1.2.1算法的定义

算法 (Algorithm)

1. 一个有限指令集
2. 接受一些输入（有时不需要）
3. 产生输出
4. 一定在有限步骤后终止
5. 每一条指令必须：
    1. 有充分明确的目标，不可以有歧义
    2. 计算机能处理的范围之内
    3. 描述应不依赖于任何一种计算机语言及具体的实现手段

例1.选择排序的伪码描述

```
 void SlectionSort(int List[],int N)
{/* 将N个整数List[0]...List[N-1]进行非递减排序 */
  for(i=0;i<N;i++)
  {
    MinPosition=ScanForMin(List,i,N-1);
    /* 从List[i]到List[N-1]中找最小元，并将其位置赋值给MinPosition */
    Swap(List[i],List[MinPosition]);
    /* 将未排序部分的最小元换到有序部分的最后位置; */
  }
}
```

List到底是数组还是链表（看上去很像数组）？

Swap用函数还是用宏去实现？

1.2.2什么是好的算法

空间复杂度S(n)——根据算法写成的程序在执行时占用存储单元的长度。这个长度往往与输入数据的规模有关。空间复杂度过高的算法可能导致使用的内存超限，造成程序非正常中断。

1. 数组的长度：如果代码中应用了数组，那么数组的长度基本上就是空间复杂度；e.g.一维数组的空间复杂度是 $O(n)$；二维数组的空间复杂度是 $O(n^2)$
2. 递归的深度：如果代码中存在递归，那么递归的深度就是代码的空间复杂度

时间复杂度T(n)——根据算法写成的程序在执行时**耗费时间的长度**。这个长度往往也与输入数据的规模有关。时间复杂度过高的低效算法可能导致我们在有生之年都等不到运行结果。

要素n的规模。空间复杂度太大，非正常退出。时间复杂度太大，等不到结果。

1.1例2

```
void PrintN(int N)
{
    if(N)
    {
        PrintN(N-1);
        Printf("%d\n",N);
    }
    return ;
}
```

$$S(N)=C\cdot N$$

如果输入100000，要保存99999，99998，一直到0，存不下这么多的数据，就非正常中止了。

1.1例3

```
double f1(int n,double a[],double x)
{
    int i;
    double p=a[0];
    for(i=1;i<=n;i++)
    {
        p+=(a[i]*pow(x,i));
    }
    return p;
}

double f2(int n,double a[],double x)
{
    int i;
    double p=a[n];
    for(i=n;i>0;i--)
    {
        p=a[i-1]+x*p;
    }
    return p;
}
```

+-法速度很快几乎可以忽略不计，主要是*/法

第一个函数：pow有 $i-1$次乘法，再加上外面的一次，一个循环进行了 i次乘法，总共进行了: $(1+2+\cdots+n)=(n^2+n)/2$次乘法，时间复杂度: $T(n)=C_1n^2+C_2n$

第二个函数：每个循环只有一次乘法，一共只做了 n次乘法。时间复杂度： $T(n)=C\cdot n$，具体的C是多少不知道，每台机器都不太一样，但是当 n很大的时候，第二个程序会比第一个程序快很多，这时候 C的取值没有太大的影响

再分析一般算法的效率时，我们经常关注下面两种复杂度：

1. 最坏情况复杂度: $T_{worst}(n)$
2. 平均复杂度： $T_{avg}(n), T_{avg}(n)\leq T_{worst}(n)$

平均复杂度有时搞不太清楚，我们重点分析的是最坏情况复杂度。

1.2.3复杂度的渐进表示

复杂度的增长的性质，不做精细的分析 **复杂度的渐进表示法**：

1. $T(n)=O(f(n))$表示存在常数 $C>0, n_0>0$使得当 $n\geq n_0$时，有 $T(n)\leq C\cdot f(n)$，这代表了 $f(n)$是某种上界。
2. $T(n)=\Omega(g(n))$表示存在常数 $C>0, n_0>0$使得当 $n\geq n_0$时，有 $T(n)\geq C\cdot g(n)$，这代表了 $g(n)$是某种下界。
3. $T(n)=\Theta(h(n))$表示同时有 $T(n)=O(h(n))$和 $T(n)=\Omega(h(n))$既是上界也是下界。

分析算法效率的时候，无论是上下界都尽量取和实际相接近的 大概就是：找最小的上界，找最大的下界

增长速率： $1>logn>n>nlogn>n^2>n^3>2^n>n!$

好的程序员会思考： $n^2\rightarrow nlogn$

复杂度分析的小窍门：

若两段算法分别有复杂度 $T_1(n)=O(f_1(n))$和 $T_2(n)=O(f_2(n))$，则：

1. $T_1(n)+T_2(n)=max(O(f_1(n)), O(f_2(n)))$
2. $T_1(n)\times T_2(n)=O(f_1(n)\times f_2(n))$

若 $T(n)$是关于 n的 k阶多项式，那么 $T(n)=\Theta(n^k)$

一个for循环的时间复杂度等于循环次数乘以循环体代码的复杂度

f-else 结构的复杂度取决于if的条件判断复杂度和两个分枝部分的复杂度，总体复杂度取三者中最大