【例：求时间复杂度】

$T(n)=O(f(n))$，也就是要求出$f(n)$

例1 {++x;s=0;}

f(n)=1

```
int sum=0,n=100;
sum=(1+n)*n/2;
printf("%d",sum);
```

f(n)=3

$T(n)==O(f(n))=O(1)$

常量阶

与问题规模n无关，执行次数固定的均记做 O(1)

例2

```
for(i=1; i<=n; ++i)
    {++x; s+=x;}
```

f(n)=n

```
int  i, sum=0, n=100;
for(i=1;i<=n;i++)
{
      sum=sum+i;
}
printf("%d", sum);
```

f(n)=1+n+1=n+2

T(n)= O(n)　线性阶

例3

```
int i,sum=0,n=100;
for(i=1;i<=n;i++)
{
  for(j=1;j<=n;j++)
  {
        x++;
       sum=sum+x;
  }
}
printf("%d",sum);
```

$f(n)=1+n*n+1= n^2+2$

$T(n)= O(n^2)$

平方阶

```
for(i=2; i<=n; ++i)
   for(j=2; j<=i-1; ++j)
      {++x; a[i,j]=x;}
```

$f(n)=1+2+3+\ldots+n-2=(n-1)(n-2)/2=(n^2-3n+2)/2$

例4
```
for(i=1; i<=n; i++)
 for(j=1; j<=n; j++)
 { c[i][j]=0;
     for(k=1; k<=n; k++)
      c[i][j]=c[i][j]+a[i][k]*b[k][j]; }
```

$T(n) = O(n^3)$　　立方阶

例5

```
int count=1;
while(count<n)
{
    count=count*2;
}
```

$2^x=n$，得到$x=\log_2 n$

$T(n)=O(\log n)$ 对数阶

常用的时间复杂度所消耗的时间从小到大依次是：

$O(1) < O(\log n) < O(n) < O(n^2) < O(n^3) < O(2^n)$

【练习】求时间复杂度

(1)

```
x=90; y=100;
while(y>0)
if(x>100)
 {x=x-10;y--;}
else x++;
```

(2)

```
for (i=0;  i<n; i++)
for (j=0; j<m; j++)
a[i][j]=0;
```

(3)

```
s=0;
for i=0; i<n; i++)
for(j=0; j<n; j++)
        s+=B[i][j];
sum=s;
```

【练习】求时间复杂度

(4)
```
i=1;
while(i<=n)
    i=i*3;
```

(5)
```
x=0;
for(i=1; i<n; i++)
  for (j=1; j<=n-i; j++)
        x++;
```

(6)
```
f(int n)
{
    if(n<=1)
      return(1) ;
    else
      return(n*f(n-1));
}
```

复杂度分析小窍门

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

```
for (i=0;i<N;i++)
    for(j=0;j<N;j++)
        {x=y*x+z;k++;}
```

$O(n^2)$

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

$$T_1(n) + T_2(n) = \max(\, O(f_1(n)), O(f_2(n)) \,)$$

$$T_1(n) \times T_2(n) = O(\, f_1(n) \times f_2(n) \,)$$

若T(n)是关于n的k阶多项式，那么$T(n)=O(n^k)$

if-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

```
if(P1)        /*P1的复杂度为O(f1)*/
      P2;   /*P1的复杂度为O(f2)*/
else
      P3;   /*P3的复杂度为O(f3)*/
```

复杂度为 max (O(f1),O(f2),O(f3))