## 方法：

### 排序算法：

稳定：如果a原本在b前面，而a=b，排序之后a仍然在b的前面；

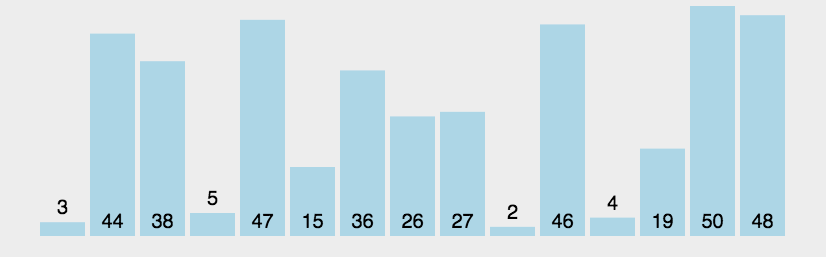
不稳定：如果a原本在b的前面，而a=b，排序之后a可能会出现在b的后面；

In-place: 占用常数内存，不占用额外内存

Out-place: 占用额外内存



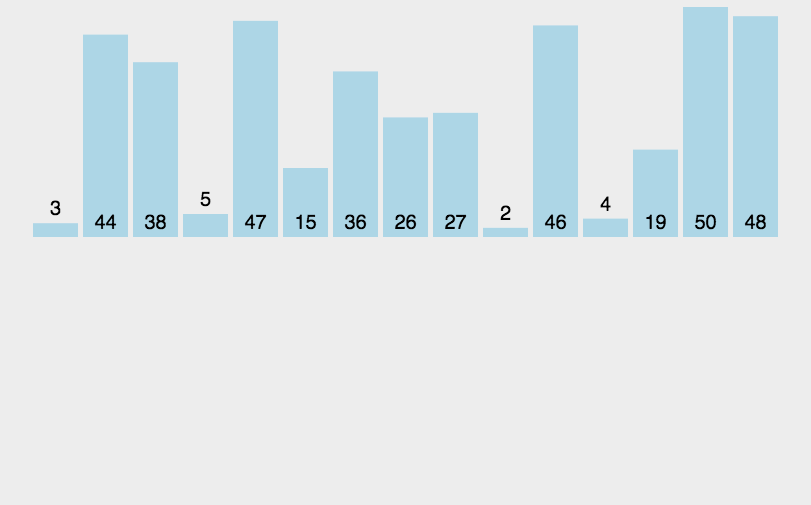
1. 冒泡排序（Bubble Sort）



1. 选择排序（Selection Sort）



3、插入排序（Insertion Sort）

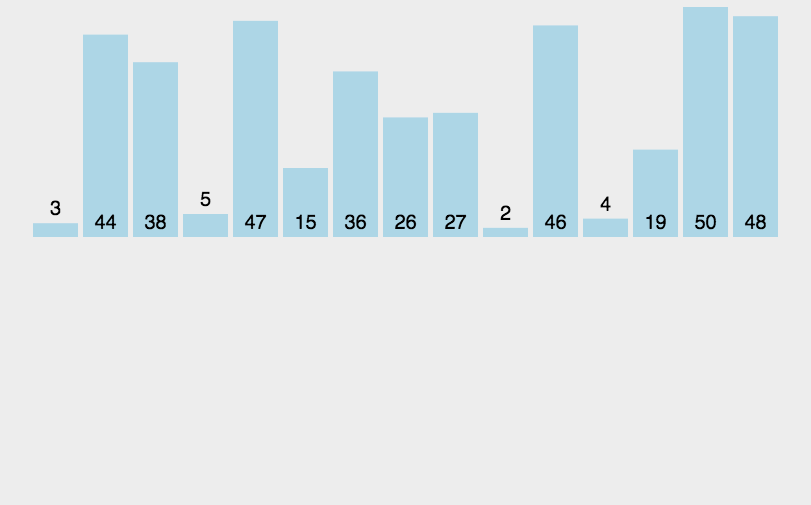


5、归并排序（Merge Sort）

把长度为n的输入序列分成两个长度为n/2的子序列；

对这两个子序列分别采用归并排序；

将两个排序好的子序列合并成一个最终的排序序列。



**6、快速排序（Quick Sort）**

从数列中挑出一个元素，称为 “基准”（pivot）；

重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作；

递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。

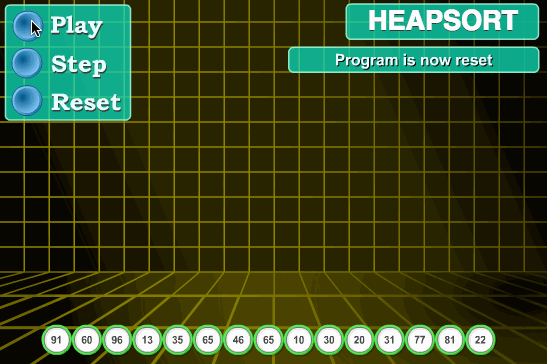


7、堆排序（Heap Sort）

初始待排序关键字序列(R1,R2….Rn)构建成大顶堆，此堆为初始的无序区；

将堆顶元素R[1]与最后一个元素R[n]交换，此时得到新的无序区(R1,R2,……Rn-1)和新的有序区(Rn),且满足R[1,2…n-1]<=R[n]；

由于交换后新的堆顶R[1]可能违反堆的性质，因此需要对当前无序区(R1,R2,……Rn-1)调整为新堆，然后再次将R[1]与无序区最后一个元素交换，得到新的无序区(R1,R2….Rn-2)和新的有序区(Rn-1,Rn)。不断重复此过程直到有序区的元素个数为n-1，则整个排序过程完成。



**添加元素功能**

push(E) 向队列头部插入一个元素,失败时抛出异常

addFirst(E) 向队列头部插入一个元素,失败时抛出异常

addLast(E) 向队列尾部插入一个元素,失败时抛出异常

offerFirst(E) 向队列头部加入一个元素,失败时返回false

offerLast(E) 向队列尾部加入一个元素,失败时返回false

**获取元素功能**

getFirst() 获取队列头部元素,队列为空时抛出异常

getLast() 获取队列尾部元素,队列为空时抛出异常

peekFirst() 获取队列头部元素,队列为空时返回null

peekLast() 获取队列尾部元素,队列为空时返回null

**删除功能**

removeFirstOccurrence(Object) 删除第一次出现的指定元素,不存在时返回false

removeLastOccurrence(Object) 删除最后一次出现的指定元素,不存在时返回false

**弹出功能**

pop() 弹出队列头部元素,队列为空时抛出异常

removeFirst() 弹出队列头部元素,队列为空时抛出异常

removeLast() 弹出队列尾部元素,队列为空时抛出异常

pollFirst() 弹出队列头部元素,队列为空时返回null

pollLast() 弹出队列尾部元素,队列为空时返回null

LinkedList

Stack

二分查找树

HashSet

ArrayList

Stringbuffer:可添加字符的字符串

“线性表(Linear List)”:由同类型数据元素构成有序序列的线性结构

>表中元素个数称为线性表的长度

>线性表没有元素时，称为空表

>表起始位置称表头，表结束位置称表尾

1、List MakeEmpty():初始化-一个空线性表L;

2、ElementType FindKth( int K, ListL ):根据位序K，返回相应元素;

3、int Find( ElementType X, ListL ):在线性表L中查找X的第- 次出现位置:

4、void Insert( ElementType X, int i, ListL):在位序i前插入一个新元素X;

5、void Delete(int i, ListL ):删除指定位序i的元素;

6、int Length( ListL ):返回线性表L的长度n.

使用链表结构：链表中每个结点存储多项式中的一个非零项，包括系数和指数两个数据域以及一个指针域

例：typedef struct PolyNode \*Polynomial;

Stryct PolyNode{

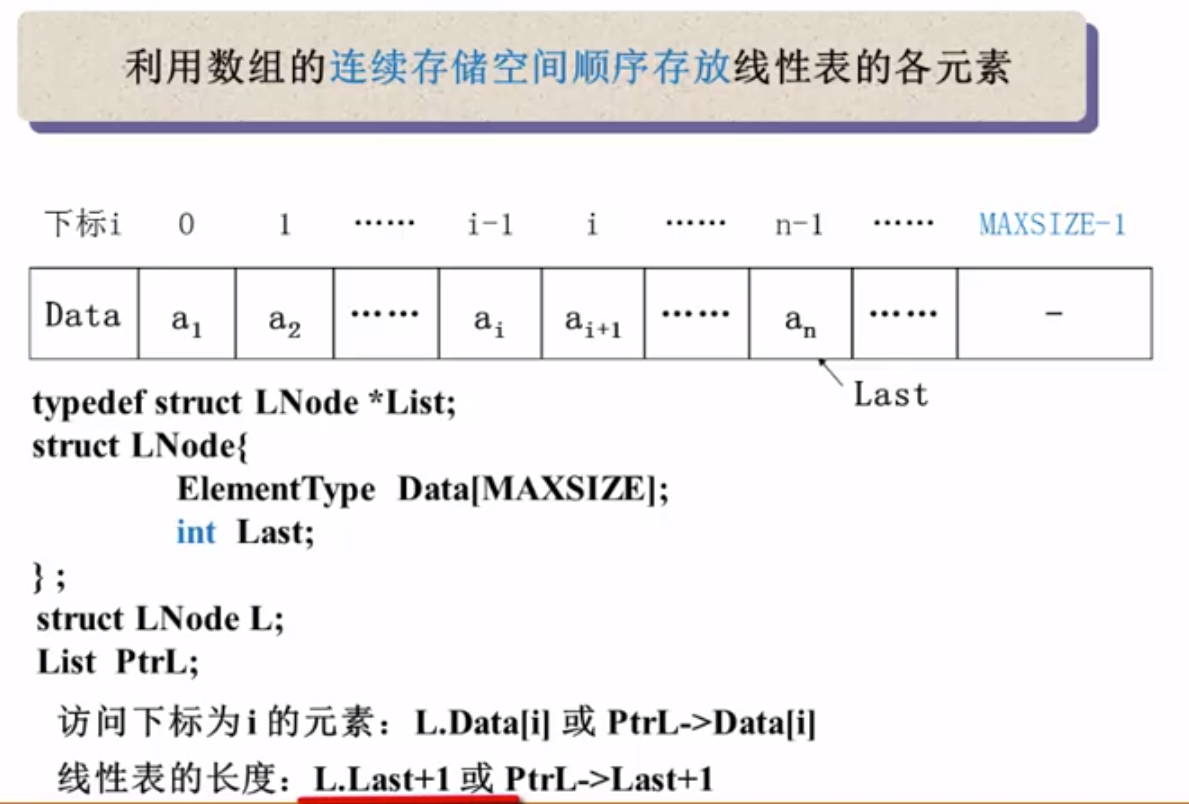
int coef;

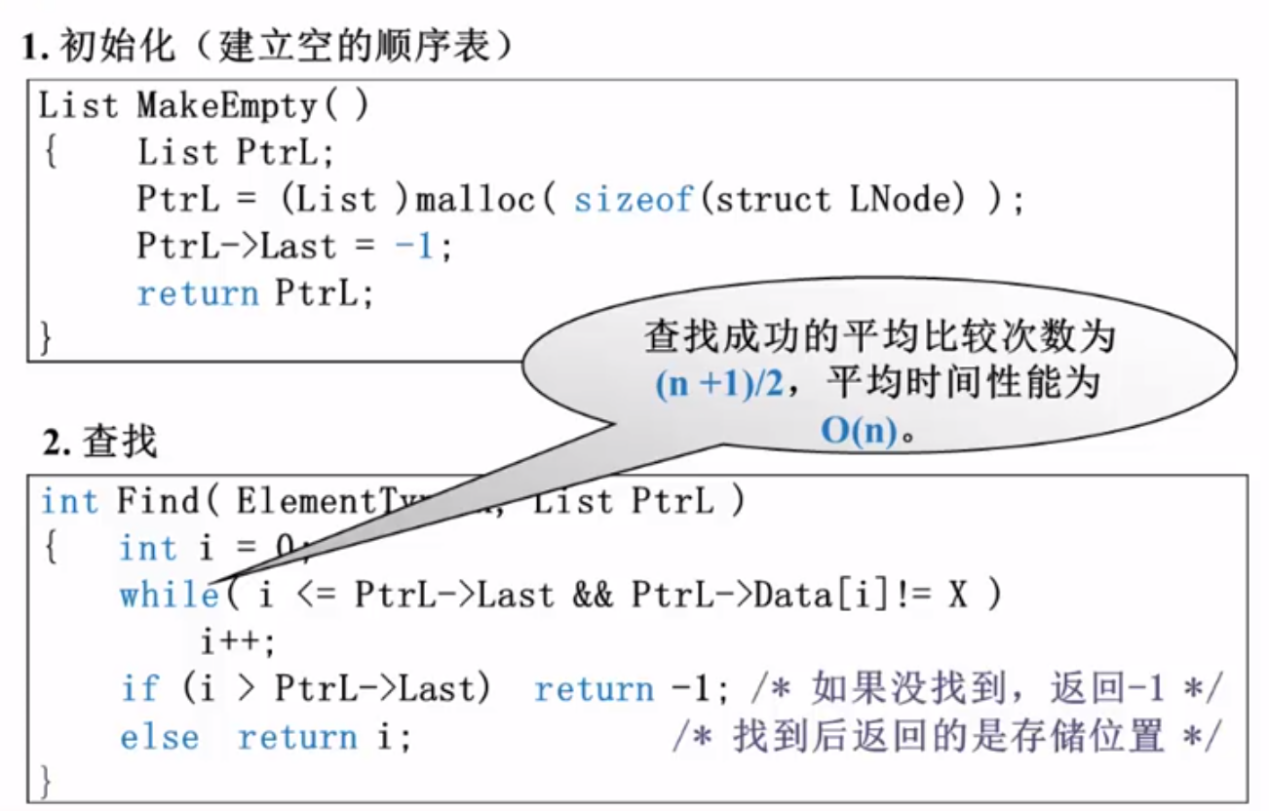
int expon

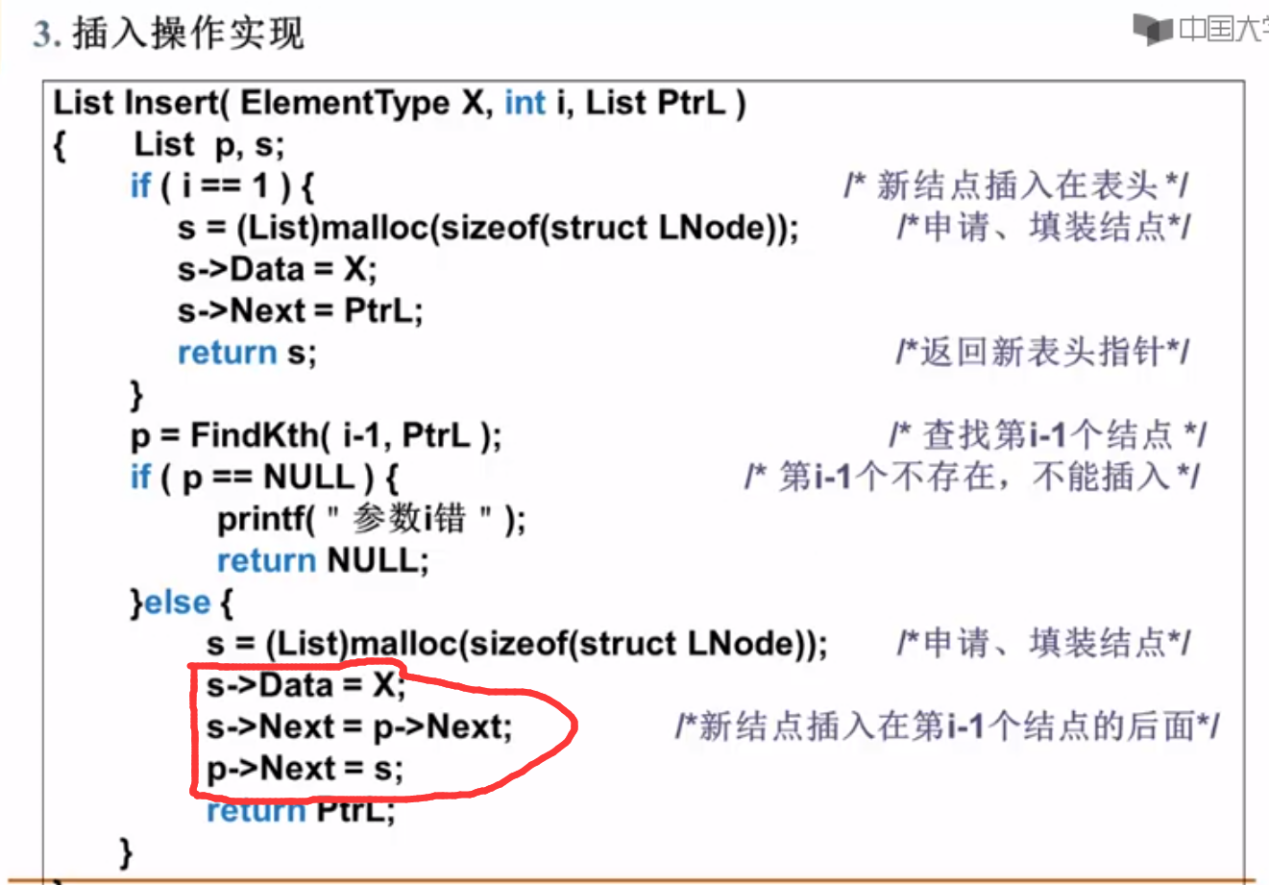
Polynomial link;

}

Node //节点







注意插入时指针位置受语局的先后顺序影响

## 算法思维

**动态规划**：让不确定的因素定下来，即假设该因素恰好满足条件，再反推前面的因素

动态规划算法的核心就是记住已经解决过的子问题的解

**回溯法**：一种通过探索所有可能的候选解来找出所有的解的算法。如果候选解被确认不是一个解（或者至少不是最后一个解），回溯算法会通过在上一步进行一些变化抛弃该解，即回溯并且再次尝试

**并查集**：并查集是一种树型的数据结构，用于处理一些不相交集（Disjoint Sets）的合并及查询问题。对于并查集，主要有如下操作：

合并两个集合（“并”）构建一个union方法：通过指定同一个父节点将两个合集并在一起，

判断两个元素是否属于同一个集合。（“查”）：查询两个的根节点是否相同

**递归：递归函数必须要有终止条件，否则会出错；**

**递归函数先不断调用自身，直到遇到终止条件后进行回溯，最终返回答案。**

**链表：闭合为环和快慢指针，以及别怕新建变量过多，伪头节点**

### TIPS:

#### StringBuffer

线程安全，比StringBuilder慢

用法：StringBuffer sb = new StringBuffer(10);定义一个初始容量为10的字符串

sb.append(String); 把string接在后面；

sb.insert(index,String)；把String插入指定位置，原位置之后的向后移动；

sb.delete(index1,inde2); 删除index1,到index2位置的字符，左闭右开；

sb.reverse(); 反转字符串；

sb.replace(index1,index2,String); 用String的字符取代index1,到index2位置的字符，左闭右开