时间与空间复杂度

B站：洛金洛梧桐

对应训练

算法竞赛入门指南：第一节“时间与空间复杂度”

练习	本课对应内容
P2242 时间复杂度 1	单层、双层循环与准确计数
P2243 时间复杂度 2	倍增、折半与多层循环
P2244 时间复杂度 3	搜索树、枚举子集、排列
P2245 空间复杂度	数组、同时占用与递归栈

时间复杂度与空间复杂度

时间复杂度用于描述算法的操作次数随数据规模增长的变化。空间复杂度用于描述算法占用的空间随数据规模增长的变化。

例如，遍历一个长度为 n 的数组，需要访问 n 个元素。若用两层循环枚举所有位置对，则需要进行 n^2 次操作。

时间复杂度
操作次数随规模的增长

空间复杂度
空间占用随规模的增长

在做题前，我们需要根据数据范围估计操作次数和内存占用，判断当前做法能否满足题目的限制。

基本操作次数

本节的三组时间复杂度练习采用以下约定：

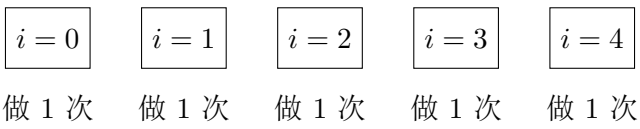
- 一次 `work()` 算一次基本操作。
- 循环判断、变量加一、赋值、普通函数调用等开销，在这些题里忽略。
- 1 秒最多允许 2×10^8 次，即两亿次。
- **超过才算超时，恰好两亿次不超时。**



以上约定只用于这三组练习。实际程序的运行时间还与具体操作、编译优化和评测机性能有关。

例：单层循环

```
for(int i=0;i<5;i++){
    work();
}
```



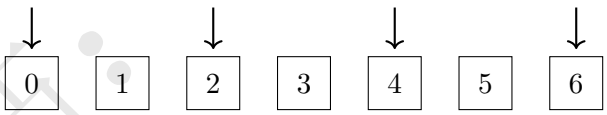
在这段代码中，i 的取值为 $0, 1, 2, 3, 4$，因此 `work()` 执行 5 次。循环结束时还会判断一次条件，但按本题约定，这次判断不计入基本操作次数。

单层循环的次数

写法	i 的取值	次数
`i=0;i<n;i++`	0 到 $n-1$	n
`i=1;i<=n;i++`	1 到 n	n
`i=0;i<=n;i++`	0 到 n	$n+1$
`i=1;i<n;i++`	1 到 $n-1$	$n-1$

步长为 2 的循环

```
1 for(int i=0;i<7;i+=2) work();
```



此时 i 的取值为 $0, 2, 4, 6$，共执行 4 次。注意，C++ 中整数除法 $7/2$ 的结果是 3，不能直接用它作为这里的循环次数。

顺序执行的循环

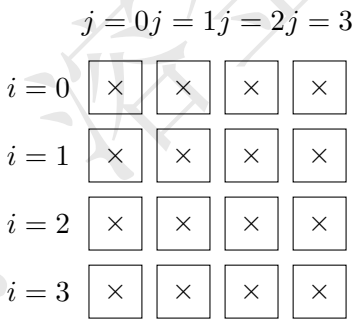
```
1 const int n=100000000;
2 for(int i=0;i<n;i+=2) work();
3 for(int i=0;i<n;i++) work();
```

第一段循环执行 50000000 次，第二段执行 100000000 次，总共执行 150000000 次，未超过本题规定的上限。两段循环顺序执行时，总次数是两段次数之和。如果循环存在嵌套，则需要计算外层每次执行时，内层分别执行多少次。

B站：溶金洛梧桐

双层循环

```
for(int i=0;i<4;i++){
    for(int j=0;j<4;j++){
        work();
    }
}
```



外层循环执行 4 次，每次都执行一次完整的内层循环。内层也是 4 次，因此总次数为 $4 \times 4 = 16$。图中每一格对应一次 `work()`。

若两层都做 n 次，总次数为 $n \times n = n^2$。

n	`work()` 总次数	本题模型下
10000	100000000	不超时
20000	400000000	超时

内层步长为 2 的情况

若外层做 30000 次，内层从 0 开始、每次加 2、小于 30000，内层做 15000 次，总共 450000000 次，超时。

B站：落金落梧桐

内层次数不固定的循环

```
for(int i=0;i<n;i++){
    for(int j=0;j<i;j++) work();
}
```

取 $n=5$，每一行的长度不同：

$$
\begin{aligned}
&i=0 \quad \text{这一行 } 0 \text{ 次}\\
&i=1 \quad \boxed{\times}\\
&i=2 \quad \boxed{\times}\ \boxed{\times}\\
&i=3 \quad \boxed{\times}\ \boxed{\times}\ \boxed{\times}\\
&i=4 \quad \boxed{\times}\ \boxed{\times}\ \boxed{\times}\ \boxed{\times}
\end{aligned}
$$

总共 $0+1+2+3+4=10$ 次，不是 25 次。

把同样的三角形倒过来拼上去，就能得到一个长方形。因此一般情况的准确次数为

$$0+1+\cdots+(n-1)=\frac{n(n-1)}{2}.$$

n	准确次数	是否超时
20000	199990000	否
20001	200010000	是

两种代码的增长方式几乎一样，但是否超过两亿次，要按**准确次数**判断。

B站：浔金落梧桐

时间复杂度的表示方法

一般使用大 O 记号表示时间复杂度。计算时保留主要增长项，忽略常数系数和低次项。

工作次数	常用复杂度	直观理解
5 或 100	$O(1)$	不随 n 增长
n 或 $3n$	$O(n)$	数据翻倍，工作量约翻倍
n^2 或 $n(n-1)/2$	$O(n^2)$	数据翻倍，工作量约变四倍
n^2+n	$O(n^2)$	数据大时，平方项决定主要增长

例如，操作次数为 n^2+n 时，随着 n 增大，n^2 所占的比例越来越大，因此时间复杂度记为 $O(n^2)$。大 O 表示增长的上界，本讲义按主要增长项填写。

操作次数与时间复杂度

```
1 for(int i=0;i<n;i++) work();
2 for(int i=0;i<n;i++) work();
```

这两段循环共执行 $2n$ 次，时间复杂度仍为 $O(n)$。不过，题目要求准确操作次数时，仍应计算为 $2n$，不能省略系数。

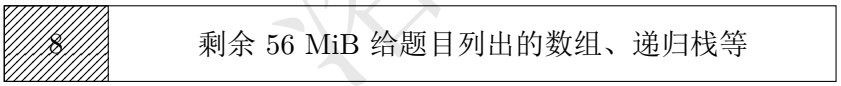
准确次数 决定本模型是否超限	复杂度 描述随规模怎样增长

循环层数不能直接决定时间复杂度，还需要结合各层的执行次数。

空间复杂度

空间复杂度练习 P2245 规定：内存上限为 64 MiB，程序已有 8 MiB 的固定开销。

单位或类型	本题约定
1 MiB	$1024 \times 1024 = 1048576$ 字节
`int`	4 字节
`long long`	8 字节
`unsigned char`	1 字节



固定开销

总占用严格超过上限才爆空间，等于上限不爆。题目中的 `vector` 容量恰好等于长度，忽略容器对象等额外开销；递归每层另计 64 字节，包括终止层。

这些是本题约定。真实环境要看类型大小、容器容量、分配与栈限制，不能默认程序始终固定占 8 MiB。

空间复杂度与准确内存

固定几个变量是 $O(1)$，长度随 n 增长的数组是 $O(n)$，n 行 n 列的数组是 $O(n^2)$。但是否超过 64 MiB，要计算实际格数与字节数，不能只写一个大 O。

数组占用的内存

```
vector<int> a(10000000);
use(a);
```

一千万格，每格 4 字节，数组占 40000000 字节；加上固定的 8388608 字节，总共 48388608 字节，不爆空间。

二维数组

```
const int n=4000;
vector<int> a(n*n);
```

$4000 \times 4000 = 16000000$ 格，元素占 64000000 字节。**6400 万字节不是 64 MiB**。加固定开销后为 72388608 字节，超过 67108864 字节，爆空间。

多个数组

```
vector<int> a(3000*3000);
vector<int> b(2000*2000);
use(a);use(b);
```

数组共占 $(9000000 + 4000000) \times 4 = 52000000$ 字节，加入固定开销为 60388608 字节，不爆空间。

内存限制的边界

```
vector<unsigned char> a(56ULL*1024*1024);
```

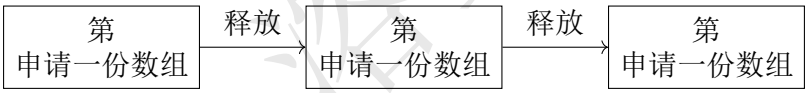
数组占 56 MiB，加上固定开销恰好为 64 MiB，不超过限制。若数组长度再增加 1，总占用为 67108865 字节，超过限制。

B站：溶金落梧桐

峰值内存

```
for(int i=0;i<20;i++){
    vector<int> temp(10000000);
    use(temp);
}
```

temp 在每轮循环结束时释放。因此任意时刻最多保留一份数组，不能把 20 轮的数组空间相加。



峰值仍是一份数组加固定开销，为 48388608 字节。在本题模型中不计已释放的内存。

递归栈

```
void f(int k){
    if(k==0) return;
    f(k-1);
}
// f(1000000)
```

调用从一百万一直到 0，同时存在 1000001 层。每层 64 字节，加固定开销，总共

$$1000001 \times 64 + 8388608 = 72388672 \text{ 字节}.$$

这个结果超过了本题上限。计算递归的时间时，需要统计全部调用；计算递归栈的空间时，则统计同时尚未返回的调用层数。

B站：资金落梧桐

递归中的局部数组

```
void f(int n){
    vector<int> temp(n);
    if(n>1) f(n/2);
    use(temp);
}
// f(8*1024*1024)
```

由于子函数返回后还要执行 `use(temp)`，当前层的 temp 不会提前释放。因此递归到最深处时，各层的数组同时占用内存。

第 1 层：8388608 个 int，等待子函数

第 2 层：4194304 个 int，继续等待

第 3 层：2097152 个 int

一直减半，到第 24 层：1 个 int

元素总数为 $8388608 + 4194304 + \cdots + 1 = 16777215$，占 67108860 字节。

还有 24 层栈，共 1536 字节，最后加固定开销：

$$67108860 + 1536 + 8388608 = 75499004 \text{ 字节.}$$

总占用超过 64 MiB，所以会超出空间限制。这里需要同时计算所有仍未释放的数组、递归栈和固定开销。

递归的时间与空间

深度为 k 的二叉递归可能有约 2^k 次调用，但深度优先执行时，一次只沿一条根到叶子的路径等待返回；若每层只用固定空间，递归栈为 $O(k)$，不是 $O(2^k)$。

B站：浴金落梧桐

练习

以下时间题只统计 `work()` 的执行次数，上限为 2×10^8 次。空间题的上限为 64 MiB，固定开销为 8 MiB。

练习 1：两段循环

```
const int n=10000;
for(int i=0;i<n;i++) work();
for(int i=0;i<n;i++)
    for(int j=0;j<n;j++) work();
```

写出准确次数、常用时间复杂度，以及是否超时。

练习 2：只改一个符号

```
for(int x=1;x<=32;x*=2) work();
```

写出每次进入循环时 x 的值。把小于等于改成小于后，次数怎样变化？

练习 3：叶子与全部节点

二叉搜索从 `dfs(3)` 开始，每层分两路，到 0 停止。只在叶子 work 与每层都 work，分别做多少次？

练习 4：内存

```
vector<long long> a(8000000);
use(a);
```

写出元素占用、峰值总占用，判断是否爆空间。

练习 5：循环与递归

循环二十轮，每轮申请数组、使用、释放；递归二十层，每层申请数组且等子函数返回后才释放。这两段程序的峰值内存为何不同？

B站：浴金落梧桐

课堂练习参考

1. $10000 + 100000000 = 100010000$ 次，$O(n^2)$，不超时。准确计数保留两段，复杂度只留下主要增长项。
2. $1, 2, 4, 8, 16, 32$，共 6 次；改成小于后共 5 次，增长方式均为 $O(\log n)$。
3. 只在叶子共 8 次；每个节点共 $1 + 2 + 4 + 8 = 15$ 次。
4. 元素占 64000000 字节，总共 72388608 字节，爆空间。
5. 循环每轮结束就释放，最多一份数组；递归深入时上层尚未返回，可能有二十份数组同时存在，还需加递归栈。

B站：洛谷落梧桐