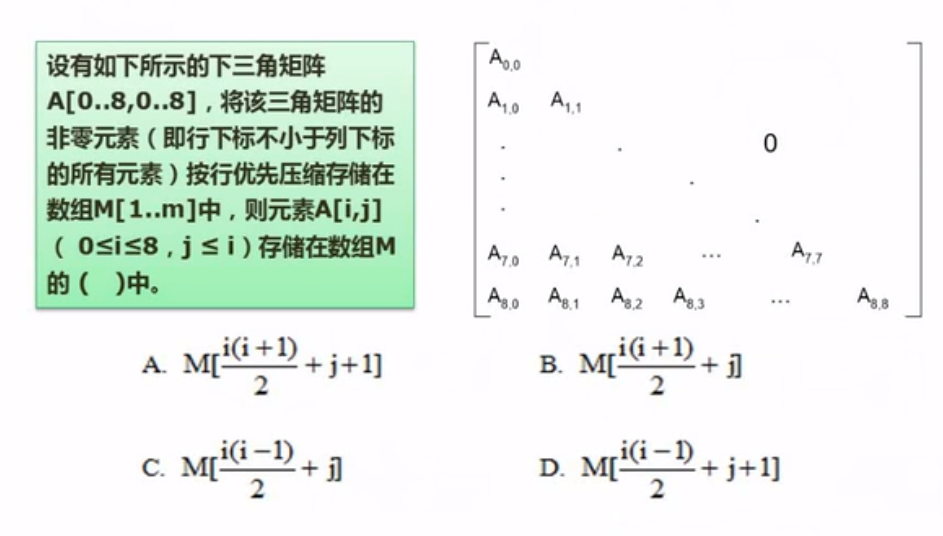
已知5行5列的二维数组a的各元素占两个字节，求元素a[2][3]按行优先存储的存储地址。

### 稀疏矩阵

在矩阵中，若数值为0的元素数目远远多于非0元素的数目，并且非0元素分布没有规律时，则称该矩阵为稀疏矩阵；与之相反若非0元素数目占大多数时，则称该矩阵为稠密矩阵。定义非0元素的总数比上矩阵所有元素的总数为矩阵的稠密度。





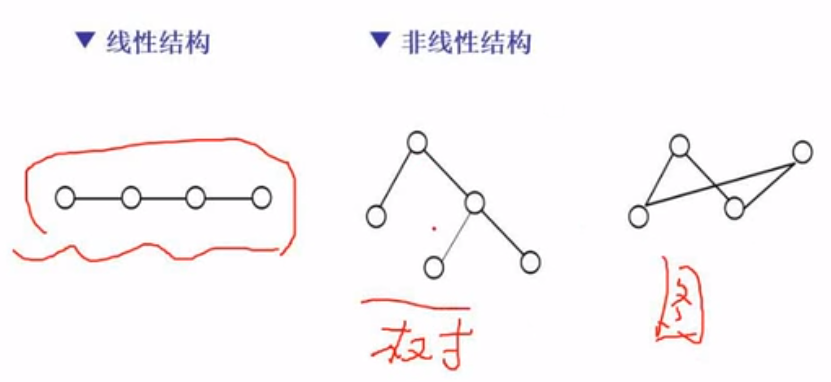


## 数据结构的定义

1. 数据结构的概念

计算机存储和组织数据的方式

1. 数据逻辑结构



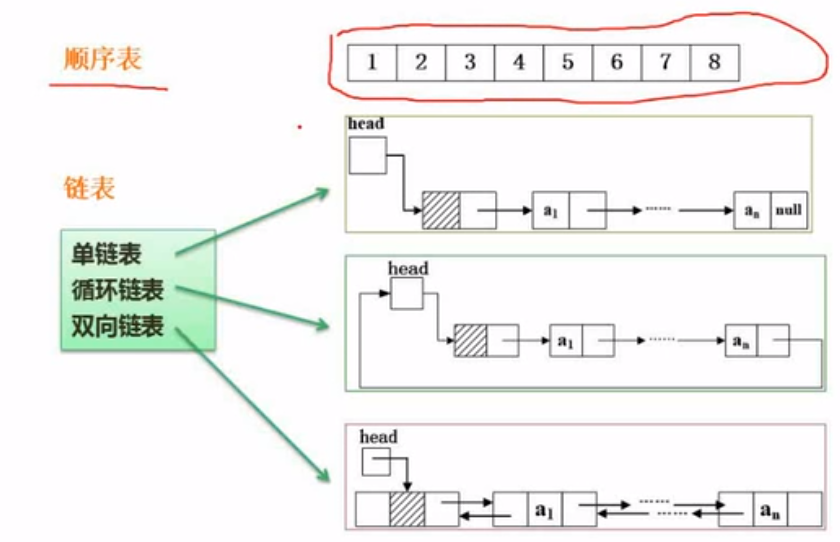
## 线性表

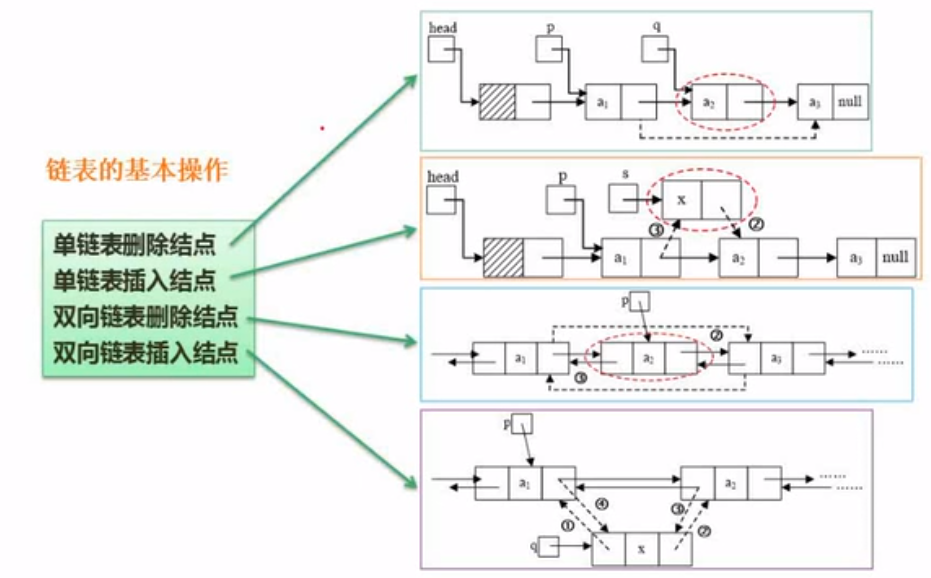
### 线性表的概念

（a1,a2,…an）

### 线性表常见的两种存储结构



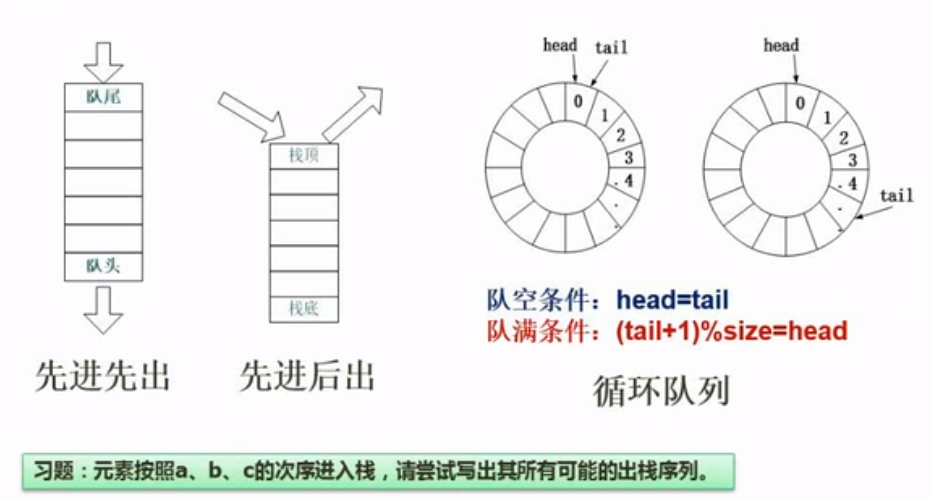


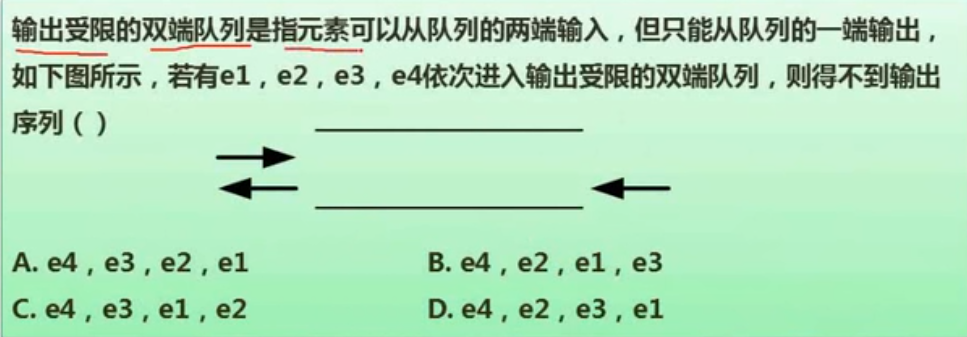


### 顺序存储与链式存储对比

|  |  |  |  |
| --- | --- | --- | --- |
| 性能类别 | 具体项目 | 顺序存储 | 链式存储 |
| 空间性能 | 存储密度 | =1（利用率100%），更优 | <1（利用率小于100%） |
| 容量分配 | 事先确定 | 动态改变，更优 |
| 时间性能 | 查找运算 | O（n/2） | O（n） |
| 读预算 | O（1），更优 | O（[n+1]/2），最好情况为1，最坏情况为n |
| 插入运算 | O(n/2)，最好情况为0，最坏情况为n | O（1），更优 |
| 删除运算 | O([n-1]/2) | O(1)，更优 |

### 队列与栈





答案D

## 广义表

广义表是n个表元素组成的有限序列，是线性表的推广。

通常用递归的形式进行定义，记作：

注：其中LS是表名，ai是表元素，它可以是表（称作子表），也可以是数据元素（称为原子）。其中n是广义表的长度（也就是最外层包含的元素个数），n=0是广义表为空表；而递归定义的重数就是广义表的深度，直观的说，就是定义中所含括号的重数（原子的深度为0，空表的深度为1）。

基本运算：取表头head（Ls）和取表尾tail（Ls）

若有：LS1 = （a,(b,c),(d,e)）

head(LS1) = a

tail(LS1) = ((b,c),(d,e))

1. 有广义表LS1 = （a,(b,c),(d,e)），则其长度为？3深度为？2
2. 有广义表LS1 = （a,(b,c),(d,e)）,要将其中的b字母取出，操作就为？

tail(LS1) = ((b,c),(d,e)) = LS2

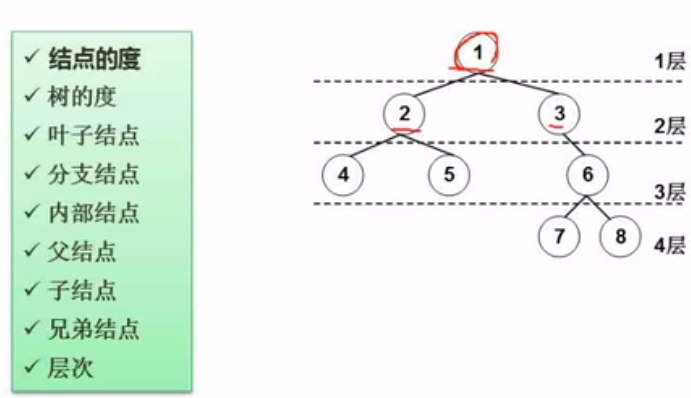
head(LS2) = (b,c) = LS3

head(LS3) = b

**head(head(tail(LS1))) = b**

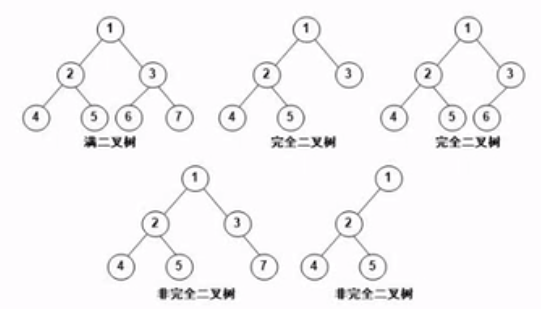
## 树与二叉树

在计算机科学中，二叉树是每个节点最多有两个子树的树结构。通常子树被称作“左子树（left subtree）”和右子树（right subtree）“。二叉树常被用于实现二叉查找树和二叉堆。

一棵深度为k，且有2^k-1个节点的二叉树，称为满二叉树。这种树的特点是每一层上的节点数都是最大节点数。而在一棵二叉树中，除最后一层外，若其余层都是满的，并且最后一层或者是满的，或者是在右边缺少连续若干节点，则此二叉树为完全二叉树。具有n个节点的完全二叉树的深度为floor(log2n)+1。深度为k的完全二叉树，至多有2k-1个叶子节点，至多有2k-1个节点。

节点的度：子节点的数目

树的度：节点的度的最大值



### 二叉树的重要特性

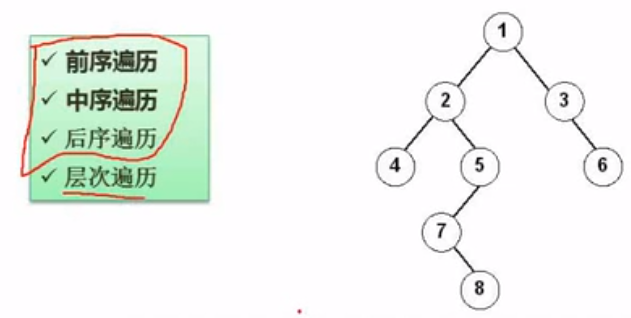
1. 在二叉树的第i层上最多有2i-1个节点，（i>=1）
2. 深度为k的二叉树最多有2k-1个节点（k>=1）
3. 对任何一颗二叉树，如果其叶子节点数为n0，度为2的节点数为n2，则n0=n2+1
4. 如果对一棵有n个节点的完全二叉树的节点按层次编号（从第1层到[log2n]+1层，每层从左到右），则对任一节点i（1<=i<=n），有：

如果i=1，则节点i无父节点，是二叉树的根；如果i>1，则父节点是i/2；

如果2i>n，则节点i为叶子节点，无左子节点；否则，其左子节点是节点2i；

如果2i + 1 > n，则节点i无右子节点，否则，其右子节点是节点2i + 1。

### 二叉树遍历



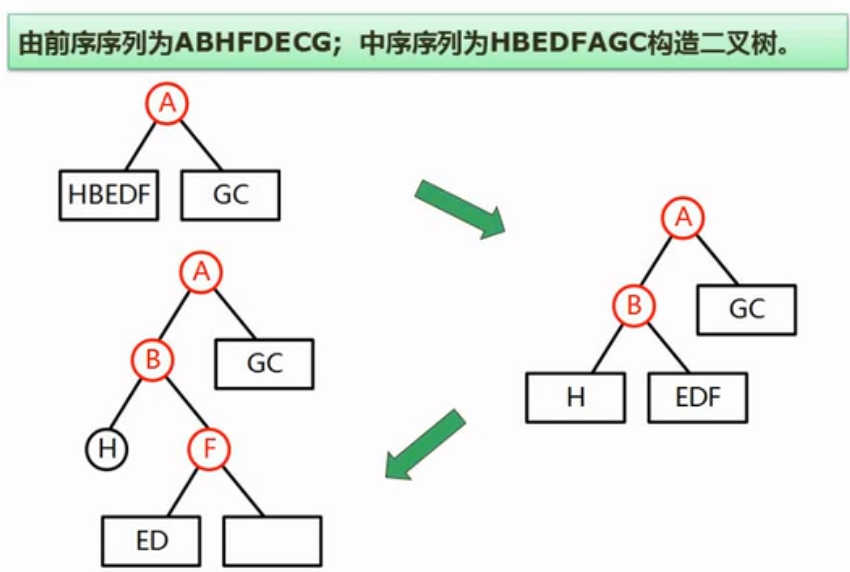
图中前序遍历结果是：（先访问根节点，再左子树，再右子树，递归）1 2 4 5 7 8 3 6

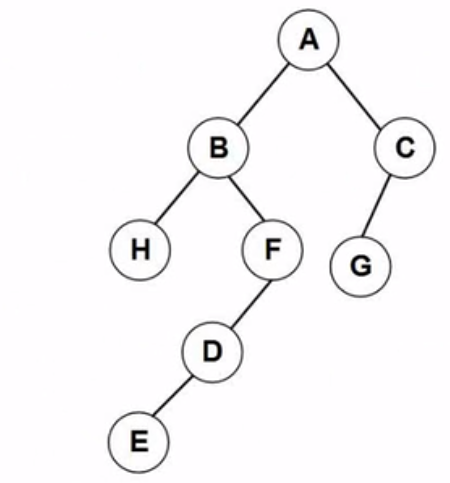
图中中序遍历结果是：（先访问左子树节点，再根节点，再右子树，递归）4 2 7 8 5 1 3 6

图中后序遍历结果是：（先访问左子树节点，再右子树，再根节点，递归）4 8 7 5 2 6 3 1

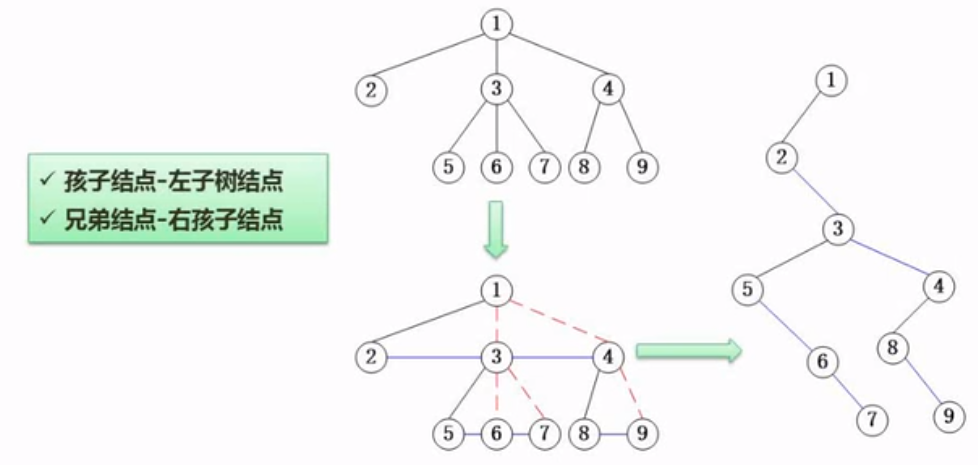
图中层次遍历结果是：1 2 3 4 5 6 7 8

### 反向构造二叉树





### 树转二叉树



### 查找二叉树

二叉查找树（Binary Search Tree），（又：二叉搜索树，二叉排序树）它或者是一棵空树，或者是具有下列性质的二叉树： 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值； 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值； 它的左、右子树也分别为二叉排序树。

二叉树排序特性：

左孩子小于根

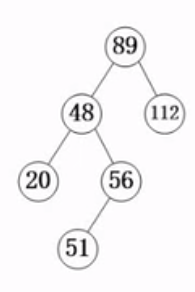
右孩子大于根

插入节点：

1. 若该键值节点已存在，则不插入，如48
2. 若查找二叉树为空，则以新节点为查找二叉树
3. 将要插入节点键值与插入后父节点键值比较，就能确定新节点是父节点的左子节点，还是右子节点

删除节点：

1. 若待删除节点是叶子节点，则直接删除
2. 若待删除节点只有一个子节点，则将这个子节点与待删除节点的父节点直接连接，如56
3. 若待删除节点p有两个子节点，则在其左子树上，用中序遍历寻找关键值最大的节点s，用节点s的值代替节点p的值，然后删除节点s，节点s必属上述①，②情况之一，如89



### 最优二叉树（哈夫曼树）

需要了解的基本概念：

树的路径长度、权、带权路径长度、树的带权路径长度（树的代价）

1. 路径和路径长度

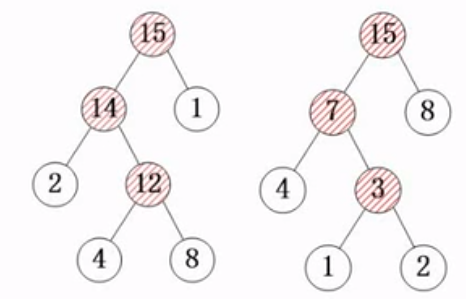
在一颗树中，从一个节点往下可以达到的孩子或孙子节点之间的通路，称为路径。通路中分支的数目称为路径长度。若规定根节点的层数为1，则从根节点到第L层节点的路径长度为L-1 。

1. 节点的权及带权路径长度

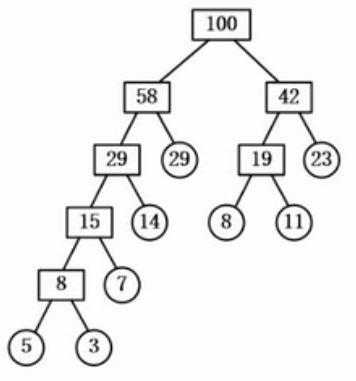
若将树中节点赋给一个有着某种含义的数值，则这个数值称为该节点的权。节点的带权路径路径长度为：从根节点到该节点之间的路径长度与该节点的权的乘积。

1. 树的带权路径长度

树的带权路径长度规定为所有叶子节点的带权路径长度之和，记为WPL。



例：假设有一组权值5，29，7，8，14，23，3，11请尝试构造哈夫曼树



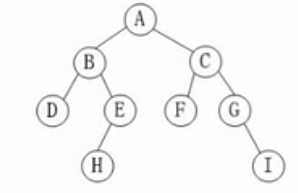
### 线索二叉树

为什么要有线索二叉树

线索二叉树的概念

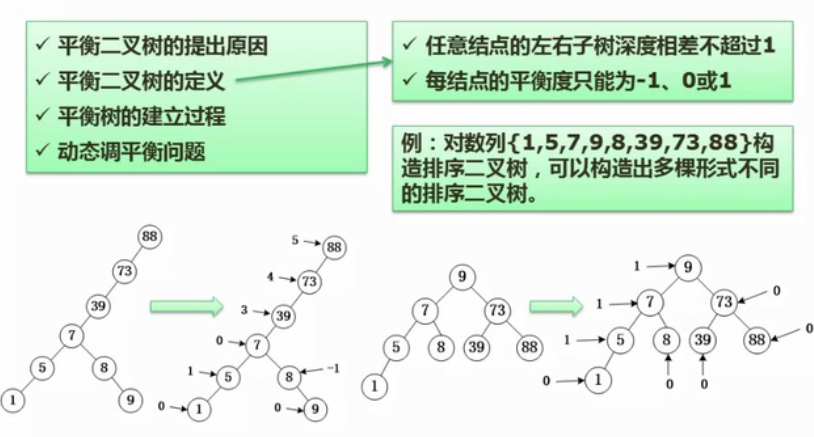
线索二叉树的表示

如何将二叉树转换为线索二叉树





### 平衡二叉树



## 图

### 基本概念

#### 完全图

在无向图中，若每对顶点之间都有一条边相连，则称改图为完全图（complete graph）。

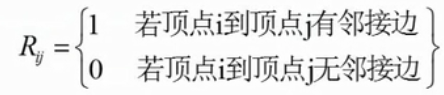
在有向图中，若每对顶点之间都有二条有向边相互连接，则称该图为完全图。

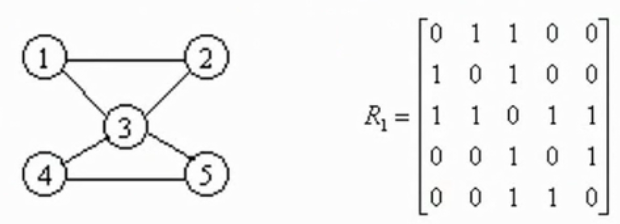


### 图的存储

#### 邻接矩阵

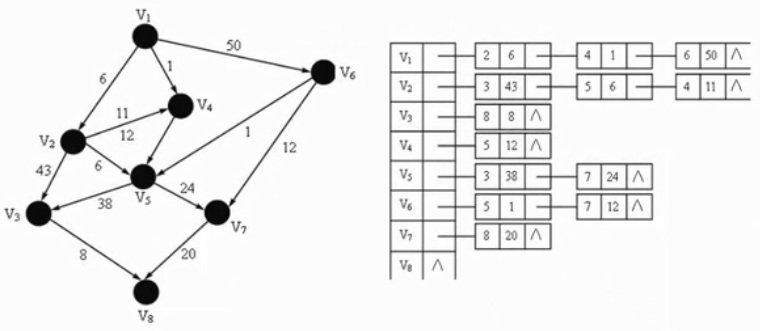
用一个n阶方阵R来存方图中各节点的关联信息，其矩阵元素Rij定义为：





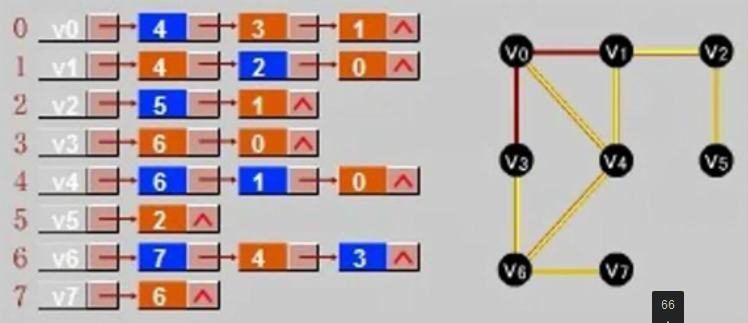
#### 邻接表

首先把每个顶点的邻接顶点用链表表示出来，然后用一个一维数组来顺序存储上面每个链表的头指针。



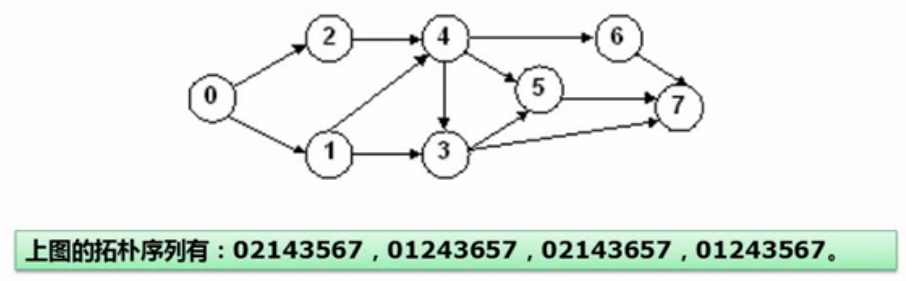
### 图的遍历

|  |  |  |  |
| --- | --- | --- | --- |
| 遍历方法 | 说明 | 示例 | 图例 |
| 深度优先 | 1. 首先访问出发顶点V 2. 依次从V出发搜索V的任意一个邻接点W 3. 若W未访问过，则从该点出发继续深度优先遍历，他类似于树的前序遍历 | V1，V2，V4，V8，V5，V3，V6，V7 |  |
| 广度优先 | 1. 首先访问出发顶点V 2. 然后访问于顶点V邻接的全部未访问顶点W、X、Y…； 3. 然后再依次访问W、X、Y…邻接的未访问的顶点 | V1，V2，V3，V4，V5，V6，V7，V8 |



### 拓扑排序

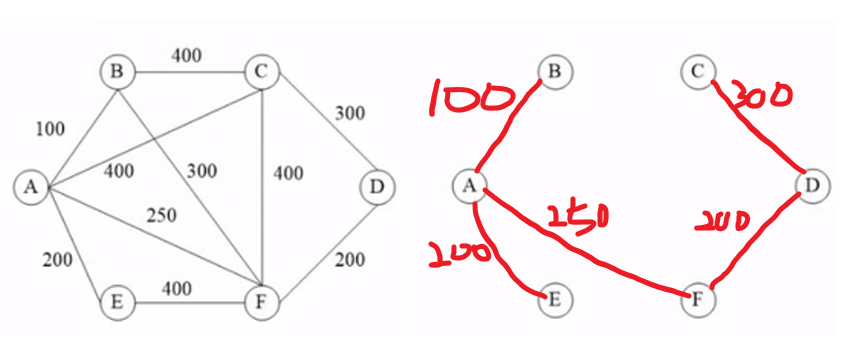
我们把用有向边表示活动之间开始的先后关系。这种有向图称为用顶点表示活动网络，简称AOV网络。



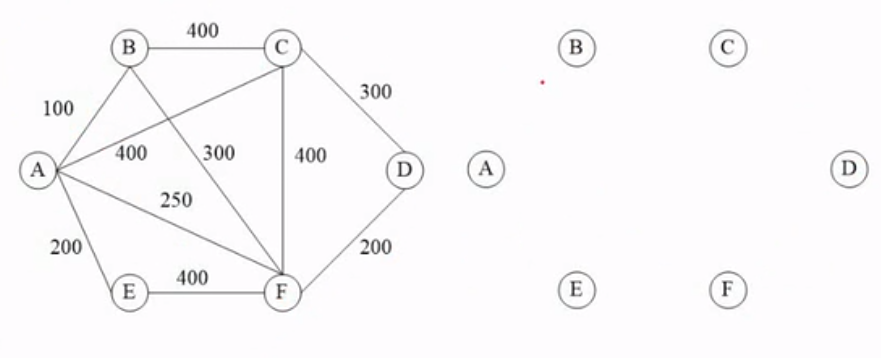
### 图的最小生成树

#### 普里姆算法

节点数-边数 = 1



#### 克鲁斯卡尔算法



## 算法基础及常见的算法

### 算法的特性

有穷性：执行有穷步之后结束

确定性：算法中每一条指令都必须有确切的含义，不能含糊不清

输入（>=0）

输出（>=1）

有效性：算法的每个步骤都能有效执行并能得到确定的结果。例如a=0，b/a就无效。

### 算法的复杂度

时间复杂度：是指程序运行从开始到结束所需要的时间。通常分析时间复杂度的方法是从算法中选取一种对于所研究的问题来说是基本运算的操作，以该操作重复执行的次数作为算法的时间度量。一般来说，算法中原操作重复执行的次数是规模n的某个函数T（n）。由于许多情况下要精确计算T（n）是困难的，因此引入了渐进时间复杂度在数量上估计一个算法的执行时间。其定义如下：

如果存在两个常数c和m，对于所有的n，当n>=m时有f（n）<=cg(n)，则有f(n)=O(g(n))。也就是说，随着n的增大，f(n)渐进的不大于g(n)。例如，一个程序的实际执行时间为T(n)=3n3+2n2+n，则T(n)=O(n3) 。

常见的对算法执行所需时间的度量：

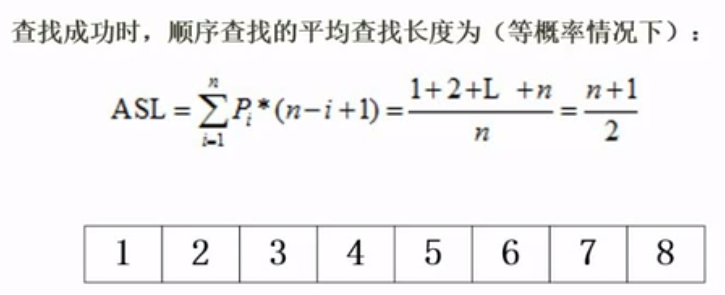
O(1)<O(log2n)<O(n)<O(n log2n)<O(n^2)<O(n^3)<O(2^n)

空间复杂度：是指对一个算法在运行过程中临时占用存储空间大小的度量。一个算法的空间复杂度只考虑在运行过程中为局部变量分配的存储空间的大小。

### 排序与查找

#### 顺序查找

顺序查找的思想：将待查找的关键字为key的元素从头到尾于表中元素进行比较，如果中间存在关键字为key的元素，则返回成功；否则，查找失败。



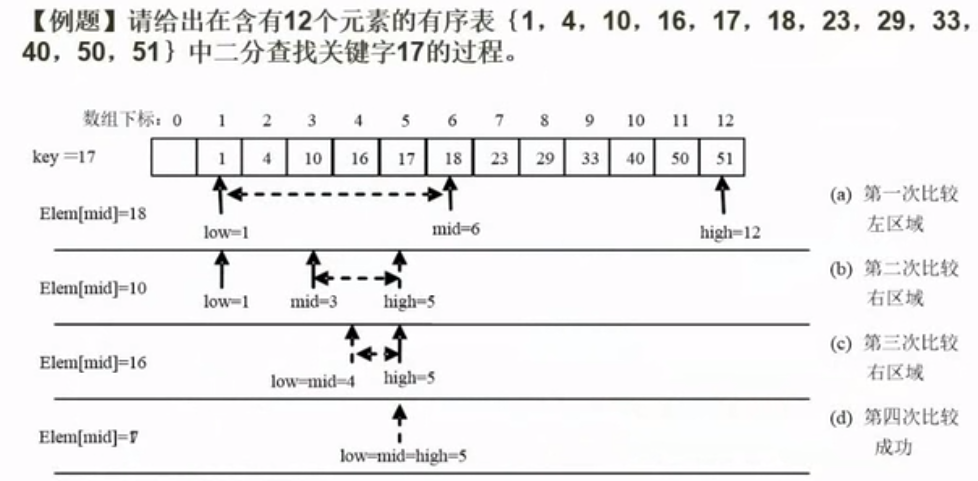
#### 二分查找

二分法查找的基本思想是：（设R[low,…,high]是当前的查找区）

1. 确定该区间的中点位置：mid = （low+high）/2；
2. 将待查的k值于R[mid].key作比较，若相等，则查找成功并返回此位置，否则需确定新的查找区间，继续二分查找，具体方法如下。

若R[mid].key>k，则由表的有序性可知，R[mid,…,high].key均大于k，因此若表中存在关键字等于k的节点，则该节点必定是在位置mid的左边子表R[low,…,mid-1]中。

1. 下一次查找是针对新的查找区间进行，重复（1）和（2）
2. 在查找过程中,low逐步 增加，而high逐渐减少。如果high<low，则查找失败，算法结束。

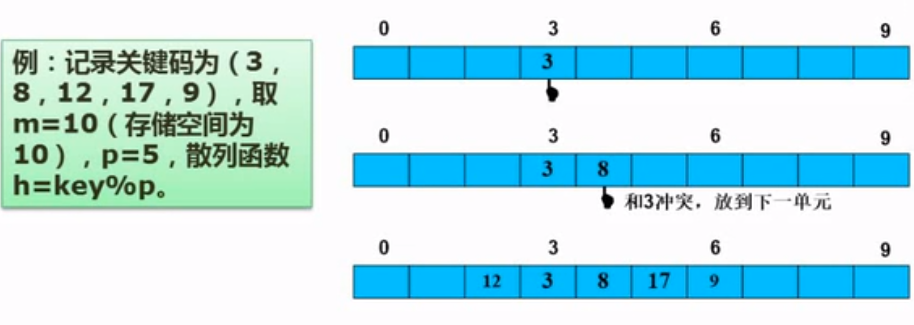


折半查找在查找成功是关键字的比较次数最多为[log2n]+1次

折半查找的时间复杂度为O(log2n)

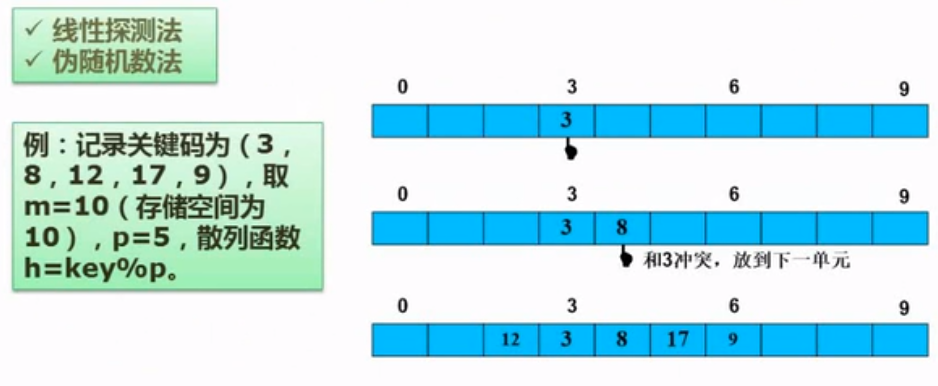
#### 散列表

散列表查找的基本思想是：已知关键字集合U，最大关键字为m，设计一个函数hash，他以关键字为自变量，关键字的存储地址为因变量，将关键字映射到一个有限的、地址连续的区间T[0…n-1](n<<m)中，这个区间就称为散列表，散列表查找中使用的转换函数称为散列函数。



#### 散列表冲突的解决办法

开放定址法是指当构造散列表发生冲突时，使用某种探测手段，产生一个探测的散列地址序列，并且逐个产找此地址中是否存储了数据元素，如果没有，则称该散列地址开放，并将关键字存入，否则继续查找下一个地址。只要散列表足够大，总能找到空的散列地址将数据元素存入。



#### 排序

##### 排序的概念

21 32 13 45 27 38 76 21

稳定与不稳定排序

序列中有两个21，排序后两个21的前后顺序不变-稳定

内排序与外排序

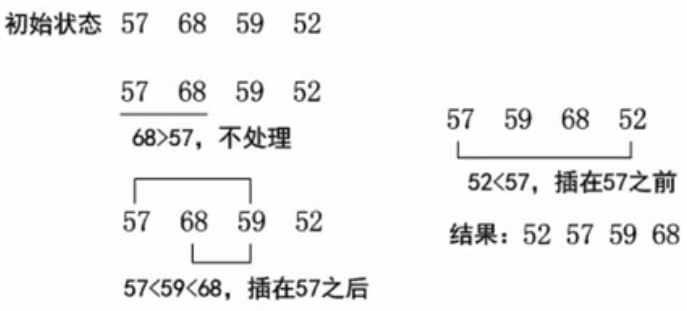
内排序：在内存中进行排序

##### 排序的方法分类

###### 插入类排序

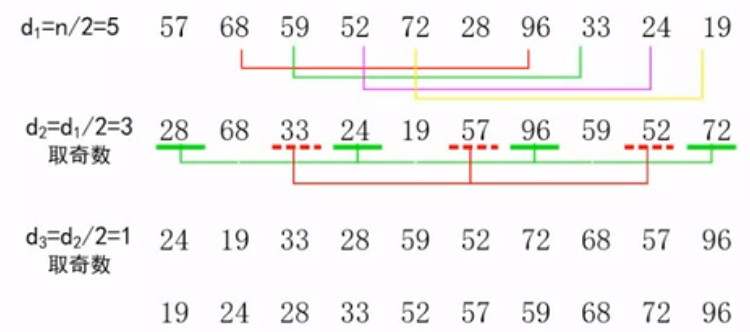
直接插入排序

直接插入排序：即当插入第i个记录时，R1,R2,…,Ri-1均以排好序，因此，将第i个记录Ri依次与Ri-1,…,R2,R1进行比较，找到合适的位置插入。他简单名明了，但速度很慢。



希尔排序

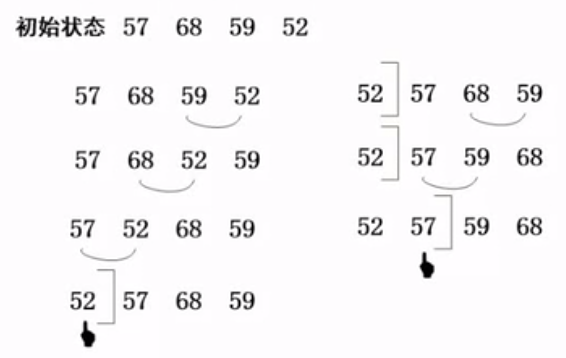
希尔（shell）排序：先选取一个小于n的整数d1作为第一个增量，把文件的全部记录分成d1个组。所有距离为d1的倍数的记录放在同一个组中。先在各组内进行直接插入排序；然后，取第二个增量d2<d1重复上述的分组和排序，直至所取得增量dt=1（dt<dt-1<o<d2<d1），即所有记录放在同一组中进行直接插入排序为止。该方法实质上是一种分组插入方法。



###### 变化类排序

冒泡排序

冒泡排序的基本思想是，通过相邻元素之间的比较和交换，将排序码较小的元素逐渐从底部移向顶部。由于整个排序的过程就像水底下的气泡一样逐渐向上冒，因此称为冒泡算法。



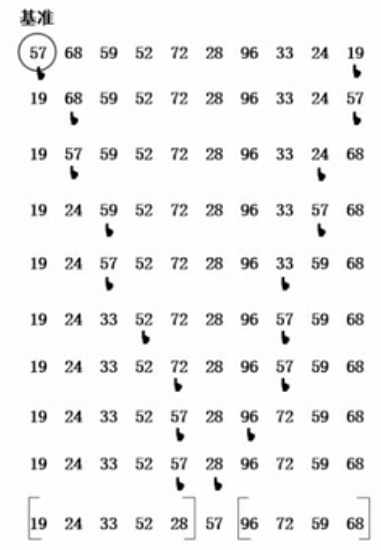
快速排序

快速排序采用的是分治法，其基本思想是将原问题分解成若干个规模更小但结构与原问题相似的子问题。通过递归地解决这些子问题，然后再将这些子问题的解组合成原问题的解。

快速排序通常包括两个步骤：

第一步：再待排序的n个记录中任取一个记录，以该记录的排序码为准，将所有记录都分成两组，第一组都小于该数，第二组都大于该数，如图所示。

第二步：采用相同的方法对左、右两组分别进行排序，直到所有记录都排到相应的位置为止。



###### 选择类排序

简单选择排序

直接选择排序

直接选择排序的过程是，首先在所有记录中选出排序码最小的记录，把他与第一个记录交换，然后在其余的记录内选出排序码最小的记录，与第二个记录交换……一次类推，直到所有记录排完为止。



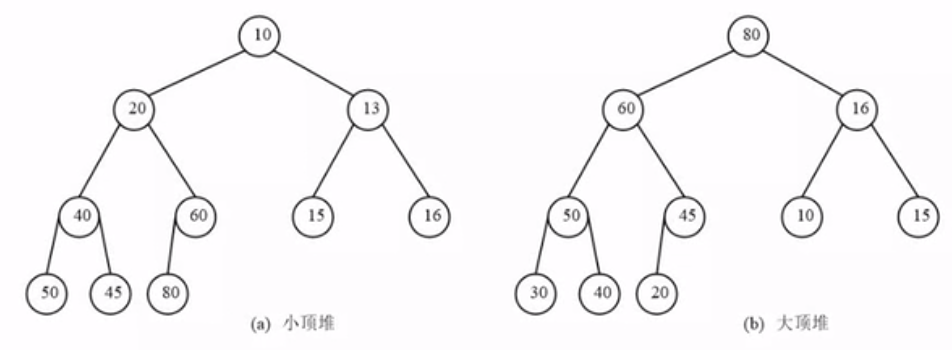
堆排序

堆的概念

设有n个元素的序列{k1,k2,k3,…,kn}，当且仅当满足下述关系之一时，称为堆（下标从0开始则应该为k2i+1和k2i+1+1）。

1. ki<=k2i且ki<=k2i+1；
2. ki>=k2i且ki>=k2i+1；

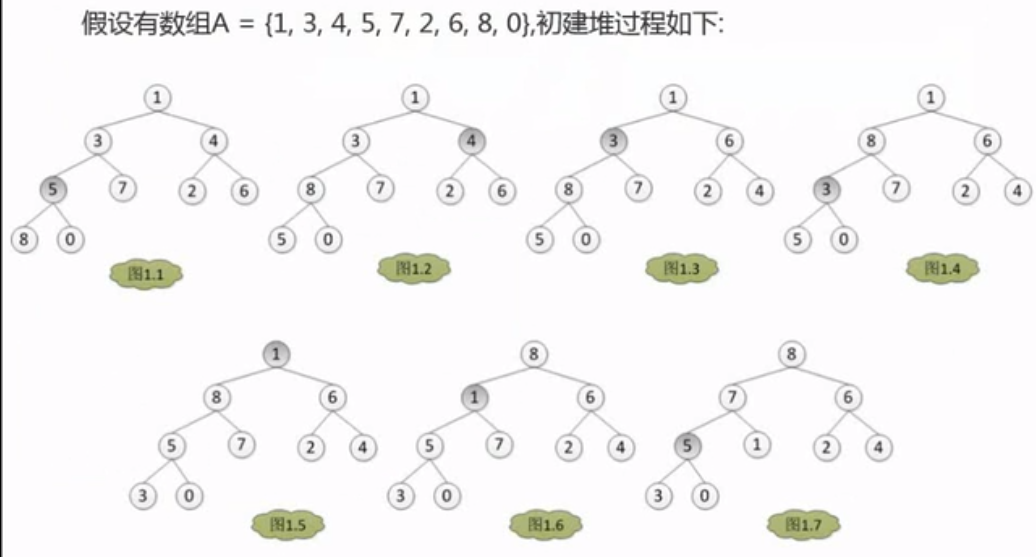
其中（1）称为小顶堆，（2）称为大顶堆。



堆排序的基本思想为：先将序列建立堆，然后输出堆顶元素，再将剩下的序列建立堆，然后再输出堆顶元素，一次类推，直到所有元素均输出为止，此时元素输出的序列就是一个有序序列。

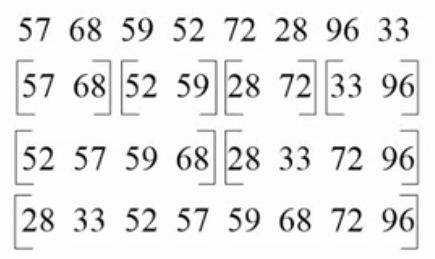
堆排序的算法步骤如下（以大顶堆为例）：

1. 初始时将顺序表R[1…n]中建立为一个大顶堆，堆顶位于R[1]，待序区为R[1…n]。
2. 循环执行步骤3~步骤4，共n-1次。
3. 假设为第i次运行，则待序区为R[1…n-i+1]，将堆顶元素R[1]与待序区尾元素R[n-i+1]交换，此时顶点元素被输出，新的待序区为R[1,…,n-i]。
4. 待序区对应的堆已经被破坏，将之重新调整为大顶堆。



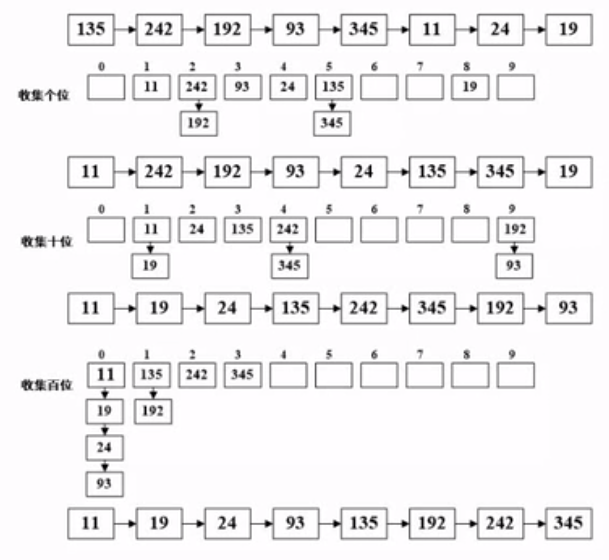
###### 归并排序

归并也称为合并，是将两个或两个以上的有序子表合并称一个新的有序表。若将两个有序表合并成一个有序表，则称为二路合并。合并的过程是：比较A[i]和A[j]的排序码大小，若A[i]的排序码小于等于A[j]的排序码，则将第一个有序表中的元素A[i]复制到R[k]中，并令i和k分别加1；如此循环下去，直到其中一个有序表比较和复制完，然后再将另一个有序表的剩余元素复制到R中。



###### 基数排序

基数排序是一种借助多关键字排序思想对单逻辑关键字进行排序的方法。基数排序不是基于关键字比较的排序方法，他适合于元素很多而关键字较少的序列。基数的选择和关键字的分解是根据关键字的类型来决定的，例如关键字是十进制数，则按个位、十位来分解。



### 排序算法的时间复杂度和空间复杂度\*\*\*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类别 | 排序方法 | 时间复杂度 | | 空间复杂度 | 稳定性 |
| 平均情况 | 最坏情况 | 辅助存储 |
| 插入排序 | 直接插入 | O(n^2) | O(n^2) | O(1) | 稳定 |
| Shell排序 | O(n^1.3) | O(n^2) | O(1) | 不稳定 |
| 选择排序 | 直接排序 | O(n^2) | O(n^2) | O(1) | 不稳定 |
| 堆排序 | O(nlog2n) | O(nlog2n) | O(1) | 不稳定 |
| 交换排序 | 冒泡排序 | O(n^2) | O(n^2) | O(1) | 稳定 |
| 快速排序 | O(nlog2n) | O(n^2) | O(log2n) | 不稳定 |
| 归并排序 | | O(nlog2n) | O(nlog2n) | O(n) | 稳定 |
| 基数排序 | | O(d(r+n)) | O(d(r+n)) | O(r+n) | 稳定 |