算法分析

算法分析

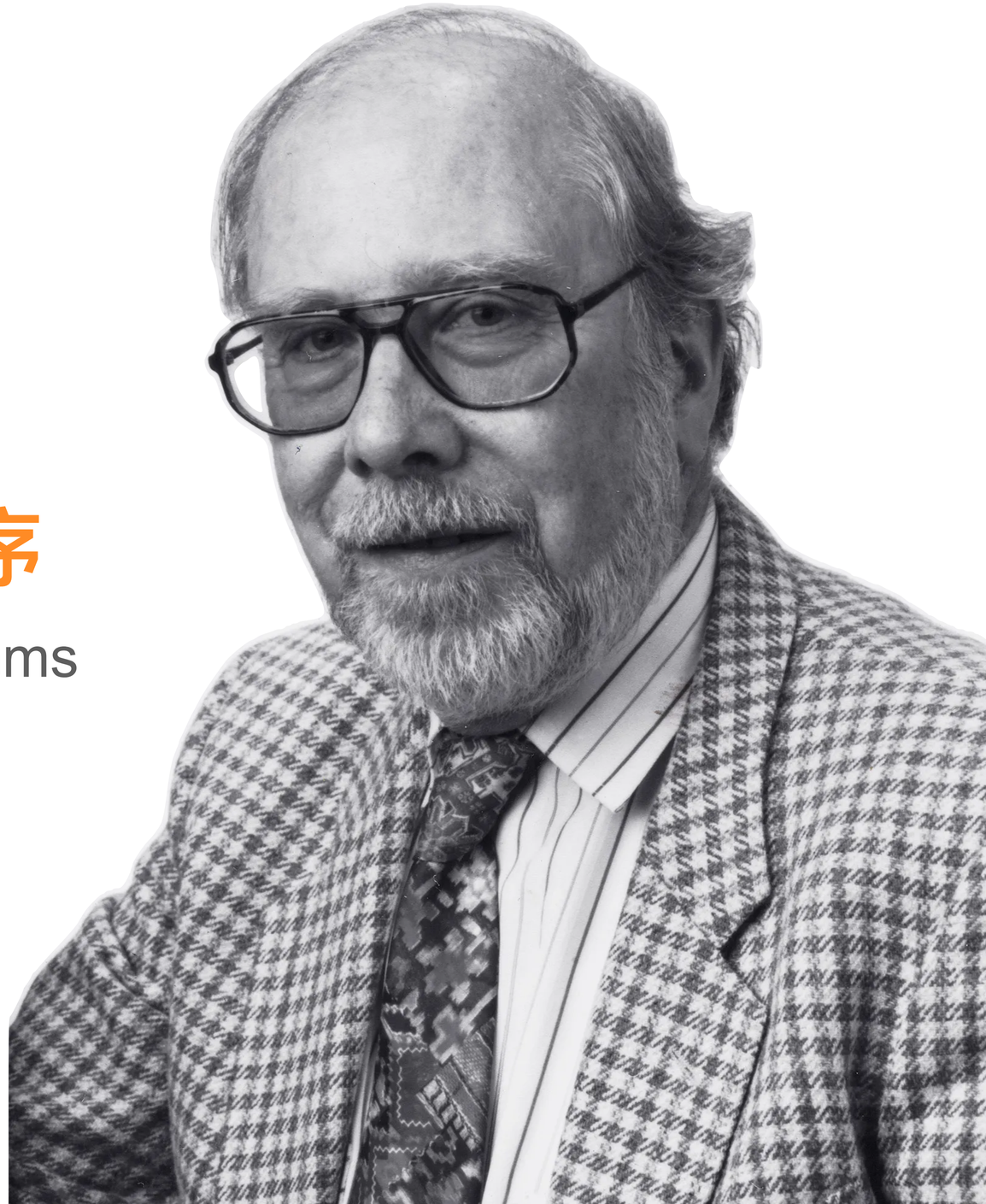
时间复杂度

空间复杂度

抽象数据类型 ADT（Abstract Data Type）

算法 + **数据结构** = **程序**

Algorithms + Data Structures = Programs



数据结构与算法之间存在着本质联系。在研究某一类型的数据结构时，总要涉及其上施加的运算。只有通过对所定义运算的研究，才能真正理解数据结构的定义和作用。

算法要满足的 5 个重要特性

有穷性

确定性

可行性

输入

输出

评价算法优劣的基本标准

正确性

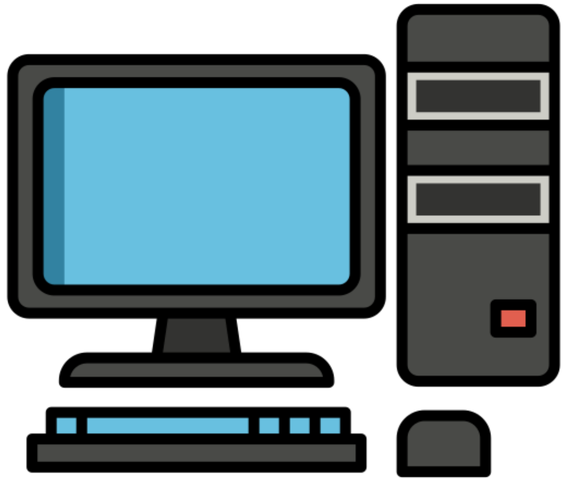
可读性

健壮性

高效性

算法的效率

A



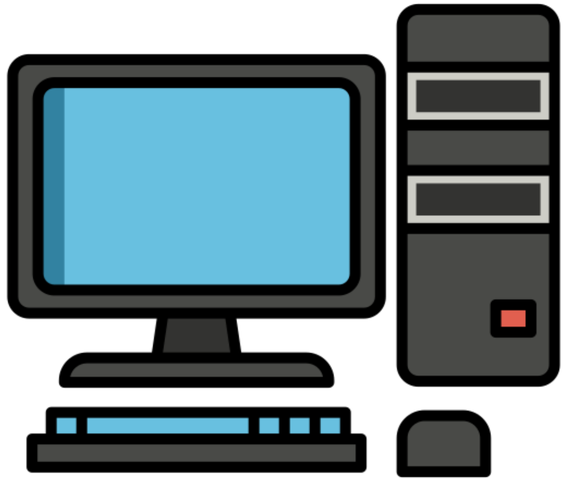
每秒执行百亿条指令

B

每秒执行千万条指令

算法的效率

A



每秒执行百亿条指令

B

每秒执行千万条指令

问题：对1000万个数进行排序，谁快?

A 用插入排序大约 5.5 小时完成

B 用归并排序大约 20 分钟完成

时间复杂度

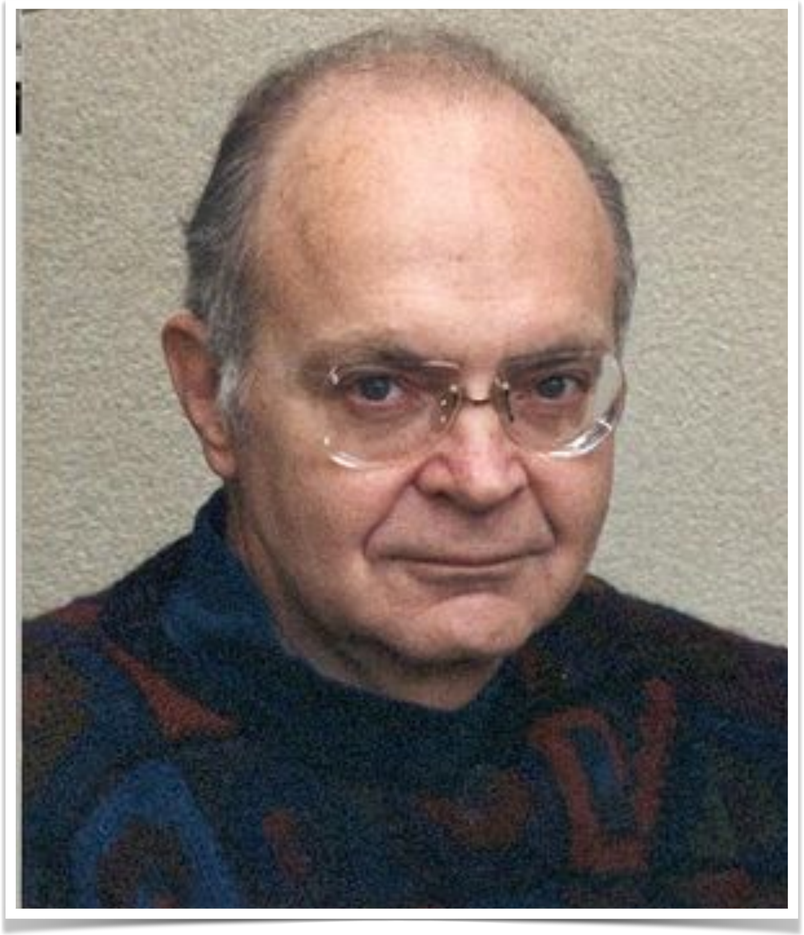
也称渐近时间复杂度，**T(n) = O(f(n))**

随着问题规模 n 的增大，算法执行时间和增长率和 f(n) 增长率成正比

程序运行的总时间主要和两点有关

执行每条语句的耗时

每条语句的执行频率



唐纳德·尔文·克努斯

Donald Ervin Knuth

关注语句频度

由于语句的执行要由源程序翻译成目标代码，目标代码经装配再执行，因此语句执行一次实际所需的具体时间是与机器的软、硬件环境（如机器速度、编译程序质量等）密切相关的。所以，所谓的算法分析并非实际执行所需时间，而是针对算法中语句的执行次数做出估计，从中得到算法执行时间的信息。

计算频度

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









}
```

频度为 n+1

计算频度

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




    }
}
```

频度为 n+1

频度为 n×(n+1)

计算频度

```
for(int i=1; i<=n;i++)
{
    for(int j=1;j<=n;j++)
    {
        c[i][j] = 0;
    }
}
```

频度为 n+1

频度为 n×(n+1)

频度为 $n \times n=n^2$

计算频度

```
for(int i=1; i<=n;i++)
{
    for(int j=1;j<=n;j++)
    {
        c[i][j] = 0;
        for(int k=1;k<=n;k++)
        {


        }
    }
}
```

频度为 n+1

频度为 $n \times (n+1)$

频度为 $n \times n = n^2$

频度为 $n \times n \times (n+1) = n^2 * (n+1)$

计算频度

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

频度为 n+1

频度为 $n \times (n+1)$

频度为 $n \times n = n^2$

频度为 $n \times n \times (n+1) = n^2 * (n+1)$

频度为 $n \times n \times n = n^3$

计算时间复杂度

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

频度为 $n+1$

频度为 $n \times (n+1)$

频度为 $n \times n = n^2$

频度为 $n \times n \times (n+1) = n^2 * (n+1)$

频度为 $n \times n \times n = n^3$

$$f(n) = n+1+n\times(n+1)+n^2+n^2\times(n+1)+n^3$$

$$f(n) = 2n^3+3n^2+2n+1$$

计算时间复杂度

若 $f(n)=a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0$ 是一个m次多项式，则 $T(n)=O(n^m)$

在计算算法时间复杂度时，可以忽略所有低次幂和最高次幂的系数，这样可以简化算法分析，也体现出了增长率的含义。

$$f(n) = 2n^3+3n^2+2n+1$$

$$T(n) = O(n^3)$$

频度计算

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

如果 n 为 0 呢?

f(n)=1

T(n)=O(1)

时间复杂度

最好时间复杂度：算法在最好情况下的时间复杂度。

最坏时间复杂度：算法在最坏情况下的时间复杂度。

平均时间复杂度：算法在所有可能的情况下，按照输入实例以等概率出现时，算法计量的加权平均值。

对算法时间复杂度的度量，通常只讨论算法在最坏情况下的时间复杂度，即分析在最坏情况下，算法执行时间的上界。

计算时间复杂度 – 常量阶示例

```
x++;        频度为 1
s = 0;      频度为 1
```

f(n)=1+1=2

T(n)=O(1)

```
for(int i=0;i<10000;i++)
{
    x++;
    s = 0;
}
```

T(n)=O(1)

计算时间复杂度 – 线性阶示例

```
for(int i=0;i<n;i++)          频度为 n+1
{
    x++;                      频度为 n
    s = 0;                    频度为 n
}
```

f(n)=(n+1)+n+n=3n+1

T(n)=O(n)

计算时间复杂度 – 平方阶示例

```
x = 0;
y = 0;
for(int k=1;k<=n;k++)
{
    x++;
}
for(int i=1;i<=n;i++)
{
     for(int j=1;j<=n;j++)
     {
          y++;
     }
}
```

频度为 1

频度为 1

频度为 n+1

频度为 n

频度为 n+1

频度为 n×(n+1)

频度为 n^2

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

$$=2n^2+4n+4$$

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

计算时间复杂度 – 立方阶示例

```
x = 1;
for(int i=1;i<=n;i++)
{
    for(int j=1;j<=i;j++)
    {
        for(int k=1;k<=j;k++)
        {
            x++;
        }
    }
}
```

频度为 1

频度为 n

频度为 (1+2+3+…+n)=n(n+1)/2

频度为 1+[1+(1+2)]+[1+(1+2)+(1+2+3)]+…+n(n+1)/2

= 1+4+10+…+n(n+1)/2

= [n(n+1)(n+2)]/6

$T(n)=O(n^3)$

	i=1	i=2	i＝3	i=4	i=n
一层	1次	2次	3次	4次	
二层	1次	3次	6次	10次	
三层	1次	4次	10次	20次	

三角形数列：

- 三角形数列的第 `n` 项是前 `n` 个自然数的和。
- 数列项的公式为：

$$T_n = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

累加和数列：

- 三阶累加数列则是在三角形数列的基础上，再进行一次累加。
- 三阶累加数列的第 `n` 项是前 `n` 个三角形数的和。
- 数列项的公式为：

$$S_n = T_1 + T_2 + T_3 + \cdots + T_n$$

即：

$$S_n = \sum_{i=1}^{n} \frac{i(i+1)}{2}$$

为了计算这种数列的总和，我们可以按以下步骤进行：

1. **计算单个三角形数**：
   - 例如，`T_1 = 1`，`T_2 = 1 + 2 = 3`，`T_3 = 1 + 2 + 3 = 6`，等等。
   - 对于任意 `n`，三角形数的公式为：

$$T_n = \frac{n(n+1)}{2}$$

2. **累加所有三角形数**：
   - 三阶累加数列的第 `n` 项为：

$$S_n = T_1 + T_2 + T_3 + \cdots + T_n$$

   - 代入三角形数公式：

$$S_n = \frac{1 \times 2}{2} + \frac{2 \times 3}{2} + \frac{3 \times 4}{2} + \cdots + \frac{n \times (n+1)}{2}$$

3. **化简并求和**：

   - 为了方便计算，我们可以把公式中的 `1/2` 提出来：

$$S_n = \frac{1}{2} \sum_{i=1}^{n} i(i+1)$$

   - 进一步展开 `i(i + 1)`：

$$S_n = \frac{1}{2} \sum_{i=1}^{n} (i^2 + i)$$

   - 将 `i^2` 和 `i` 分开求和：

$$S_n = \frac{1}{2} \left(\sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i \right)$$

4. **利用求和公式**：

   - 对于 `i` 的平方和，我们有公式：

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

   - 对于 `i` 的自然数和，我们有公式：

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

4. **利用求和公式**：

   - 对于 `i` 的平方和，我们有公式：

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

   - 对于 `i` 的自然数和，我们有公式：

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

5. **代入并化简**：

   - 将这两个公式代入总和公式：

$$S_n = \frac{1}{2} \left(\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \right)$$

   - 最终化简为：

$$S_n = \frac{n(n+1)(n+2)}{6}$$

计算时间复杂度 – 对数阶示例

```
for(int i=1;i<=n;i=i*2)
{
    x++;
    s = 0;
}
```

$$2^{t-1} > n$$

$$\log_2 2^{t-1} > \log_2 n$$

$$t-1 > \log_2 n$$

$$t > \log_2 n+1$$

次数	1	2	3	4	t
i	1	2	4	8	?
i	2^0	2^1	2^2	2^3	2^{t-1}

$$T(n)=\log_2 n+1$$
$$=O(\log_2 n)$$

时间复杂度汇总

时间	名称
1	常数阶
n	线性阶
n^2	平方阶
n^3	立方阶
logn	对数阶
nlogn	线性对数阶
2^n	指数阶
n!	阶乘阶

计算时间复杂度

[2011] 设 n 是描述问题规模的非负整数，下面程序片段的时间复杂度是_____。

```
x = 2;
while(x<n/2)
    x = 2 * x;
```

A. $O(\log_2 n)$　B. $O(n)$　C. $O(n\log_2 n)$　D. $O(n^2)$

次数	1	2	3	4	t
x	2^2	2^3	2^4	2^5	2^{t+1}

$$2^{t+1} = n / 2$$

$$2^{t+2} = n$$

$$\log_2 2^{t+2} = \log_2 n$$

$$t+2 = \log_2 n$$

$$t = \log_2 n - 2$$

$$T(n) = \log_2 n - 2 = O(\log_2 n)$$

计算时间复杂度

[2011] 设 n 是描述问题规模的非负整数，下面程序片段的时间复杂度是_____。

```
x = 2;
while(x<n/2)
    x = 2 * x;
```

A. $O(\log_2 n)$　B. $O(n)$　C. $O(n\log_2 n)$　D. $O(n^2)$

次数	1	2	3	4	t
i	2^2	2^3	2^4	2^5	2^{t+1}

$$2^{t+1} = n / 2$$

$$2^{t+2} = n$$

$$\log_2 2^{t+2} = \log_2 n$$

$$t+2 = \log_2 n$$

$$t = \log_2 n - 2$$

$$T(n) = \log_2 n - 2 = O(\log_2 n)$$

计算时间复杂度

[2017] 下列函数的时间复杂度是_____。

```
int fun(int n){
    int i = 0, sum = 0;
    while(sum<n) sum += ++i;
    return i;
}
```

A. O(logn)　　B. $O(n^{1/2})$

C. O(n)　　D. O(nlogn)

次数	1	2	3	4	t
i	1	2	3	4	t
sum	1	3	6	10	t(t+1)/2

$t(t+1)/2 = n$

$t(t+1) = 2n$

$t^2+t = 2n$

$t^2 \approx 2n$

计算时间复杂度

[2017] 下列函数的时间复杂度是_____。

```
int fun(int n){
    int i = 0, sum = 0;
    while(sum<n) sum += ++i;
    return i;
}
```

A. O(logn)　　B. $O(n^{1/2})$

C. O(n)　　D. O(nlogn)

次数	1	2	3	4	t
i	1	2	3	4	t
sum	1	3	6	10	t(t+1)/2

$t \approx \sqrt{2n}$

$T(n) = O(\sqrt{n})$

$= O(n^{1/2})$

计算时间复杂度

[2017] 下列函数的时间复杂度是_____。

```
int fun(int n){
    int i = 0, sum = 0;
    while(sum<n) sum += ++i;
    return i;
}
```

A. O(logn)　　B. $O(n^{1/2})$

C. O(n)　　D. O(nlogn)

次数	1	2	3	4	t
i	1	2	3	4	t
sum	1	3	6	10	t(t+1)/2

$t \approx \sqrt{2n}$

$T(n)=O(\sqrt{n})$

$=O(n^{1/2})$

计算时间复杂度

[2019] 设 n 是描述问题规模的非负整数，下列程序的时间复杂度是_____。

```
x = 0;
while(n>=(x+1)(x+1))
     x = x + 1;
```

A. O(logn)　　B. $O(n^{1/2})$　　C. O(n)　　D. $O(n^2)$

$n < (x+1)^2$

$\sqrt{n} < x+1$

$\sqrt{n}-1 < x$

$T(n)=O(\sqrt{n})$

$=O(n^{1/2})$

计算时间复杂度

[2019] 设 n 是描述问题规模的非负整数，下列程序的时间复杂度是_____。

```
x = 0;
while(n>=(x+1)(x+1))
     x = x + 1;
```

A. O(logn)　　B. O($n^{1/2}$)　　C. O(n)　　D. O(n^2)

$n < (x+1)^2$

$\sqrt{n} < x+1$

$\sqrt{n}-1 < x$

$T(n)=O(\sqrt{n})$

$=O(n^{1/2})$

计算时间复杂度

[2022] 下列程序的时间复杂度是_____。

```
int sum = 0;
for(int i=1;i<n;i*=2)
    for(int j=0;j<i;j++)
        sum++;
```

A. O(logn)　　B. O(n)

C. O(nlogn)　　D. O(n^2)

次数	1	2	3	4	t
i	2^0	2^1	2^2	2^3	2^{t-1}

$$2^{t-1} = n$$

$$\log_2 2^{t-1} = \log_2 n$$

$$t-1 = \log_2 n$$

$$t = \log_2 n+1$$

计算时间复杂度

[2022] 下列程序的时间复杂度是_____。

```
int sum = 0;
for(int i=1;i<n;i*=2)
    for(int j=0;j<i;j++)
        sum++;
```

A. O(logn)　　B. O(n)

C. O(nlogn)　　D. O(n^2)

次数	1	2	3	4	t
i	2^0	2^1	2^2	2^3	2^{t-1}
内层次数	2^0	2^1	2^2	2^3	2^{t-1}

$$t = \log_2 n+1$$

$$1+2+4+\cdots+2^{t-1}$$

$$=1+2+4+\cdots+2^{\log_2 n+1-1}$$

$$=1+2+4+\cdots+2^{\log_2 n}$$

$$=1\times(1-2^{\log_2 n})/(1-2)=n-1$$

计算时间复杂度

[2022] 下列程序的时间复杂度是_____。

```
int sum = 0;
for(int i=1;i<n;i*=2)
    for(int j=0;j<i;j++)
        sum++;
```

A. O(logn)　　B. O(n)

C. O(nlogn)　　D. $O(n^2)$

次数	1	2	3	4	t
i	2^0	2^1	2^2	2^3	2^{t-1}
内层次数	2^0	2^1	2^2	2^3	2^{t-1}

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

空间复杂度

空间复杂度主要用来描述某个算法对应的程序想在计算机上执行，除了用来存储代码和输入数据的内存空间外，还需要额外的空间。

$$S(n) = O(f(n))$$

抽象数据类型 ADT（Abstract Data Type）

ADT 是一种编程概念，用于定义数据的类型及其操作，而不涉及具体实现细节。它提供了一种将数据的逻辑表示与物理实现分离的方法，从而使程序更具可维护性和可扩展性。

在C语言中，ADT 通常通过结构体和函数的结合来实现。结构体用于定义数据的类型，而函数用于操作这些数据。通过这种方式，程序员可以隐藏数据的内部结构，仅暴露出操作数据的接口。

设计一台电视机

```
ADT 电视机 {
    打开电视机；
    关闭电视机；
    调节音量；
    换台；
}
```

设计一个复数

ADT Complex {

数据对象：D={e1,e2|e1,e2∈R,R是实数}

数据关系：S={<e1,e2>e1是复数的实部，e2是复数的虚部}

基本操作：

create(&C,x,y)

操作结果：构造复数C,其实部和虚部分别赋值参数x和y的值。

GetReal(C)

初始条件：复数C已存在。

操作结果：返回复数C的实部值

} ADT Complex

复数表示与实现

案例：算法分析/test1.c

```
typedef struct
{
    float Realpart;
    float Imagepart;
} Complex;
```

```
void create(Complex *C, float x, float y)
{
    C->Realpart = x;
    C->Imagepart = y;
}
```

作业 P16-6

(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;
```

(4)

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

作业　P16-6

(5)

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

(6)

```
x = n; //n>1
y = 0;
while(x>=(y+1)*(y+1))
    y++;
```