W15/16笔记

算法分析

排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	稳定性
选择排序	O(n^2)	O(n^2)	O(n^2)	不稳定
冒泡排序	O(n^2)	O(n^2)	O(n)	稳定
插入排序	O(n^2)	O(n^2)	O(n)	稳定
归并排序	O(nlog2n)	O(nlog2n)	O(nlog2n)	稳定
快速排序	O(nlog2n)	O(n^2)	O(nlog2n)	不稳定
希尔排序	O(nlog2n)	O(n^2)	O(n^1.3)	不稳定

选择排序

Selection sort takes N steps to fill the first position, N-1 steps to fill the second, and so on, the total running time is proportional to N + N-1 + N-2 +... + 3 + 2 +1

冒泡排序

- 最好情况：待排序数组本身就是正序的，一轮即可完成排序，此时时间复杂度仅为比较时的时间复杂度，为O(n)
- 最坏情况：要进行n-1轮排序循环，每轮排序循环中序列都是非正序的，则每轮排序循环需要进行n-i次比较，最坏时间复杂度为O(n^2)
- 平均情况：时间复杂度为O(n^2)

插入排序

- 最好情况：数组本身已经有序，只需进行n-1次比较，时间复杂度为O(n)
- 最坏情况：数组逆序排序，此时需要进行n(n-1)/2次比较和n-1次赋值，时间复杂度为O(n^2)
- 平均情况：时间复杂度为O(n^2)

归并排序

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。归并排序中第二步，对两个有序数组排序法则非常简单，同时对两个数组的第一个位置比较大小，将小的放入一个空数组，然后被放入空数组的那个位置的指针往后移一个，然后继续和另一个数组的上一个位置进行比较，以此类推。直到最后任何一个数组先出栈完，就将另外一个数组里的所有元素追加到新数组后面。

快速排序

- 最好情况：每轮划分都将待排序列正好分为两部分，那么每部分需要的时间为上一轮的1/2。如果排序n个元素的序列，其递归树深度为[log2n]+1即仅需递归log2n次，需要总时间为T(n)的话，第一次需要扫描整个序列，做n次比较，然后将序列一分为二，这两部分各自还需要T(n/2)的时间，依次划分下去：T(n) = 2*T(n/2)+n T(n) = 2*(2*(T(n/4)+n/2)+n = 4*T(n/4)+2n 等等，且T(1) = 0，所以T(n) = n*T(1)* + *n*log2n = O(nlog2n)

- 最坏情况：当待排序列为有序序列(正序或倒序)，每次划分后得到的情况是一侧有1个元素，另一侧是其余元素，则最终要进行n-1轮循环，且第i次循环要进行n-i次比较，总比较次数为n-1 + n-2 + ... + 1 = n(n-1)/2，即时间复杂度为O(n2)
- 平均情况：时间复杂度为O(nlog2n)

归并和快排的区别

共同点：都是分治法的应用。

```
1. 分解（divide）：将原问题分解为一系列子问题；

2. 解决（conquer）：递归地解各子问题。若子问题足够小，则直接求解；

3. 合并：将子问题的结果合并为原问题的解。
```

不同点：

```
快速排序--自顶向下---先受罪后享福

1. 分解：数组A[p..r]被划分为两个（可能空）子数组；

2. 解决：通过递归调用快速排序，对子数组A[p..q-1]和A[q+1..r]排序；

3. 合并：因为两个子数组是就地排序的，将它们的合并并不需要操作，整个A[p..r]已排序。

归并排序--自底向上---先享福后受罪

归并排序算法完全依照分治模式，直观的操作如下：

1. 分解：将n个元素分成各含 n/2 个元素的子序列；

2. 解决：用归并排序法对两个子序列递归地排序；

3. 合并两个已排序的子序列以得到排序结果。
```

稳定性

排序算法**稳定性**：假设待排序序列中有两个元素相等，而且在排序前和排序后两个相等的元素的相对位置不变，即有 a = b，排序前a在b前面，那么排序后，a还是要在b前面。

不稳定的几种排序：

- 选择排序： A = {4， **4**， 2， 5},排序后 A = {2，**4**， 4， 5}。
- 快速排序：A = {2， **2**， 1}，排序后A = {1，**2**，2}。
- 希尔排序：A = {1，2，5，4，**4**，7}，排序后（k = 2）；A = {1， 2， **4**， 4， 5， 7} 。

结构化程序设计

早期程序

早期的计算机存储器容量非常小，人们设计程序时首先考虑的问题是如何减少存储器开销，硬件的限制不容许人们考虑如何组织数据与逻辑，程序本身短小，逻辑简单，也无需人们考虑程序设计方法问题。

结构化程序

三种基本结构：顺序、选择、循环

嵌套连结成具有复杂层次的“结构化程序”，严格控制`goto`语句的使用。

以控制结构为单位，入口、出口单一，可以从上到下阅读程序。

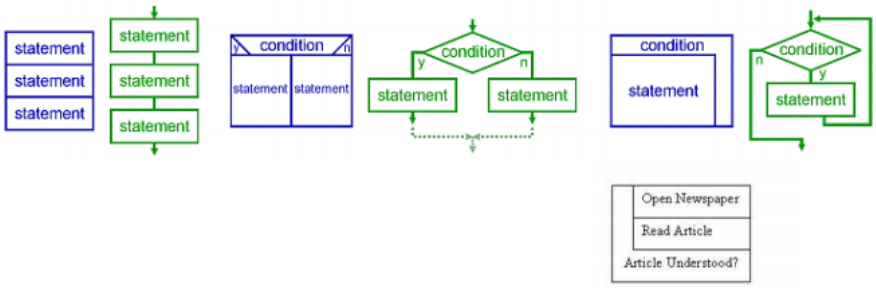
```
int a,b;
int min;

scanf("%d %d", &a, &b);
if ( a<b ) {
    min = a;
} else {
    min = b;
}
int ret = 0;
int i;
for ( i = 1; i < min; i++ ) {
    if ( a%i == 0 ) {
        if ( b%i == 0 ) {
            ret = i;
        }
    }
}
printf("%d和%d的最大公约数是%d.\n", a, b, ret);
```

代码段：以一条流程控制语句（需要被看作一个整体）为基础，包含前后做准备和处理的代码的几条代码。

一个函数是由若干个结构（类似代码段这样的小结构）组成的。

流程图

Nassi-Shneiderman图



面向对象方法

- 建筑在结构化方法之上
- 在一个更大的粒度上理解问题和程序，结构化是其中对具体动作的组织方式