数据结构

## 哈希表

在记录的存储位置和他的关键字之间建立一个确定的对应关系f，使得每个关键字和结构中一个唯一的存储位置对应，f叫哈希函数，按这个思想建立的表为哈希表。

### 1.1、哈希函数构造方法

**A直接定址法**

H(key)=key或H(key)=a\*key+b

直接定址法所得地址集合和关键字集合大小相同，因此对于不同的关键字不会产生冲突，但是实际中能使用这种哈希函数的情况很少。

**B数字分析法**

**C平方取中法**

取关键字平方后的中间几位为哈希地址

**D折叠法**

将关键字分割成位数相同的几部分（最后一部分位数可以不同），然后取这几部分的叠加和（舍去进位）作为哈希地址。

**E除留余数法**

取关键字被某个不大于哈希表表长m的数p除后所得的余数作为哈希地址

H（key）=key MOD p ,p<=m (p的选择很重要)一般情况下，可以选p为质数或者不包含小于20的质因数的合数。

**F随机数法**

选择一个随机函数，取关键字的随机函数值为他的哈希地址。H(key)=random（key）

### 1.2 处理冲突的方法

**A 开放定址法**

HI=(H(key)+di) MOD m i=1，2，……，k（k<=m-1）

H(key)为哈希函数m为哈希表表长，di为增量序列，有3种取法：线性探测再散列，二次探测再散列，伪随机数序列。

线性探测再散列：只要哈希表未填满，总能找到一个不发生冲突的地址，但可能发生“二次聚集”；

**B 再哈希法**

在同义词产生地址冲突时计算另一个哈希函数地址，不易产生“聚集”，但是增加了计算时间。

**C 链地址法**

将所有关键字为同义词的记录存储在同一线性链表中。

**D 建立公共溢出区**

将发生冲突的词填入溢出表

## 2 排序算法

### 2.1算法总结



n: 数据规模

k: “桶”的个数

In-place: 占用常数内存，不占用额外内存

Out-place: 占用额外内存

### 2.2算法分类

### 2.3 插入排序

插入排序（Insertion-Sort）的算法描述是一种简单直观的排序算法。它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。插入排序在实现上，通常采用in-place排序（即只需用到O(1)的额外空间的排序），因而在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间。

2.3.1 算法描述

一般来说，插入排序都采用in-place在数组上实现。具体算法描述如下：

从第一个元素开始，该元素可以认为已经被排序；

取出下一个元素，在已经排序的元素序列中从后向前扫描；

如果该元素（已排序）大于新元素，将该元素移到下一位置；

重复步骤3，直到找到已排序的元素小于或者等于新元素的位置；

将新元素插入到该位置后；

重复步骤2~5。

### 2.4 希尔排序

希尔排序又叫缩小增量排序。希尔排序的核心在于间隔序列的设定。既可以提前设定好间隔序列，也可以动态的定义间隔序列。动态定义间隔序列的算法是《算法（第4版）的合著者Robert Sedgewick提出的。

先将整个待排序的记录序列分割成为若干子序列分别进行直接插入排序，具体算法描述：

2.4.1 算法描述

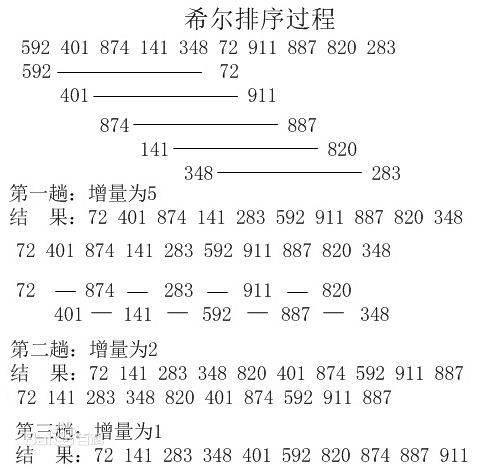
先将整个待排序的记录序列分割成为若干子序列分别进行直接插入排序，具体算法描述：

选择一个增量序列t1，t2，…，tk，其中ti>tj，tk=1；

按增量序列个数k，对序列进行k 趟排序；

每趟排序，根据对应的增量ti，将待排序列分割成若干长度为m 的子序列，分别对各子表进行直接插入排序。仅增量因子为1 时，整个序列作为一个表来处理，表长度即为整个序列的长度。

2.4.2 过程演示



### 2.5 选择排序

选择排序(Selection-sort)是一种简单直观的排序算法。它的工作原理：首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。

2.5.1 算法描述

n个记录的直接选择排序可经过n-1趟直接选择排序得到有序结果。具体算法描述如下：

初始状态：无序区为R[1..n]，有序区为空；

第i趟排序(i=1,2,3…n-1)开始时，当前有序区和无序区分别为R[1..i-1]和R(i..n）。该趟排序从当前无序区中-选出关键字最小的记录 R[k]，将它与无序区的第1个记录R交换，使R[1..i]和R[i+1..n)分别变为记录个数增加1个的新有序区和记录个数减少1个的新无序区；

n-1趟结束，数组有序化了。

### 2.6堆排序

堆排序（Heapsort）是指利用堆这种数据结构所设计的一种排序算法。堆积是一个近似完全二叉树的结构，并同时满足堆积的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。

2.7.1 算法描述

将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。

### 2.7 冒泡排序

冒泡排序是一种简单的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果它们的顺序错误就把它们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端。

2.7.1 算法描述

比较相邻的元素。如果第一个比第二个大，就交换它们两个；

对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对，这样在最后的元素应该会是最大的数；

针对所有的元素重复以上的步骤，除了最后一个；

重复步骤1~3，直到排序完成。

### 2.8快速排序

快速排序的基本思想：通过一趟排序将待排记录分隔成独立的两部分，其中一部分记录的关键字均比另一部分的关键字小，则可分别对这两部分记录继续进行排序，以达到整个序列有序。

2.8.1 算法描述

快速排序使用分治法来把一个串（list）分为两个子串（sub-lists）。具体算法描述如下：

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。

### 2.9 归并排序

归并排序是建立在归并操作上的一种有效的排序算法。该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。归并排序是一种稳定的排序方法。将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为2-路归并。

2.9.1 算法描述

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。

### 2.10 基数排序

基数排序是按照低位先排序，然后收集；再按照高位排序，然后再收集；依次类推，直到最高位。有时候有些属性是有优先级顺序的，先按低优先级排序，再按高优先级排序。最后的次序就是高优先级高的在前，高优先级相同的低优先级高的在前。基数排序基于分别排序，分别收集，所以是稳定的。

2.10.1 算法描述

取得数组中的最大数，并取得位数；

arr为原始数组，从最低位开始取每个位组成radix数组；

对radix进行计数排序（利用计数排序适用于小范围数的特点）；