

# Part I 完全数

Q1. 该程序在你的电脑上执行了多久？共有多少个完全数，分别是哪些？

A: 执行时间如图所示：

```
SimpleTest CppLegs
Tests from perfect.cpp

Correct (PROVIDED_TEST, line 94) Confirm divisorSum of small inputs
Correct (PROVIDED_TEST, line 100) Confirm 6 and 28 are perfect
Correct (PROVIDED_TEST, line 105) Confirm 12 and 98765 are not perfect
Correct (PROVIDED_TEST, line 110) Test oddballs: 0 and 1 are not perfect
Correct (PROVIDED_TEST, line 115) Confirm 33550336 is perfect

Correct (PROVIDED_TEST, line 119) Time trials of findPerfects on doubling input sizes
Line 120 TIME_OPERATION findPerfects(10000) (size = 10000) completed in 0.265 secs
Line 121 TIME_OPERATION findPerfects(20000) (size = 20000) completed in 0.536 secs
Line 122 TIME_OPERATION findPerfects(40000) (size = 40000) completed in 1.615 secs

Passed 6 of 6 tests. Bravo!
```

A: 搜索到4000之前，共有4个完全数，分别是6, 28, 496, 8128

Tests are grouped by filename or by type.  
Select the test groups you wish to run:  
-----  
0) None (instead execute ordinary main() function)  
1) From PROVIDED\_TEST  
2) From perfect.cpp  
3) From soundex.cpp  
4) All of the above tests  
Enter your selection: 0  
No tests selected.  
  
Found perfect number: 6  
Found perfect number: 28  
Found perfect number: 496  
Found perfect number: 8128  
...  
Done searching up to 40000  
  
main() completed.

Qt 5.3.2 is available. Check the [Qt blog](#) for details.

## 练习TIME\_OPERATION

添加 4 个**TIME\_OPERATION**测试案例。要求是，每个测试案例的数据量翻倍，并保证最后一个测试案例的执行时间在一分钟左右。

```
// TODO: add your student test cases here
// 添加 4 个TIME_OPERATION 测试案例。
// 要求是，每个测试案例的数据量翻倍，
// 并保证最后一个测试案例的执行时间在一分钟左右。
STUDENT_TEST("Time trials of findPerfects on doubling input sizes") {
    TIME_OPERATION(20000, findPerfects(20000));
    TIME_OPERATION(80000, findPerfects(80000));
    TIME_OPERATION(240000, findPerfects(240000));
    TIME_OPERATION(320000, findPerfects(320000));
}
```

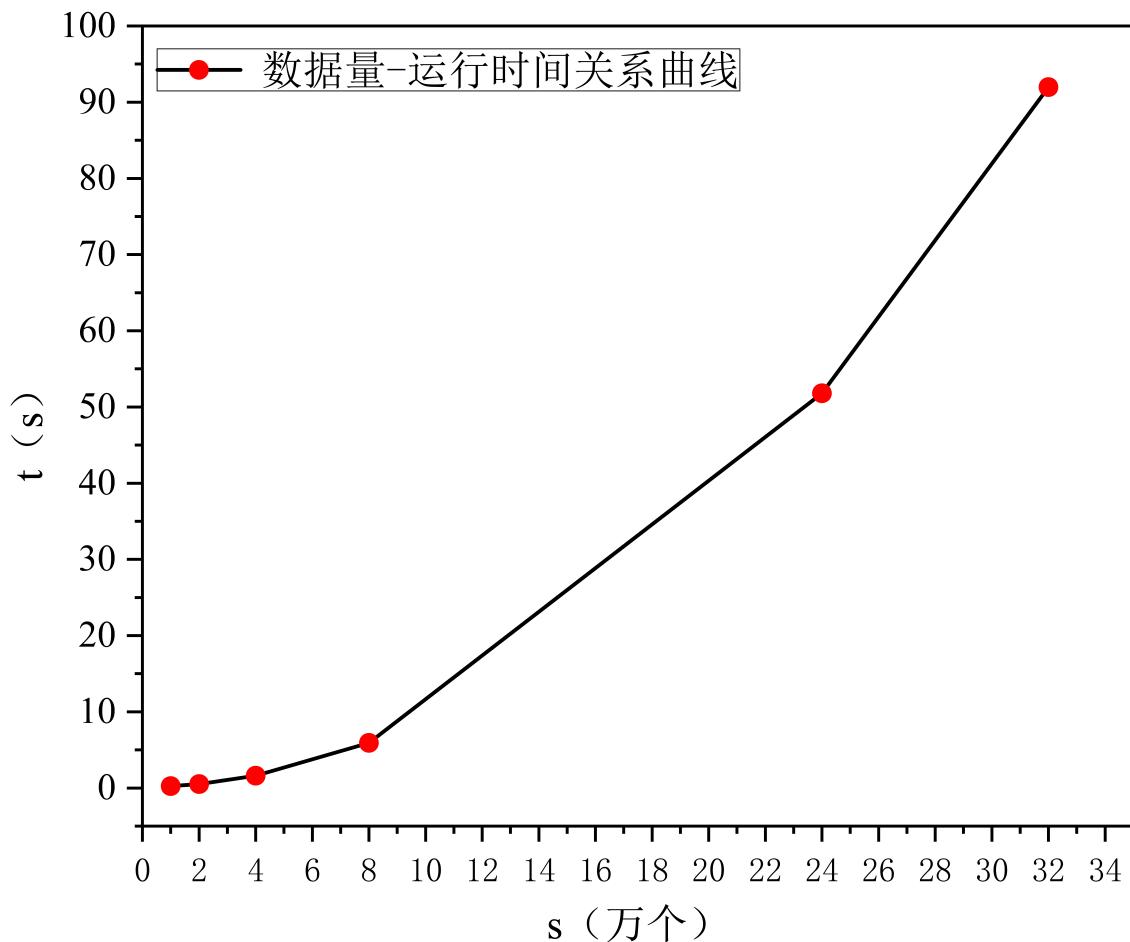
```
Correct (STUDENT_TEST, perfect.cpp:129) Time trials of findPerfектs on doubling input sizes
Line 130 TIME_OPERATION findPerfектs(20000) (size = 20000) completed in 0.544 secs
Line 131 TIME_OPERATION findPerfектs(80000) (size = 80000) completed in 5.925 secs
Line 132 TIME_OPERATION findPerfектs(240000) (size = 240000) completed in 51.815 secs
Line 133 TIME_OPERATION findPerfектs(320000) (size = 320000) completed in 91.981 secs

Passed 1 of 1 tests. Good on ya!
```

## Q2. 在表格中记录 `findPerfектs` 的执行时间。

A: 绘制数据量大小及其运行时间关系曲线

size	time(s)
10000	0.265
20000	0.536
40000	1.615
80000	5.925
240000	51.815
320000	91.981



**Q3. 对于 10 与 1000，计算 `isPerfect` 是否做了同样多的工作？为什么一样多，或者为什么不一样多？对于搜索范围为 1-1000 的数，`findPerfect` 其工作量是否与 1000-2000 时一致？为什么一样多，或者为什么不一样多？**

A: 对于10与1000，函数 `isPerfect` 工作量一致，这是由于该函数是一个bool函数，返回true或false，对于特定的整数输入没有区别。

对于搜索范围为1-1000的数，函数 `findPerfect` 中有循环的功能，这使得其的工作量与1000-2000时不一致。这是因为1000之前完全数有3个，而第4个完全数是8128，其间整数的数量大于1000个，穷举法需要搜索每一个数据，增大搜索范围后这需要很长时间。

## Q4. 根据你收集到的数据进行推测：找出第五个数 findPerfect 要花费多少时间？

根据Q2的数据量-时间曲线，前4个完全数包含在10000以内，测试的时间为0.265s。第5个完全数为33550336，所需数据量为34000000，用二阶函数拟合曲线得到 $y = 0.0896x^2 + 0.0008x + 0.177$ ，代入 $x = 3400$ ，算得 $y = 1035778.897s \approx 12\text{天}$

## Q5. 这个函数测试失败时，其他测试是否还能通过？为什么？

函数 `divisorSum` 测试失败时，其他有些测试还是可以通过，原因是通过的这些测试的调用方式不同于未通过的测试：`EXPECT(!function)` 形式的测试可以通过，而 `EXPECT(function)` 形式的测试不能通过。  
ps：我还不知道这两种调用方式的区别，所以不知道本质原因。

## Q6. 描述一下你的测试策略。

先对1进行判断，输出为0；其他数都可被1整除，total初值为1，除数从2起，与待判断数字的平方根比较。

若可被除数整数，则将除数求和后再加除法的结果，如果为完全平方数，则需减去一个重复的除数。

## Q7. 在表格中记录 `findPerfектsSmarter` 的执行时间。

size	time(s)
20000	0.204
80000	0.363
240000	0.970
320000	1.312
64000000	75.322

SimpleTest CppLegs

## Tests from STUDENT\_TEST

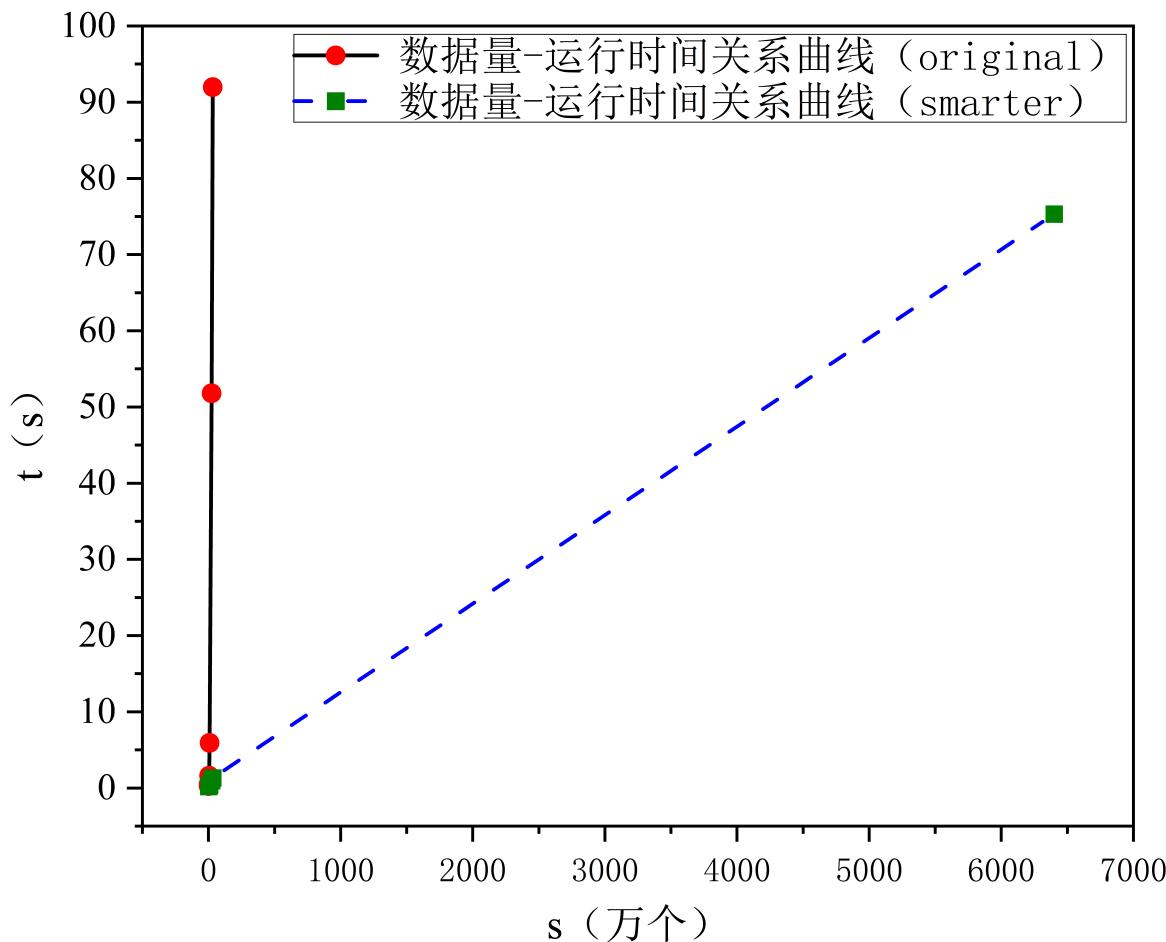
**Correct** (STUDENT\_TEST, perfect.cpp:166) Test minus: return 'false'

**Correct** (STUDENT\_TEST, perfect.cpp:172) Confirm smarterSum of small inputs

**Correct** (STUDENT\_TEST, perfect.cpp:182) Time trials of findPerfектsSmarter on doubling input sizes

```
Line 183 TIME_OPERATION findPerfектsSmarter(20000) (size = 20000) completed in 0.204 secs
Line 184 TIME_OPERATION findPerfектsSmarter(80000) (size = 80000) completed in 0.363 secs
Line 185 TIME_OPERATION findPerfектsSmarter(240000) (size = 240000) completed in 0.970 sec
Line 186 TIME_OPERATION findPerfектsSmarter(320000) (size = 320000) completed in 1.312 sec
Line 187 TIME_OPERATION findPerfектsSmarter(6400000) (size = 6400000) completed in 75.322 se
```

Passed 3 of 3 tests. Love it!



## Q8. 推测： `findPerfectsSmarter` 找到第 5 个完全数需要多久？

根据Q7的数据量-时间曲线，前4个完全数包含在10000以内，测试20000个数据的时间为0.204s。第5个完全数为33550336，所需数据量为34000000，用线性函数拟合曲线得到 $y = 0.0117x + 0.5188$ ，代入 $x = 3400$ ，算得 $y = 40.2988$ s

## Q9. 阐述下你选择的测试案例的动机，解释下为什么你认为这几个测试可以确定函数 `findNthPerfectEuclid` 正常工作。

前四个完全数已经在之前得到了，用已知的结论做测试很正常，也很容易检查出函数的正确性

## Part II 姓氏编码检索程序

### Q10. "Angelou" 的编码是多少呢？

1. A-0 N-5 G-2 E-0 L-4 O-0 U-0 **0520400**
2. 合并相邻重复数字 **052040**
3. 首字母大写替换第一个数字 **A52040**
4. 删除所有0 **A524**
5. 填充0保证编码为四位 **A524**

### Q11. 在写代码之前，自行进行头脑风暴，尝试把这个任务分解成几个小任务。简单描述一下你的分解过程。

0. 去除非字母字符
1. 大写
2. 字母编码化处理
3. 相邻重复数字的处理
4. 大写的首字母替换无相邻重复编码的第一位
5. 删除所有的0
6. 计算编码位数，不足四位最后补充0，超过四位删去四位之后的数字