# 一、算法基础知识

http://blog.csdn.net/happy\_wu/article/details/51841244

## 1. 算法的效率

(1) 时间复杂度：执行程序所需的时间。可以估算出程序对处理器的使用程度。

(2) 空间复杂度：评估执行程序所需的存储空间。可以估算出程序对计算机内存的使用程度。

(3) 设计算法时，一般是要先考虑系统环境，然后权衡时间复杂度和空间复杂度，选取一个平衡点。不过，时间复杂度要比空间复杂度更容易产生问题，因此算法研究的主要也是时间复杂度，不特别说明的情况下，复杂度就是指时间复杂度。

## 2. 时间复杂度

(1) 时间频度

**一个算法花费的时间与算法中语句的执行次数成正比例**，哪个算法中语句执行次数多，它花费时间就多。一个算法中的语句执行次数称为语句频度或时间频度。记为T(n)。

(2) 时间复杂度

时间频度T(n)中，n称为问题的规模，当n不断变化时，时间频度T(n)也会不断变化。但有时我们想知道它变化时呈现什么规律，为此我们引入时间复杂度的概念。一般情况下，算法中基本操作重复执行的次数是问题规模n的某个函数，用T(n)表示，若有某个辅助函数f(n)，使得当n趋近于无穷大时，T（n)/f(n)的极限值为不等于零的常数，则称f(n)是T(n)的同数量级函数，记作T(n)=O(f(n))，它称为算法的渐进时间复杂度，简称时间复杂度。

## 3. 大O表示法

O( )来体现算法时间复杂度的记法，称之为大O表示法。

(1) **算法复杂度可以从最理想情况、平均情况和最坏情况三个角度来评估**，由于平均情况大多和最坏情况持平，而且评估最坏情况也可以避免后顾之忧，因此一般情况下，我们设计算法时都要直接估算最坏情况的复杂度。

(2) 大O表示法O(f(n)中的f(n)的值可以为1、n、logn、n²等，因此我们可以将O(1)、O(n)、O(logn)、O(n²)分别可以称为常数阶、线性阶、对数阶和平方阶，

### 3.1 推导大O阶

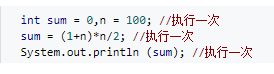
(1) 用常数1来取代运行时间中所有加法常数。

(2) 修改后的运行次数函数中，只保留最高阶项

(3) 如果最高阶项存在且不是1，则去除与这个项相乘的常数。

### 3.2 常数阶

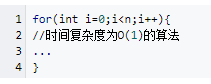
(1) 实例



(2) 算法的运行的次数的函数为f(n)=3，根据推导大O阶的规则1，将常数3改为1，则这个算法的时间复杂度为O(1)。即使将代码再执行10遍，因为这与问题大小n的值并没有关系，所以这个算法的时间复杂度仍旧是O(1)，可以称之为常数阶。

### 3.3 线性阶

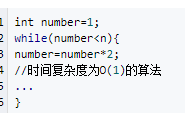
(1) **线性阶主要要分析循环结构的运行情况**

//线性递增，循环了n次

(2) 上面算法循环体中的代码执行了n次，因此时间复杂度为O(n)。

### 3.4 对数阶

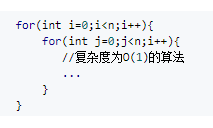
(1) 实例

//乘数递增，循环了x次

(2) 可以看出上面的代码，随着number每次乘以2后，都会越来越接近n，当number不小于n时就会退出循环。假设循环的次数为X，则由2^x=n得出x=log₂n，因此得出这个算法的时间复杂度为O(logn)。

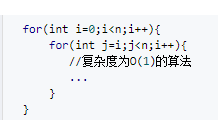
### 3.5 平方阶

(1) 循环嵌套



(2) 内层循环的时间复杂度在讲到线性阶时就已经得知是O(n)，现在经过外层循环n次，那么这段算法的时间复杂度则为O(n²)。

### 3.6 实例



(1) 需要注意的是内循环中int j=i，而不是int j=0。当i=0时，内循环执行了n次；i=1时内循环执行了n-1次，当i=n-1时执行了1次，我们可以推算出总的执行次数为：

n+(n-1)+(n-2)+(n-3)+……+1

=(n+1)+[(n-1)+2]+[(n-2)+3]+[(n-3)+4]+……

=(n+1)+(n+1)+(n+1)+(n+1)+……

=(n+1)n/2

=n(n+1)/2

=n²/2+n/2

(2) 根据此前讲过的推导大O阶的规则的第二条：只保留最高阶，因此保留n²/2。根据第三条去掉和这个项的常数，则去掉1/2,最终这段代码的时间复杂度为O(n²)。

## 4. 其他常见复杂度

(1) 除了常数阶、线性阶、平方阶、对数阶，还有如下时间复杂度：

f(n)=nlogn时，时间复杂度为O(nlogn)，可以称为nlogn阶。

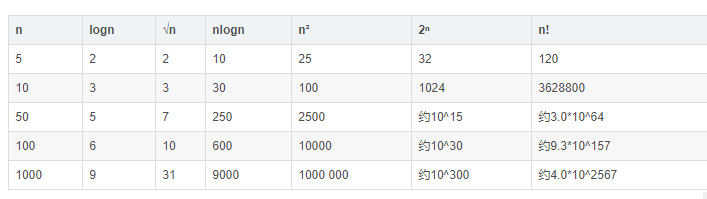
f(n)=n³时，时间复杂度为O(n³)，可以称为立方阶。

f(n)=2ⁿ时，时间复杂度为O(2ⁿ)，可以称为指数阶。

f(n)=n!时，时间复杂度为O(n!)，可以称为阶乘阶。

f(n)=(√n时，时间复杂度为O(√n)，可以称为平方根阶。

(2) 复杂度比较



从上表可以看出，O(n)、O(logn)、O(√n )、O(nlogn )随着n的增加，复杂度提升不大，因此这些复杂度属于效率高的算法，反观O(2ⁿ)和O(n!)当n增加到50时，复杂度就突破十位数了，这种效率极差的复杂度最好不要出现在程序中，因此在动手编程时要评估所写算法的最坏情况的复杂度。

# 二、八大排序算法

## 简介



(1) 当n较大，则应采用时间复杂度为O(nlog2n)的排序方法：快速排序、堆排序或归并排序。

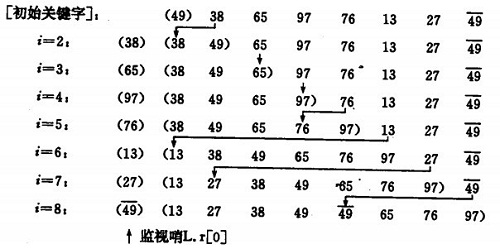
(2) 快速排序：是目前基于比较的内部排序中被认为是最好的方法，当待排序的关键字是随机分布时，快速排序的平均时间最短；

## 1. 插入排序—直接插入排序(Straight Insertion Sort)

(1) 基本思想（从小到大）

取出数组第i个值，和前面的值逐个比较。如果大于该值，则向右移动一位，记录索引位置。直到小于等于该值，则将这个值插入在记录的位置。相等元素排序前后位置不发生改变，所以是稳定的。通俗地讲就是能保证排序前2个相等的数其在序列的前后位置顺序和排序后它们两个的前后位置顺序相同。在简单形式化一下，如果Ai = Aj，Ai原来在位置前，排序后Ai还是要在Aj位置前。

1. 要点：设立哨兵，作为临时存储和判断数组边界之用。



② **效率：时间复杂度：O（n^2） 稳定的**

## 2. 插入排序—希尔排序（Shell`s Sort）（缩小增量排序）

(1) 基本思想

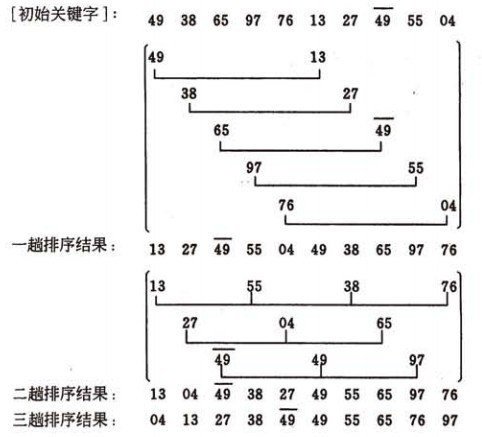
记录一个dk值，从i开始增量为dk进行插入序。然后再缩小增量，继续进行插入排序。最后增量为1就是普通的直接插入排序。总的来说，就是先大跨度的调整顺序，再慢慢缩小增量至1排序，相对于直接插入排序会快一点。

(2) 操作方法

① 选择一个增量序列t1，t2，…，tk，其中ti>tj，tk=1；

② 按增量序列个数k，对序列进行k 趟排序；

③ 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。



(3) 希尔排序时效分析很难，关键码的比较次数与记录移动次数依赖于增量因子序列d的选取，特定情况下可以准确估算出关键码的比较次数和记录的移动次数。目前还没有人给出选取最好的增量因子序列的方法。增量因子序列可以有各种取法，有取奇数的，也有取质数的，但需要注意：增量因子中除1 外没有公因子，且最后一个增量因子必须为1。**希尔排序方法是一个不稳定的排序方法。**

## 3. 选择排序—简单选择排序（Simple Selection Sort）

(1) 基本思想

在要排序的一组数中，选出最小（或者最大）的一个数与第1个位置的数交换；然后在剩下的数当中再找最小（或者最大）的与第2个位置的数交换，依次类推，直到第n-1个元素（倒数第二个数）和第n个元素（最后一个数）比较为止。

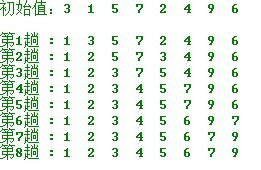
(2) 操作方法

① 第一趟，从n 个记录中找出关键码最小的记录与第一个记录交换；

② 第二趟，从第二个记录开始的n-1 个记录中再选出关键码最小的记录与第二个记录交换；以此类推.....

③ 第i 趟，则从第i 个记录开始的n-i+1 个记录中选出关键码最小的记录与第i 个记录交换，直到整个序列按关键码有序。

(3) 实例



(4) 时间复杂度是O(n^2)，不稳定排序。

## 4. 选择排序—堆排序（Heap Sort）

(1) 基本思想

初始时把要排序的n个数的序列看作是一棵顺序存储的二叉树（一维数组存储二叉树），调整它们的存储序，使之成为一个堆，将堆顶元素输出，得到n 个元素中最小(或最大)的元素，这时堆的根节点的数最小（或者最大）。然后对前面(n-1)个元素重新调整使之成为堆，输出堆顶元素，得到n 个元素中次小(或次大)的元素。依此类推，直到只有两个节点的堆，并对它们作交换，最后得到有n个节点的有序序列。称这个过程为堆排序。

(2) 因此，实现堆排序需解决两个问题：

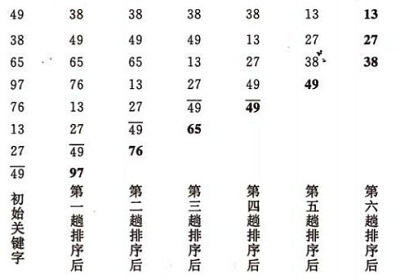
① 如何将n 个待排序的数建成堆；

② 输出堆顶元素后，怎样调整剩余n-1 个元素，使其成为一个新堆。

## 5. 交换排序—冒泡排序（Bubble Sort）

(1) 基本思想

在要排序的一组数中，对当前还未排好序的范围内的全部数，自上而下对相邻的两个数依次进行比较和调整，让较大的数往下沉，较小的往上冒。即：每当两相邻的数比较后发现它们的排序与排序要求相反时，就将它们互换。



(2) 时间复杂度O(n²)，稳定排序

## 6. 交换排序—快速排序（Quick Sort）

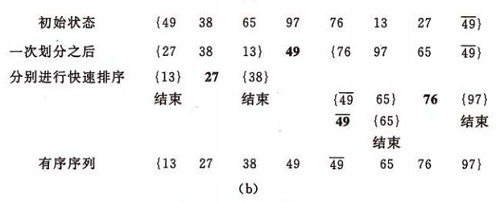
(1) 基本思想

① 选择一个基准元素,通常选择第一个元素或者最后一个元素,

② 通过一趟排序讲待排序的记录分 割成独立的两部分，其中一部分记录的元素值均比基准元素值小。另一部分记录的 元素值比基准值大。

③ 此时基准元素在其排好序后的正确位置

④ 然后分别对这两部分记录用同样的方法继续进行排序，直到整个序列有序。



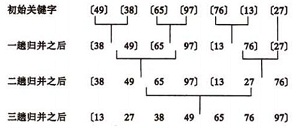
(2) 快速排序是通常被认为在同数量级（O(nlog2n)）的排序方法中平均性能最好的。但若初始序列按关键码有序或基本有序时，快排序反而蜕化为冒泡排序。

(3) 快速排序是一个不稳定的排序方法。时间复杂度是O (nlogn)

## 7. 归并排序

(1) 基本思想

归并（Merge）排序法是将两个（或两个以上）有序表合并成一个新的有序表，即把待排序序列分为若干个子序列，每个子序列是有序的。然后再把有序子序列合并为整体有序序列。



(2) 合并方法：

设r[i…n]由两个有序子表r[i…m]和r[m+1…n]组成，两个子表长度分别为n-i +1、n-m。

① j=m+1；k=i；i=i; //置两个子表的起始下标及辅助数组的起始下标

② 若i>m 或j>n，转⑷ //其中一个子表已合并完，比较选取结束

③ //选取r[i]和r[j]较小的存入辅助数组rf

如果r[i]<r[j]，rf[k]=r[i]； i++； k++； 转⑵

否则，rf[k]=r[j]； j++； k++； 转⑵

④ //将尚未处理完的子表中元素存入rf

如果i<=m，将r[i…m]存入rf[k…n] //前一子表非空

如果j<=n , 将r[j…n] 存入rf[k…n] //后一子表非空

合并结束。

(3) 分组规则，先左（left到center）再右（ceter+1 到right），再把各自分组排序好的小组合并起来，合并的时候是分别从左组和右组一个一个挑选，保证了有序性。

(4) 时间复杂度是O(n log n)，是稳定排序。

## 8. 桶排序/基数排序(Radix Sort) -- 放弃不看

## 9. 总结

9.1 各种排序的稳定性，时间复杂度和空间复杂度总结：



注意：对n较大的排序记录。一般的选择都是时间复杂度为O(nlog2n)的排序方法。

9.2 时间复杂度

(1) 平方阶(O(n2))排序：各类简单排序:直接插入、直接选择和冒泡排序；

(2) 线性对数阶(O(nlog2n))排序：快速排序、堆排序和归并排序；最重要

(3) O(n1+§))排序,§是介于0和1之间的常数。：希尔排序

(4) 线性阶(O(n))排序：基数排序，此外还有桶、箱排序。

说明：

(1) 当原表有序或基本有序时，直接插入排序和冒泡排序将大大减少比较次数和移动记录的次数，时间复杂度可降至O（n）；

(2) 而快速排序则相反，当原表基本有序时，将蜕化为冒泡排序，时间复杂度提高为O（n2）；

(3) 原表是否有序，对简单选择排序、堆排序、归并排序和基数排序的时间复杂度影响不大

平衡二叉树的插入时间复杂度是 O(logN)