数据类型从低到高自动转换(int - float)

逻辑结构：数据元素之间的逻辑关系，即结构中定义的‘关系’

线性表：

1. 线性表的元素个数是有限的
2. 线性开始节点没有前驱
3. 线性结束节点没有后继

算法：

1. 时间复杂度 （执行时间效率）
2. 空间复杂度 （占用存储空间)

例子：

1. 水仙花数

所谓水仙花数：指一个3位数，其各位数字的立方和等于数字本身，例如：153=1\*\*\*3+5\*\*\*3+3\*\*\*3

求出100~999之间的水仙花数

算法：比如 i

百位 a (1) a=i//100 (python //整除 其他语言/整除)

A = int(i /100)

十位 b (1) b=(i%100)//10

B = int((i-100\*a)/10)

个位 c (1) c=i%10

C = i-a\*100-b\*10

C = i-int(i/10)\*10

1. 计算10000以内的自守数

自守数：某个数的平方的末尾数等于这个数

Eg: 5x5=25

6x6 =36

25x25=625

76x76=5776

找规律：3位自守数是625和376，四位自守数是9376，5位自守数是90625

可以得知,n+1位自守数出自n位自守数

如果知道n位自守数为a，那么n+1的自守数应该在a的前面再加一个数

算法：n\*n%(10\*\*len(str(n)))) ==n

1. 出售金鱼

鱼商A将养的一缸金鱼分5次出售，第一次卖出全部的1/2加1/2条，

第二次卖出余下的1/3加1/3条，第三次卖出余下的1/4加1/4条，

第四次卖出余下的1/5加1/5条,最后卖出余下的11条.问原来鱼缸多少条鱼?

算法:第j次卖余下的1/(j+1)加/(j+1)条

假设第j次鱼的总数为x条

第j次留下（x-(x+1)/(j+1)）条

第一次x x -1/2\*x-1/2

第二次x -1/2\*x-1/2 x -1/2\*x-1/2-(1/3x-1/3)

第三次x -1/2\*x-1/2-(1/3x-1/3) x -1/2\*x-1/2-(1/3x-1/3)-(1/3(x-1/2\*x-1/2-(1/3x-1/3) +1/3)

第四次x -1/2\*x-1/2-(1/3x-1/3)-(1/3(x-1/2\*x-1/2-(1/3x-1/3) +1/3) x -1/2\*x-1/2-(1/3x-1/3)-(1/3(x-1/2\*x-1/2-(1/3x-1/3)+1/3)-1/4(x-1/2\*x-1/2-(1/3x-1/3)-(1/3(x-1/2\*x-1/2-(1/3x-1/3) +1/3)+1/4)

递归算法：

59

第一次卖出30， 剩余 29

第二次卖出10， 剩余19

第三次卖出5， 剩余14

第四次卖出3， 剩余11

小明有5本书，借给甲乙丙三位小朋友，每人每次只能借一本，有多少种不同借法

A,b,c,d,e,f

(a,b,c) (a,b,d)(a,b,e)(a,b,f)

某天夜里，a,b,c,d,e,五人一起去捕鱼，到第二天凌晨时都疲惫不堪，于是各自找地方睡觉，天亮啦，a第一个醒来，他将鱼分为5份，把多余的一条鱼扔掉，拿走自己的一份，

b第二个醒来，他将鱼分为5份，把多余的一条鱼扔掉，拿走自己的一份，c,d,e依次醒来，按照同样方法拿走鱼，问他们伙伴至少捕捉了多少条鱼？ 3121

栈

顺序栈：是利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素

构成：（1）一维数组：data

1. 栈顶指针top

栈空：top=-1

栈满：top=maxSize=-1

概念：栈是一种特殊的线性表：插入、删除受限

特殊：只能在一端进行操作，称为栈顶，top

入栈：插入数据

出栈：删除数据

入栈序列为

栈的顺序存储

1. 栈的初始化 构造函数 \_\_init\_\_(self)
2. 构造函数：一种特殊的方法，创建对象，初始化对象，对成员变量赋初值

栈的链式存储

节点构成：（1）数据：data

1. Next

链栈：

进栈函数

思路：

1. 创建一个新的结点，并将要进栈的元素存入该结点的数据域

tStackNode = StackNode()

tStackNode.data = da

1. 将新的结点的指针域向栈顶结点指针域指向的结点

tstackNode.next = self.top.next

1. 将栈顶结点的指针域指向新结点

Self.top.next = tStackNode

出栈函数

1. 判断栈是否为空
2. 如果为空，则无法执行出栈操作，给出栈为空提示，否则执行(3)
3. 记下此时栈顶结点指针域指向的结点
4. 秀发栈顶结点的指针域，在其中存入(3)记下结点指针域的值
5. 将数值域为x的结点出栈。

链式栈

构成：（1）数据：data

1. 指针：next

约定：不带头结点

优点：多个栈可共享空间

Top 栈顶结点

|  |  |
| --- | --- |
| 28 | ... |
| 12 | ... |
| top |  |

出栈:

使用tstackNode记住栈顶结点指向的结点

tstackNode = self.top.next

Self.top.next = tstackNode.next

等同于self.top.next.next

Return tstackNode.data

队列

队列包括顺序存储结构和链式储存结构

概念:一种特殊的线性表，插入在限定在表的某一端进行，删除在表的另一端进行

队尾：允许插入的一端

队头：允许删除的一端

空队列：不含任何元素的队列

出队 a1 a2 a3 ... an 入队

队头 队尾

特点：先进先出(FIFO)，后进后出 (LILO)

基本运算：

初始化

销毁

进队

出队

取队头元素

判断队空

顺序队列：通常约定队尾指针指示队尾元素的当前位置，队头指针指示队头元素的前一个位置

顺序存储结构：（1）一维数组 data

1. 队头和队尾指针：front,rear

约定：队尾指针指示队尾元素的当前位置

队头指针指示队头元素的前一个位置

队空条件：front==rear

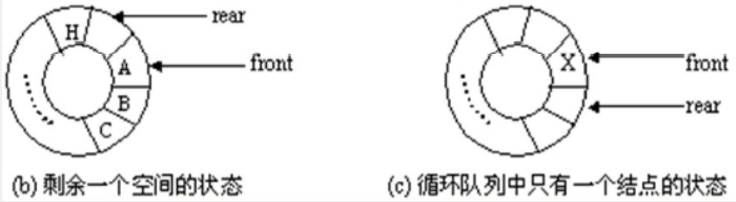
队满条件:rear=Maxszie-1

假溢出：用循环队列或环形队列解决

已知顺序队列sq:

1. 队空条件：sq.front==sq.rear
2. 队满条件:（sq.rear+1)%MaxSize==sq.front

front跟rear相邻，而且在rear前面



1. 元素x入队列
2. sq.rear=(sq.rear+1)%MaxSize
3. 将元素x存入sq.data[sq.rear]中：sq.data[sq.rear]=x
4. 队列顶元素x出队列
5. sq.front = (sq.front+1)%MaxSize
6. 取出队列元素:x=sq.data[sq.front]

注意：显然在这样设置的队满条件下，队满条件成立时队中还有以-一个空闲队列，也就是说这样的队中最多只能进MaxSize-1个元素

循环队列中队头，队尾指针变化

(rear+MaxSize-front)%MaxSize

# 链式队列

队列的链式存储结构简称为链队，它实际上是一个同时带有队头指针front和队尾指针rear的单链表

队头指针指向队头结点，队尾指针指向队尾结点即单链表的最后一个结点，并将队头和队尾指针结合起来构成链队结点



已知链队列ls

1. 队空条件：front==rear
2. 队满条件：不考虑（因为每个结点都是动态分布的）
3. 元素x入队：

创建结点P,将其插入到队尾，并由rear指

P.data = x

1. 出队操作:

删除队头的结点

进队运算操作

主要操作:创建一个新结点，将其链接到链队的末尾，并由rear指向它

出队

1. 队列例里有没有数据 front==rear

没有数据（空），队列为空，无法出队

1. 队列里有一个数据

If self.front.next.next == None:

Self.front.next=None

Self.rear=self.front

1. 队列里有2个或以上数据

tNode = self.front.next

Self.front = tNode.next

Return tNode.data

串

串的定义：串是由零个或者多个字符组成的有限序列，一般记为:

Str=’a1a2....an’(n≥0）

Str是串名，用双引号括起来的字符序列是串的值：ai(1≤i≤n)可以是字母，数字或其他字符，该字符的逻辑序号为i

串中的字符个数n称为串的长度，长度为零的串称为空串其长度为0

子串：串中任意一个连续的字符组成的子序列称为该串的子串，空串是任何串的子串，例如：串’abc’的子串有’’,’a’,’b’,’c’,’ab’,’bc’,’abc’

串相等：两个串相等当且仅当它们的长度相等且对应位置上的字符相同

# 树和二叉树

树的特点：非线性结构，一个直接前驱，但可能有多个直接后继(1:n)

树的定义：由一个或多个(n≥0)结点组成的有限集合，在任何一颗非空树T中：

1. 有且仅有一个结点称为根(root)
2. 当n>1时，其余的结点分为m(m≥0)个互不相交的有限集合T1,T2...Tm。每个集合本身又是树，被称为这个根的子树.

注：树的定义具有递归性，即树中还有树.

树的表式法：

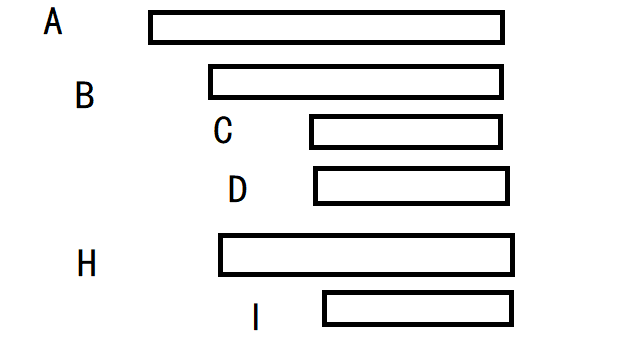
图形表示法

嵌套集合表示法：（画圆圈）

广义表表示法：根作为由字数森林组成的表的名字写在表的左边(括号表示)

示例：A(B(E(K,L),F),(C(G),D(H(M),I,J)

凹入表示法（目录表示法）



1. 示例术语

根--根结点（没有前驱）

有序树---结点各子树从左至右有序，不能互换（左为第一）

无序树---结点各子树可互换位置

双亲：上层的那个结点（直接前驱）

孩子：下层结点的子树的根（直接后继）

兄弟：同一双亲下的同一层结点（孩子间互称兄弟）

堂兄弟：双亲位于同一层的结点（但并非同一双亲)

祖先：从根到该结点所经分支的所有结点

子孙：该结点下层子树的任一结点

结点：树中的数据元素

结点的度：结点拥有的子树的数目（有几个直接后继就是几）

结点的层次：从根到该结点的层数（根结点算第一层）

叶子：度为0的结点（终端结点）

分支结点：度不为0 的点（非终端结点）

树的度：所有结点度中的最大值（Max{各结点的度}）

树的深度(或高度)：所有结点中最大的层数（Max{各结点的层次}）

森林：n(n>0)个互不相交的树的集合

删除树的根结点就成了森林。反之，给n棵独立的树增加一个结点，将n棵作为新增结点的树，树就成了森林

二叉树

二叉树的结构最简单，规律性最强

可以证明，所有的树都能转化为唯一对应的二叉树，不失一般性

1. 二叉树的定义：

基本特征：

1. 每个结点最多只有两棵子树（不存在大于2的结点）
2. 左子树和右子树次序不能颠倒（有序树）

性质1：在二叉树的第i层上至多有2i-1 个结点（i>0)

性质2：深度为k的二叉树至多有2k-1个结点(k>0)

性质3：对于任何一棵二叉树，若度为2的结点数有n2个，叶子数结点数为n0，则n0=n2+1

证明性质3：

全部结点数：n=n0+n1+n2（叶子数+度为1的结点数+度为2 的结点数）

（出根结点外，每个结点必有一个直接前驱，即一个分支）

二叉树中全部结点数n=B+1(总分支数+根结点）

总分支数：B=n1+2n2

## 满二叉树

二叉树中每层都是满的

定义:深度为且有2k-1个结点的二叉树,可以是完全二叉树

特点：每一层的结点数都是最大结点数。可以对满二叉树的结点纪进行连续编号

完全二叉树：

除最后一层外，其余层都是满的.

深度为k，有n个结点的二叉树，当且仅当每一个结点都与深度为k的满二叉树编号从1至n的结点一一所对应时，称为完全二叉树.

特点：（1）叶子结点只可能在层次最大的两层上面出现.

1. 对任一结点，若其右分支下的子孙的最大层次为h，则其左分支下的子孙的最大层数必为 h或h+1

性质4：具有n个结点的完全二叉树的深度为[log2n]+1 []表示向下取整

遍历二叉树

1. 先序遍历
2. 中序遍历
3. 后序遍历

# 二叉树的存储结构

1. 顺序存储结构：用一组地址连续的存储单元依次自上而下，自左至右存储二叉树上的结点单元
2. 仅适合于完全二叉树
3. 对于非完全二叉树：将各层空缺处全部补上‘虚结点’，其内容为0

## 链式存储结构

二叉树表中包含2个指针域，一般从根节点开始存储

为了便于找到结点的双亲，可再增加一个双亲域指针，将二叉表变为三叉表

遍历二叉树

1. 先序遍历：根结点，左子树，右子树
2. 中序遍历：左子树，根结点，右子树
3. 后序遍历：左子树，右子树，根结点

讨论：若已知先序序列（或后序序列）和中序序列，能否恢复出对应的二叉树

前序，中序，唯一

前序，后序，不唯一

# 树和森林

树的存储方式：（1）双亲表示法 （2）孩子表示法 （3)孩子兄弟表示法

1. 用双亲表示法存储

方法：用一组连续空间来存储树的结点，同时在每个结点中附设一个指示器，指示其双亲结点在链表中的位置。

1. 用孩子存储法来表示

方法：将每个结点的孩子排列起来，形成一个带表头（装父亲结点）的线性表（n个结点要设立n个链表),再将n个表头用数组存放起来，这样就形成一个混合结构

1. 孩子兄弟表示法

用二叉链表来存储树，但链表的两个指针域含义不同

# 哈夫曼树(最优二叉树)

路径：由一个结点到另一个结点间的分支所构成

路径长度：路径上的分支数目

树的路径长度：从根到每一结点长度之和

带权路径长度：结点到根的路径长度与结点上权值的乘积，公式:WPL=∑

树的带权路径长度：书中所有叶子结点的带权路径长度之和

哈夫曼树：带权路径长度最小的树

1. 构造哈夫曼树的基本步骤

一颗有n个结点的哈夫曼树共有2n-1个结点

## 哈夫曼编码

1. 统计频度
2. 用频度构造哈夫曼树
3. 约定：左分支为0，右分支为1,叶结点为对应字符
4. 从根到叶子路径的0或1的序列即为字符的哈夫曼编码

# 图

概念：图由两个集合V和E组成，记为G=(V,E)

V(G):顶点的有限集合

E(G)：连接V中两个不同顶点的边的有效集合

(V,E)代表无向图

<V,E>代表有向图

度,出度，入度

在无向图中，每个顶点的度为端点的边数

在有向图中，顶点的度=入度+出度

入度：以该顶点为终点的入边数目

在一个无向图中，若存在一条边(i,j)，则称顶点i、j为该边的两个端点，并称它们互为邻接点。

在一个有向图中，若存在一条边<i,j>，则称此边是顶点i的一条出边，同时也是顶点j的一条入边，称顶点i和j分别为此边的起始端点(起点)和终止端点(终点)。

1、在无向图中，每个顶点的度为端点的边数；

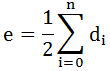
2、在有向图中，顶点v的度=入度+出度

(1)入度：以该顶点为终点的入边数目；

(2)出度：以该顶点为起点的出边数目；

边数、顶点数之间的关系

图(有向图或无向图)中有n个顶点、e条边，每个顶点的度为di(0≤i≤n-1)：



结论：一个图中所有顶点的度之和等于边数的两倍。

(图中每条边被两个邻接点的度重复计算)

子图

设有两个图G=(V,E)和G‘=(V’,E‘)，若V’是V的子集，即V‘ 属于V，且E'是E的子集，即E‘ 属于E，则称G'是G的子图。

完全无向图、完全有向图

1、完全无向图：具有n(n-1)/2条边的无向图；

2、完全有向图：具有n(n-1)条边的有向图。

简单路径：路径中顶点序列中的顶点不重复出现

回路(环)：路径中的开始点和结束点为同一点

简单回路(简单环)：

回路中除开始点、结束点相同外，其余顶点不同

连通、连通图、连通分量：

1、无向图中的两顶点有路径，称两顶点是连通的；

2、连通图：图中的任意两顶点都是连通的；

3、非连通图：图中至少有一对顶点不是连通的；

4、连通分量：无向图中的极大连通子图。

强连通图、强连通分量：

1、强连通图：有向图中的任意两顶点都是连通的；

3、强连通分量：有向图中的极大连通子图。

权、网：

1、边的权：边上标注的具有某种含义的数值；

3、网(带权图)：边上带权的图。

约定：边的权为非负数。

生成树、生成森林

假设一个连通图有 n 个顶点和 e 条边，其中 n-1 条边和 n 个顶点构成一个极小连通子图，称该极小连通子图为此连通图的生成树。

对非连通图，则称由各个连通分量的生成树构成的集合为此非连通图的生成森林。

图的基本运算

① 建立图CreateGraph(G)：建立图G的某种存储结构。

② 销毁图DestroyGraph(G)：释放图G占用的内存空间。

③ 输出图DispGraph(G)：显示图G的结构。

④ 添加顶点AddaVex(G)：在图G中添加顶点，编号自动累加求得。

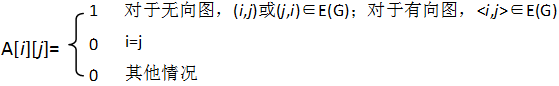
⑤ 插入边InsertEdge(G,u,v,w)：在图G中插入边<u,v>，其权值为w。

⑥ 求顶点的度Degree(G,v)：求图G中顶点v的度。

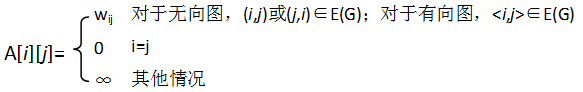
邻接矩阵：表示顶点间相邻关系的矩阵

1、已知图G=(V,E) ，顶点编号依次为0、1、…、n-1，

则G的邻接矩阵是n阶方阵A：



1. 已知图G=(V,E) 是带权图或网，其中wij为边(i,j)或<i,j>的权，则G的邻接矩阵A



邻接表：图的链式存储结构

1、每个顶点建立一个带头结点的单链表，用来串接该顶点的所有邻接点；

2、所有的头结点构成一个数组，称为头结点数组(adjlist[])；

3、第i个单链表adjlist[i]中的结点表示依附于顶点i的边；

4、单链表中的结点由3个域组成：

(1)顶点域adjvex：指示该邻接点在头结点数组中的下标；

(2)权值域weight：存放对应边的权值；

(3)指针域nextarc：指向依附于顶点i的下一条边对应结点。

约定：(1)不带权图的weight域为1，

(2)带权图的weight为边的权值。

图的遍历

从图G=(V,E)中的任意顶点v出发，访问图G中的所有顶点，每个顶点仅被访问一次。

说明：图的遍历要比树的遍历复杂。

建议：为避免同一顶点被多次重复访问，设立辅助数组visited[] ，初值为0，表示顶点没被访问过，一旦顶点i被访问，将visited[i]设置为1。

深度优先遍历(Depth First Search：DFS)

1、任选一个顶点v，并访问v;

2、从顶点v的邻接点中选择一个没有被访问过的邻接点w，从顶点w出发深度优先遍历图；

3、重复2，直到图中所有与v有路径相通的顶点都被访问。

说明：深度优先遍历类似树的先根遍历

DFS思路：

一步一步向前走，当没有可走的相邻顶点时便回退。



广度优先遍历(Breath First Search：BFS)

1、首先访问初始点vi，并将其访问标记设置为1；

2、接着访问顶点vi所有未被访问过的邻接点 vi1, vi2, …,vit ，并将访问标记均设置为1；

3、再按照 vi1, vi2, …, vit 的次序访问每一个顶点所有未被访问过的邻接点，并将访问标记均设置为1；

4、直到图中所有与初始点vi有路径相通的顶点都被访问。

说明：广度优先遍历类似树的层次遍历

顺序一致：用队列实现

无向图的生成树

无向连通图G的全部顶点和部分边构成的子图G’满足：

(1)子图G’中的所有顶点连通；

(2)子图G’中没有回路；

称子图G’是原图G的一棵生成树。

无向图的生成树：由遍历得到

(1) 连通图

遍历一次连通图所经过的边集、所有顶点的集合就构成了该图的一棵生成树(不唯一)；

(2)非连通图(由多个连通分量构成)

从每个连通分量的任一顶点出发进行遍历，可得该连通分量的顶点集，每个连通分量产生的生成树的集合构成整个非连通图的生成树。

说明：

1. 深度优先生成树：深度优先遍历
2. 广度优先生成树：广度优先遍历产生的生成树。

最小生成树：各边的权之和最小的生成树

是一棵树：

(1)无回路(2) V个顶点有V-1条边。

是生成树：

(1)包含全部顶点(2) V-1条边都在图里。

最小生成树的构造算法

(1) 普里姆算法：构造性算法

(2) 克鲁斯卡尔(Kruskal)算法：

按权值递增次序选择边来构造最小生成树。

普里姆(Prim)算法：构造法

1、假设：

①G=(V,E)是具有n个顶点的带权、无向、连通图；

② T=(U,TE)是G的最小生成树，U是T的顶点集，TE是T的边集。

2、从起始顶点 v构造最小生成树T的步骤：

(1)初始化U={v}。以v到其他顶点的所有边为候选边；

(2)重复以下步骤n-1次，使得其他n-1个顶点并入U中。

普里姆算法关键：在顶点集U、U-V之间选择最小边

1、辅助数组closest[n]

已知j∈V-U，closest[j]为该边在U中的顶点编号；

2、辅助数组lowcost[n]

已知j∈V-U，lowcost[j]存储该边的权值。

① 从候选边中挑选权值最小的边加入TE，设该边在V-U中的顶点是k，将k加入U中；

② 考察当前V-U中的所有顶点j，修改j的候选边：

如果(k,j) < j的候选边，用(k,j)取代j的候选边。

# 排序

排序：将一组无序数据按一定的规律顺序排列起来

排序目的：便于进行查找

排序算法好坏的衡量 标准

时间：排序速度(排序所花费的全部比较次数)

空间：占内存辅助空间大小

稳定性：

若关键字值相等的两个记录A和B，排序后A、B的相对先后次序保持不变，则称这种排序算法是稳定的。

排序种类：内部排序、外部排序

内部排序：待排序数据都在内存

外部排序：待排序记录一部分在内存，一部分在外存

注意：外部排序时，要将数据分批调入内存来排序，中间结果需要及时存入外存。

待排序数据在内存的存储和处理

顺序排序：直接移动记录；

链表排序：只修改指针。

交换排序思想

两两比较待排序记录的关键码，如果发生逆序(即排序前次序与排序后次序正好相反)，则交换之，直到所有记录都排好序为止。

交换排序算法

1、冒泡排序

2、快速排序每趟结束时，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；一旦某一趟没有交换发生，就可以提前结束排序。

3、前提：顺序存储结构

冒泡排序思想

通过对无序序列的相邻数据进行“比较”和位置”交换”，实现小数向“一头”飘浮，大数向“另一头”下沉，从而达到记录按值非递减有序排列的目标。

1. 每趟不断将相邻数据比较，并按“前小后大”(或“前大后小”)规则交换。
2. 优点

直接插入排序思想：

初始状态：

先将序列的第一个数看成有序的子序列，然后从第二个数开始逐个插入排序

直到整个序列有序为止.

折半插入排序思想：

是直接插入排序的改进版，每次从序列的无序部分取出第一个数据，将它与序列的有序部分中间位置(middle)的数据进行比较，如果比中间位置上的数据大 ，则在该数据的右侧继续进行比较，否则在左侧继续进行比较，直到找到合适的插入位置，并且序列的有序部分该位置之后的数据(后移 )。然后将该数据插入到该位置，使得序列的有序部分依然保持有序。

例如：

【12 23 34 56 78】77 45

快速排序算法分析

1、时间效率O(nlog2n)

因为每趟确定的元素呈指数增加

2、空间效率O(log2n)

因为算法的递归性，要用到栈空间

3、稳定性：不稳定

基数排序的基本思想是：

借助多关键字排序的思想对单逻辑关键字进行排序。即：用关键字不同的位值进行排序。

归并排序的基本思想是：将两个（或以上）的有序表组成新的有序表。

快速排序算法分析

1、时间效率O(nlog2n)

因为每趟确定的元素呈指数增加

2、空间效率O(log2n)

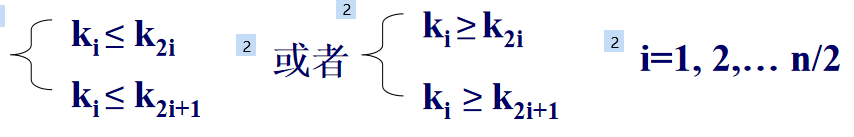
因为算法的递归性，要用到栈空间

3、稳定性：不稳定

因为可选任一元素为支点)。

### 堆

堆的定义：设有n个元素的序列 k1，k2，…，kn，当且仅当满足下述关系之一时，称之为堆。



小根堆 大根堆

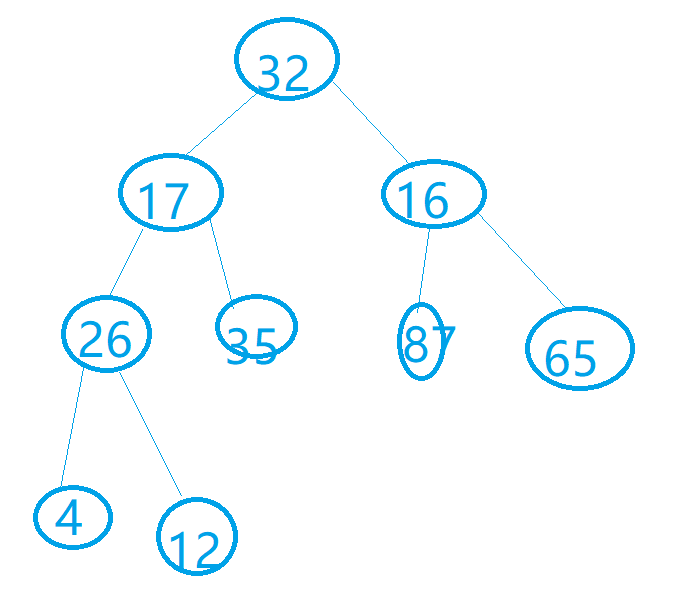
堆排序如果让满足以上条件的元素序列 （k1，k2，…，kn）

顺次排成一棵完全二叉树，则此树的特点是：

树中所有根结点的值均大于（或小于）其左右孩子，

此树的根结点（即堆顶）必最大（或最小）。

建堆：1.按照数据元素顺序构建一个完全二叉树



1. 把此二叉树转换为堆积树

步骤：

数组来存储二叉树的所有节点的值，

a[0]=32,a[1]=17,a[2]=16,a[3]=24,a[4]=35,a[5]=87,a[6]=65,a[7]=4,a[8]=12

第1步：a[0]=32为树根，若a[1]大于父节点必须互换，此处a[1]=17<a[0]=32，故不交换

第2步：a[2]=16<a[0]故不交换

第3步：a[3]=24>a[1]=17,故交换

第4步：a[4]=35>a[1]=24,故交换，再与a[0]=32进行比较，a[1]=35>a[0]=32故交换

堆排序算法思路(大根堆为例)

1. 初始建堆，初始将n个记录的序列建成一个大根堆
2. 输出堆项记录
3. 调整剩余记录，由于在输出大根堆顶记录后，剩余的记录不符合堆得定义，需要把剩余记录重新调整为大根堆

例：101 843 206 156 423 366 624

排序重点：

1. 冒泡排序
2. 直接插入序列
3. 折半插入排序
4. 快速排序
5. 选择排序