数据结构与算法本身解决的问题是快和省的问题，即如何让代码运行的更快，如何让代码更省存储空间。

一. 大 O 复杂度表示法

```
int cal(int n) {
  int sum = 0;
  int i = 1;
  for (; i <= n; ++i) {
    sum = sum + i;
  }
  return sum;
}

```

假设每行代码的运行时间为 u_time,总执行时间 $T(n)=(2n+2)*unit\_time$

可以看出所有的代码执行时间与每行代码的执行次数成正比。

所有 $T(n)=O(f(n))$;$f(n)$代表没行代码执行的次数总和。O 代表代码的执行时间 $T(n)$与 $f(n)$表达式成正比。

大 O 时间复杂度实际表达代码执行时间随着数据规模增长的变化趋势，所以也叫作渐进时间复杂度

当 n 很大，我们只需要记住最大的量级就可以了。

二。时间复杂度分析

2.1 只关注循环执行次数最多的一段代码

在分析一个算法，一段代码的时间复杂度的时候，只关心循环执行次数最多的那段代码就可以了。常量，低阶都可以忽略。

```
int cal(int n) {
  int sum = 0;
  int i = 1;
  for (; i <= n; ++i) {
    sum = sum + i;
  }
  return sum;
}

```

上面这段代码就可以表示为 **O(n)**

2.2 加法法则：总复杂度等于量级最大的那段代码的复杂度。

```
int cal(int n) {
    int sum_1 = 0;
    int p = 1;
    for (; p < 100; ++p) {
      sum_1 = sum_1 + p;
    }

    int sum_2 = 0;
    int q = 1;
    for (; q < n; ++q) {
      sum_2 = sum_2 + q;
    }

    int sum_3 = 0;
    int i = 1;
    int j = 1;
    for (; i <= n; ++i) {
      j = 1;
      for (; j <= n; ++j) {
        sum_3 = sum_3 +   i * j;
      }
    }

    return sum_1 + sum_2 + sum_3;
```

}

综合，这三段代码的时间复杂度，我们取最大的量级 O(n).**就是说：总的时间复杂度等于量级最大的那段代码的时间复杂度。**

2.3 乘法法则：嵌套代码的复杂度等于嵌套内外代码的复杂度的乘积。

```
int cal(int n) {
   int ret = 0;
   int i = 1;
   for (; i < n; ++i) {
     ret = ret + f(i);
   }
 }

 int f(int n) {
  int sum = 0;
  int i = 1;
  for (; i < n; ++i) {
    sum = sum + i;
  }
  return sum;
 }
```

上面代码的时间复杂度就是 **T(n）=T1(n)*T2(n)=O(n*n)=O(n2).**

三。几种常见的时间复杂度实例分析

分为两大类：多项式量级与非多项式量级（上图画波浪线的两个，带数据规模 **n** 的增加，执行时间会急剧增加，是非常低效的算法）

1.O(1)

常量级别的时间表达式

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

2. O(logn)、O(nlogn)

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

如果一段代码的时间复杂度是 O(logn)，我们循环执行 n 遍，时间复杂度就是 O(nlogn) 了。

3. O(m+n)、O(m*n)

```
int cal(int m, int n) {
    int sum_1 = 0;
    int i = 1;
    for (; i < m; ++i) {
        sum_1 = sum_1 + i;
    }

    int sum_2 = 0;
    int j = 1;
    for (; j < n; ++j) {
        sum_2 = sum_2 + j;
    }

    return sum_1 + sum_2;
}
```

我们无法事先评估 m 和 n 谁的量级大

所以，上面代码的时间复杂度就是 O(m+n)。

我们需要将加法规则改为：T1(m) + T2(n) = O(f(m) + g(n))。但是乘法法则继续有效：T1(m)*T2(n) = O(f(m) * f(n))。

四。空间复杂度分析

时间复杂度的全称是渐进时间复杂度，表示算法的执行时间与数据规模之间的增长关系。

空间复杂度全称就是渐进空间复杂度，表示算法的存储空间与数据规模之间的增长关系。

```
void print(int n) {
    int i = 0;
    int[] a = new int[n];
    for (i; i <n; ++i) {
        a[i] = i * i;
    }

    for (i = n-1; i >= 0; --i) {
        print out a[i]
    }
}
```

第 3 行申请了一个大小为 n 的 int 类型数组，除此之外，剩下的代码都没有占用更多的空间，所以整段代码的空间复杂度 O(n)。

我们常见的空间复杂度就是 O(1)、O(n)、O(n2)，像 O(logn)、O(nlogn) 这样的对数阶复杂度平时

都用不到。

越高阶复杂度的算法，执行效率越低。常见的复杂度并不多，从低阶到高阶有：O(1)、O(logn)、O(n)、O(nlogn)、O(n2)