**算 法**

是指令的集合，是为解决特定问题而规定的一系列操作

有明确定义的可计算过程，以一个数据集合作为输入，并产生一个数据集合作为输出

**特征**

1）输入:一个算法应以待解决的问题的信息作为输入

2）输出:输入对应指令集处理后得到的消息（结果集）

3）可行性：算法是可行的，即算法中的每一条指令都是可以实现的

4）有穷性：算法执行的指令个数是有限的，每个指令又是在有限时间内完成的，因此整个算法也是在有限时间内可以结束的

5）确定性:算法对于特定的合法输入，其对应的输出是唯一的，即当算法从一个特定输入开始，多次执行同一指令集结果总是相同的

举例：如何求1+2+3+...+100=?

算法1：依次相加 for while do…while

算法2：高斯算法（1+100）\*100/2

算法3：递归算法sum(100)=sum(99)+100

Sum(99)=sum(98)+99

……

Sum(1)=1

**评价算法优劣的依据：复杂度（时间复杂度和空间复杂度）**

时间复杂度是指执行算法所需要的计算工作量

是一个算法执行的规模

空间复杂度是指算法需要消耗的内存空间

**时间复杂度**

一个算法花费的时间与算法中语句的执行次数成正比，

**时间频度：**

一个算法中的语句执行次数被称为语句频度和时间频度，表示为T（n）（Time）,n表示问题的规模

但是在实际的开发过程中我们其实很少去使用时间频度这个概念，为什么：

比如：某几个算法的时间频度是：

T（n）=100000n2+10n+6

            T（n）=10n2+10n+6

            T（n）=n2

时间复杂度

有些情况下我们想知道算法实现的规模，而不是具体的次数，此时引入时间复杂度

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

             T（n）=O(n2）时间复杂度

**时间复杂度是一个算法执行的规模,或者可以说时间复杂度就是时间频度去掉低阶项和首项常数**

最坏时间复杂度和平均时间复杂度

最坏时间复杂度:

最坏情况下的时间复杂度称为最坏时间复杂度

这样做的原因是：最坏情况下的时间复杂度是算法在任何输入实例上运行时间的上限，这就保证了算法的运行时间不会比任何情况下的时间更长

最坏情况下的时间复杂度：

 T（n）=O(n），表示对于任何输入实例，该算法的运行时间不可能大于O（n）

平均时间复杂度:

所有可能输入实例均以等概率出现的情况下，算法的期望运行时间

鉴于平均复杂度1.难计算   2.有很多算法的平均情况和最差情况的复杂度是一样的

所以一般只讨论最坏时间复杂度

时间复杂度的符号定义：

O（欧米可荣）符号给出了算法时间复杂度的上界（最坏情况<=）,比如 T（n）=O(n2）

Ω（欧米伽）符号给出了时间复杂度的下限（最好情况>=）,比如 T（n）=Ω(n2）

θ（西塔）给出了算法时间复杂度的精确阶（最好最坏是同一个阶），比如T（n）=θ(n2）

时间复杂度计算：

根本没有必要计算时间频度，即使计算处理还要忽略常量、低次幂和最高次幂的系数，所以可以采用如下简单方法：

1）找出算法中的基本语句：

             算法中执行次数最多的那条语句就是基本语句，通常是最内层循环的循环体

2）计算基本语句的执行次数的数量级

     只需要计算基本语句执行次数的数量级，就意味着只要保证基本语句执行次数的函数中的最高次幂正确即可

     可以忽略所有低次幂的最高次幂的系数，能够简化算法分析，并且使注意力集中在最重要的一点：增长率

3）用大O记号表示算法的时间性能

              将基本语句执行次数的数量级放入大O记号中

时间复杂度计算举例：

1.一个简单语句的时间复杂度为O（1）

        int   count=0;

T（n）=1 T(n)=O(1)

2. 100个简单语句的时间复杂度为O（1）

T（n）=1 T(n)=O(1)

3.一个循环的时间复杂度是O(n)

             int  n=8,count=0;

                for(int i=1;i<=10n+100；i++){

                        count++;

                        }

T(n)=O(n)

T(n)=10n+100(常数项去掉)

4.时间复杂度为O(log2n)的循环语句

             int  n=8,count=0;

                for(int i=1;i<=n；i\*=2){

                        count++;

                        }

230：10亿 根据对数求指数

5.时间复杂度为O(n2)的二重循环

                int  n=8,count=0;

                for(int i=1;i<=n;i++){

                          for(int j=1;j<=n;j++){

                                    count++;

                        }

                    }

6.时间复杂度为O(nlog2n)的二重循环

             int  n=8,count=0;

                for(int i=1;i<=n;i\*=2){

                      for(int j=1;j<=n;j++){

                                    count++;

                        }

                       }

 7.时间复杂度为O(n2)的二重循环

             int  n=8,count=0;

                for(int i=1;i<=n;i++){

                      for(int j=1;j<=i;j++){

                                    count++;

                        }

                       }

1+2+…+n=(1+n)\*n/2 T（n）=O(n2)

时间复杂度是衡量算法优劣的指标

**常用的级别**

常见的时间复杂度按数量级递增排列依次为:

常数阶O(1)

对数阶O(log2 n)

线性阶O(n)

线性对数阶O(n\*log2 n)

平方阶O(n2)

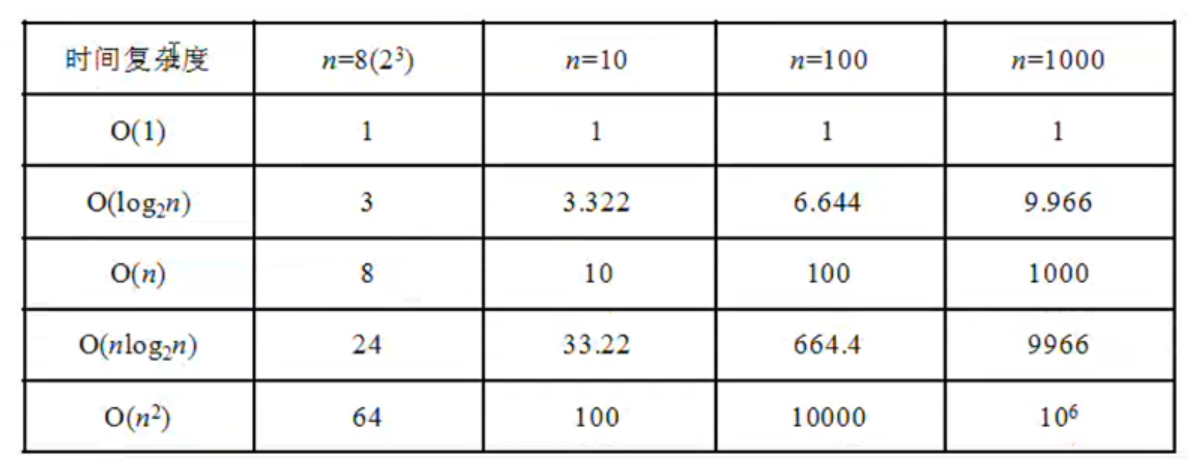
立方阶O(n3)

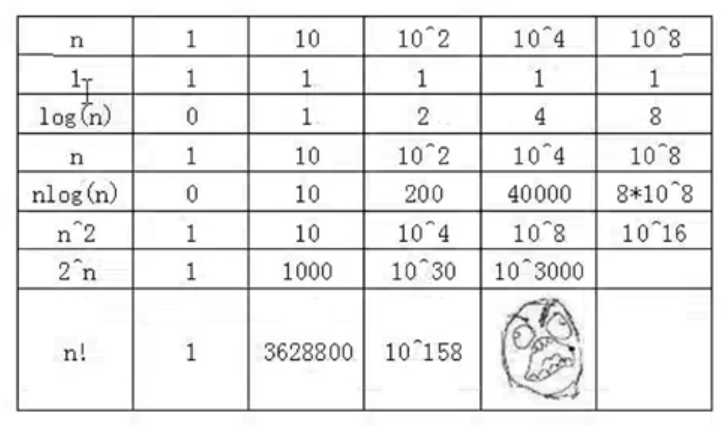
...

K次方阶O(nk)

指数阶O(2n)

阶乘阶O(n!)





**空间复杂度：**

算法的存储量包括：

1.程序本身所占空间

2.输入数据所占空间

3.辅助变量所占空间

空间复杂度是对一个算法在运行过程中临时占用的存储空间大小的量度，一般也作为问题规模n的函数，以数量级形式给出，记作：

**S(n)=O(g(n))** (Space)

int fun(int n){

    int i,j,k,s;

   s=0;

for(i=0;i<=n;i++)

     for(j=0;j<=i;j++)

         for(k=0;k<=j;k++)

                    s++;

       return(s);

   }

S(n)=O(1)

由于算法中变量的个数和问题规模n无关，所以空间复杂度均为S(n)=O(1)

无论循环结构如何循环，占用的都是四个空间

**注意**：空间复杂度相对时间复杂度分析要少