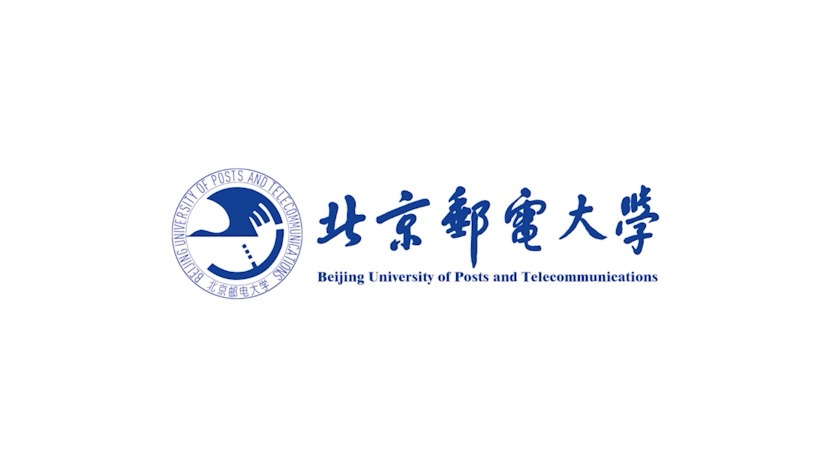
**算法设计与分析实验报告**



实验题目：最大子段和三种实现算法的时间复杂度分析

姓名： 马天成

学号： 2020211376

日期： 2022-09-27

# 一、实验环境

## 1.1 设备规格

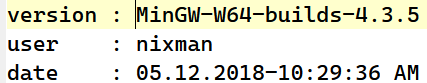


## 1.2 操作系统



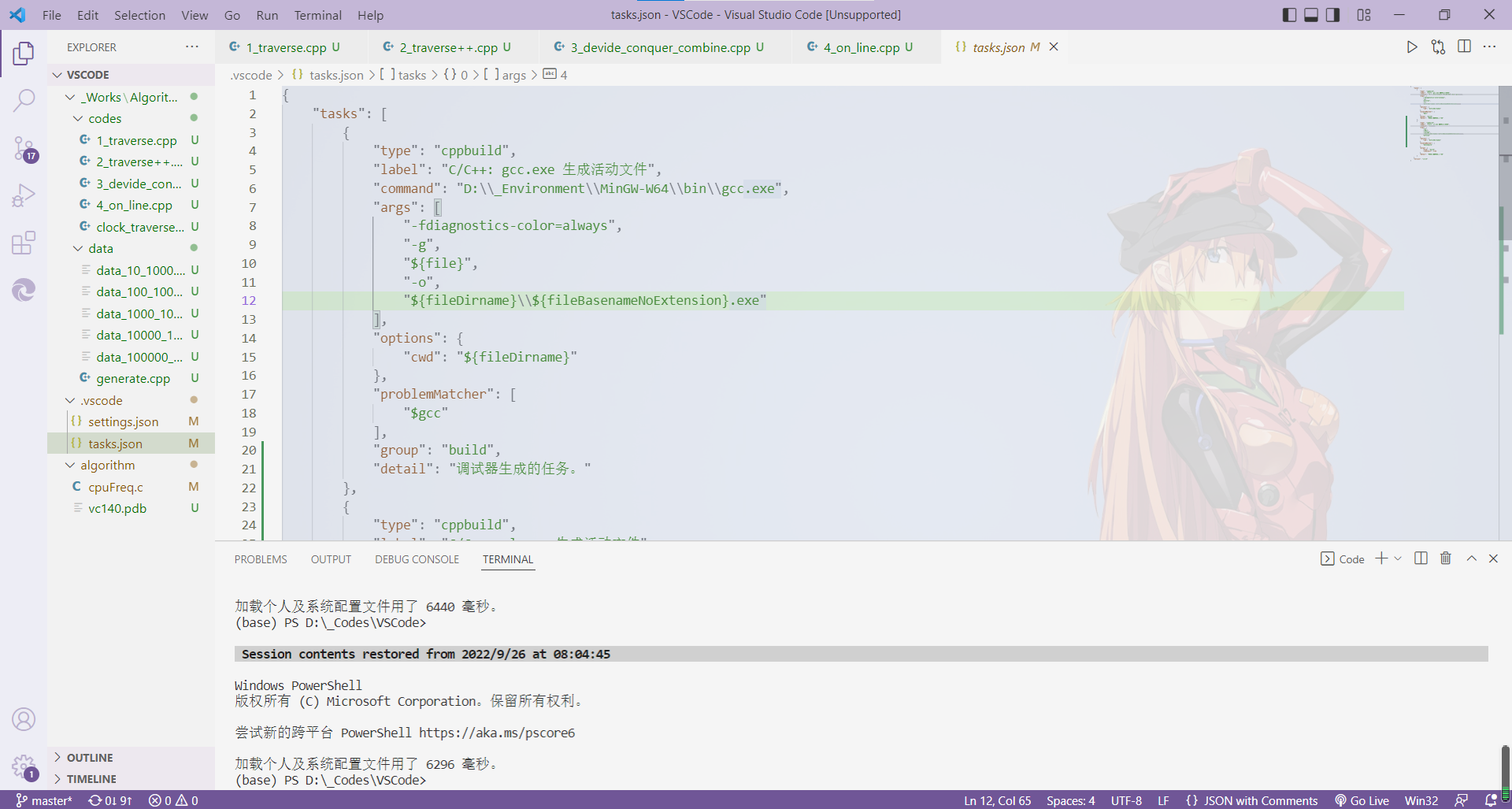
## 1.3 编程语言&编译器

C++11



## 1.4 开发工具

VSCode



# 二、实验内容

## 2.1实验要求

分别实现课件中给出的基于暴力枚举、分治法和动态规划方法实 现的最大字段和算法程序，测试在不同数据规模下，三种算法的时间复杂度。

## 2.2 实验目的

1. 理解算法时间复杂度的评价方法，初步了解不同的算法策略对算 法性能的影响程度；

2. 掌握算法时间复杂度测试的基本流程。

## 2.3 实验内容

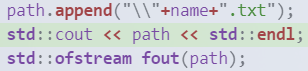
### 2.3.1 数据准备

进行数据的生成，采用随机数的思路，生成 “dataSize \n data\_1 data\_2 …”的文件形式：

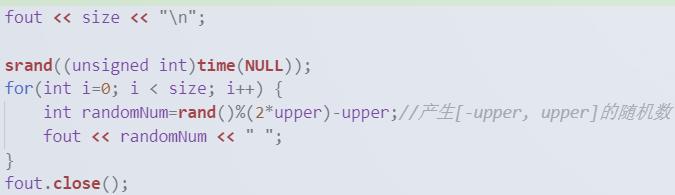


* 进行当前path的读取，以便生成文件
* 进行数据规模，数据random，文件名称的读取（从io）
* 生成数据，写入文件：

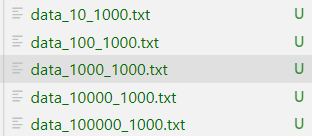
1. 生成文件路径并打开文件：



2. 写入dataSize和data并关闭文件：

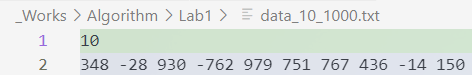


3. 根据实验要求，选择实现5个文件；数据范围都在[-1000, 1000]



命名格式：dtat\_dataSize\_dataRandom.txt

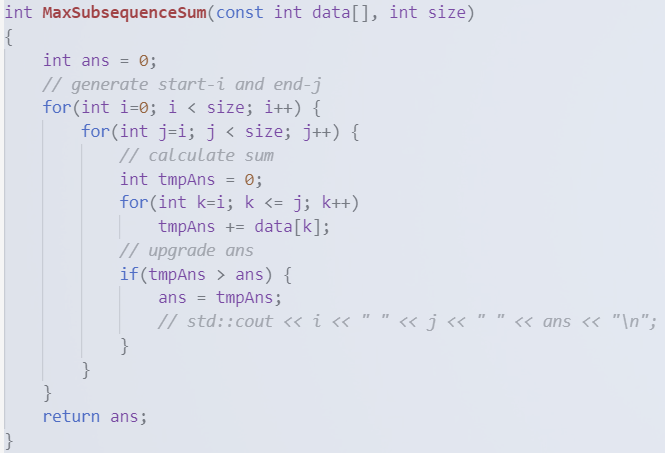
形如：data\_10\_1000.txt



## 2.2 代码编写

根据精简原则，我们这里只展示运算函数部分，不考虑主函数。

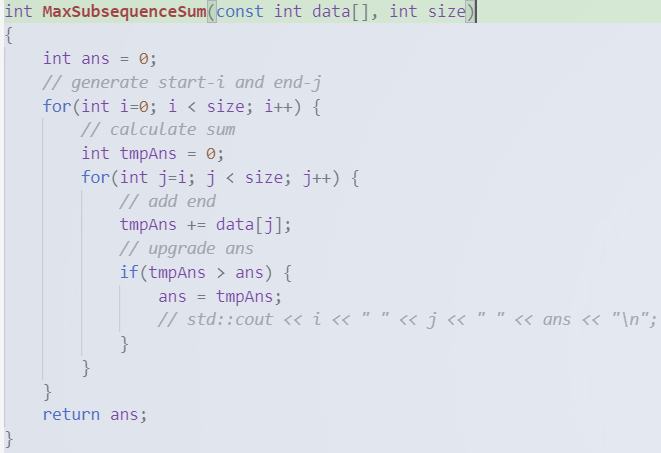
### 2.2.1 暴力枚举



非常简单的三重循环。按照课上所说，枚举每一个可能出现的start和end区间，计算之间的字段和，获得最大值。

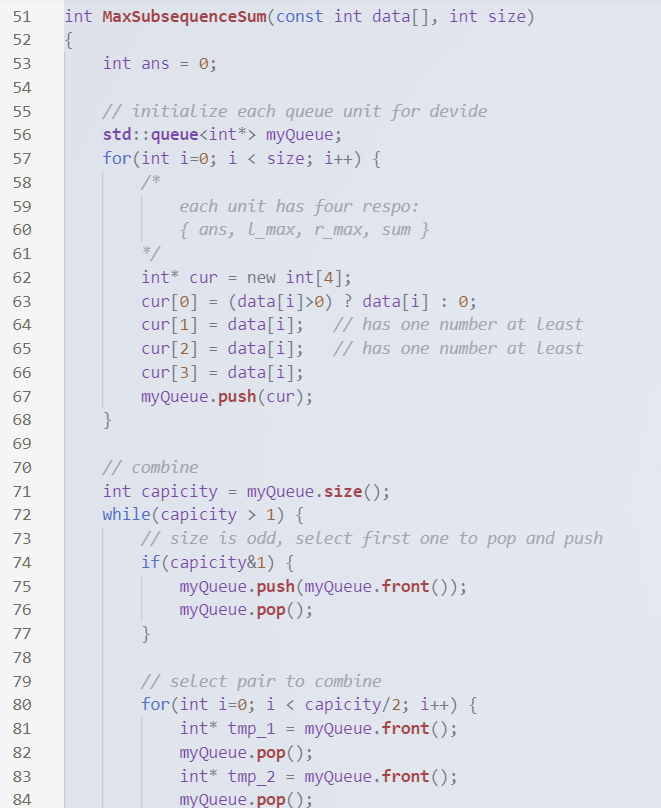
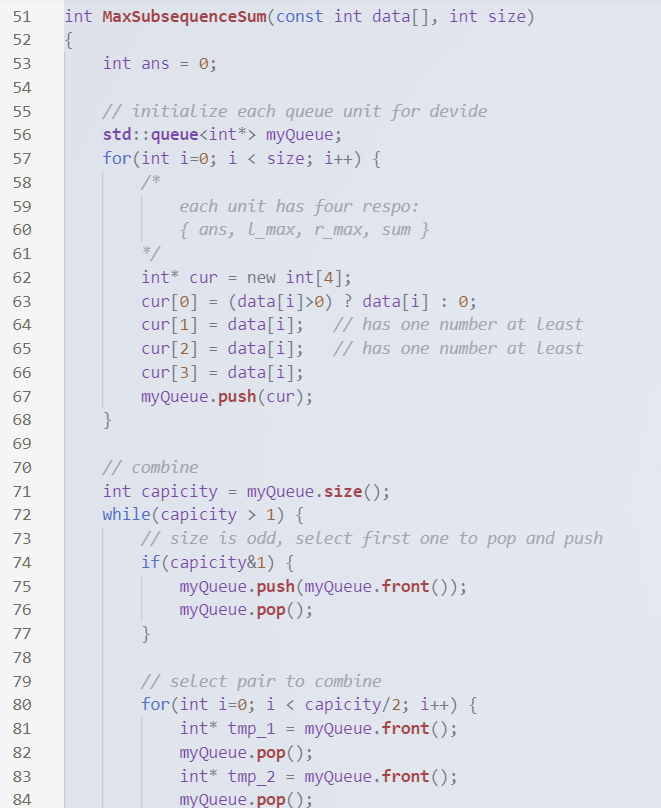
只不过这里需要多用一个int存储空间存储当前循环计算得出的字段和，并且在循环中一直维护最大值ans，最后在枚举完成后返回ans即可。

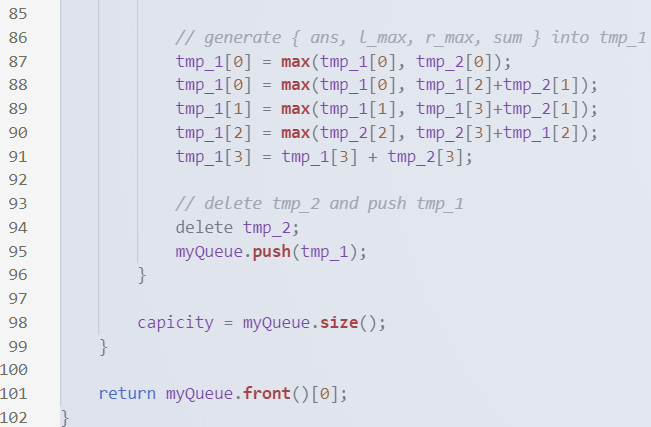
### 2.2.2 暴力枚举优化



减小一层循环，用之前短一个的子段和来算出现在新的子段和，充分利用了之前子段和的计算结果，把复杂度从O(n^3)到O(n^2)。

### 2.2.3 分治法





**ans:**

**l\_max:**

**r\_max:**

**sum:**

分治法是最巧妙的一个方法。

感谢顾佳澜同学的分享。在她的维护四个变量***{ ans, l\_max, r\_max, sum }***

的基础上,写出了这一份代码。两个单元四个属性的Combine思路如上图所示。

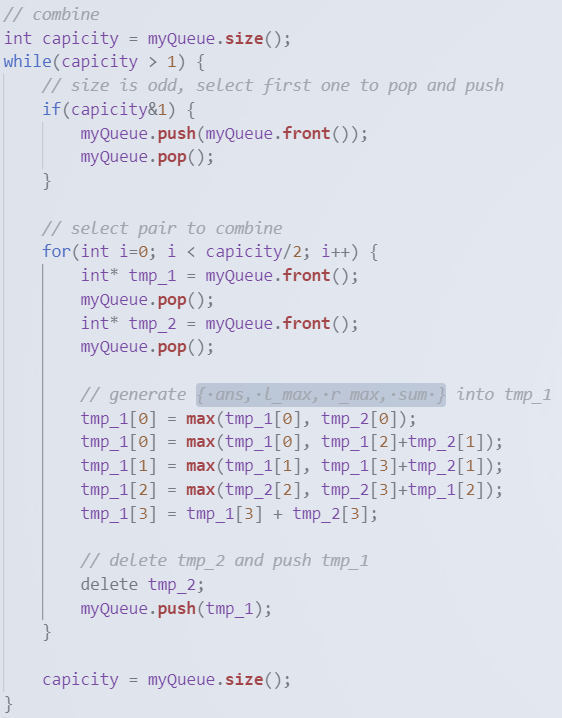
思路就是通过左右结合计算相应的四个属性，来从最小单元维护到最后整体。最小问题就是如何结合两个单元的四个属性。其解法如上图代码标注所示。

此外我加入了队列思路：

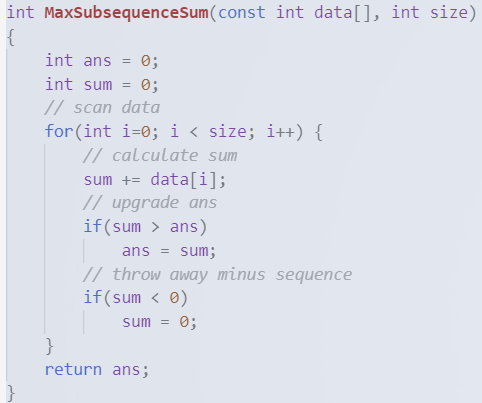
* + 在combine的时候，从queue中pop两个单元（此处为指针），然后合并两个单元的属性，再把一个覆盖新答案后的单元push进的队列，并释放另一个没用的单元空间。
  + 通过当前子段Size奇偶数的区别（奇数意味着有一个不用combine，那就先把第一个pop出来再push进去），来进行相邻子段循环叠加到最后只剩一个子段（原段）。

此外，我抛弃了vector来作为数据结构，因为vector即使erase，空间只增不减。

**combine核心代码：**



### 2.2.4 动态规划

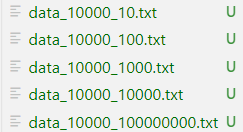


这个思路直接以最简的方式来进行计算。它建立在得出了最大子段和不需要负数字段和这一强结论，使得整个过程变得极为简单。

dp是最好的方法，只需要扫描一次，用的空间也只有一个int。

## 2.3 实践测试方式

对于随机生成的random数据可控的变量为random范围和数据量大小。所以我们先考虑random范围这个变量对于测试的影响：分别用算法4测试了以下五个文件：





本地测试采用了不同的random范围，发现影响不大，运行速度比较随机，且没有什么数量级上的差距。所以测试用例都取random=1000，可认为合理。

### 2.3.1 clock测试

要获得运行时间，就设置时间戳，在进入函数和返回函数两个地方获得时间：



一共有两次计算：

1. 进入main函数之后到打开文件并读数据到内存完毕的时间；

2. 在最大字段和函数运行完之后计算整个时间。

### 2.3.2 CPU计数器测试

看到数量级之后，我打算尝试用更精准的方法进行时间测量。鉴于之前有过做帧率的经历，我掏出了尘封已久的cpuFreq代码。通过CPU计数单元来获得最精准的测量方式。



按照相同的思路，我们在相同的位置采用时间戳获得。但是通过CPU每秒计数来获得相应的运行时间。

runTime =

((endTime.QuadPart - startTime.QuadPart)\*1.0f / cpuFreq.QuadPart);

这样时间测试的精准度就会高很多。

### 2.3.3 实验记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Algorithm\_1 | Algorithm\_2 | Algorithm\_3 | Algorithm\_4 |
| 10 | 0.0005007s | 0.0004677s | 0.0006279s | 0.0004522s |
| 100 | 0.0008349s | 0.0005496s | 0.0007359s | 0.0004102s |
| 1000 | 0.318669s | 0.0018608s | 0.0004758s | 0.0003931s |
| 10000 | 310.168s | 0.160161 | 0.0016142s | 0.000649s |
| 100000 | / | 16.2342s | 0.0149988s | 0.0007848s |

10：

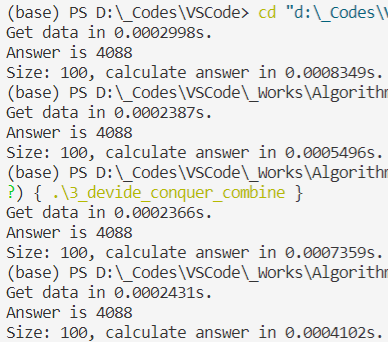




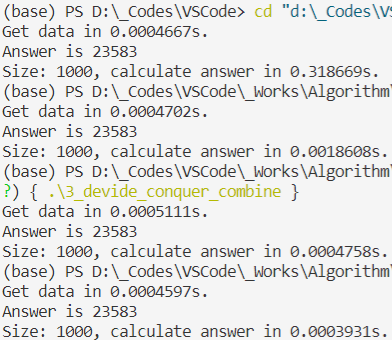




100：



1000：



10000：









100000:



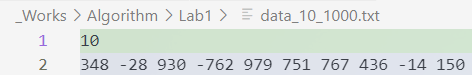




# 三、出现问题及解决

**Q1. 生成随机测试数据文件的问题**

首先是生成数据用什么形式。最终采用如下文件形式：



其次是随机数生成数据溢出的问题：



这个式子可以说是非常想当然的一个式子。完全没有考虑到数据类型。当输入的upper=100000000时，文件根本就是全负数。

**Q2. 分治算法如何确定维护的属性，以及combine**

非常感谢董佳澜同学的分享，要不然我真想不出来。虽然想法比较简单，但combine这种思维方式对于我来说确实是比较新颖。

**Q3. 如何测试**

对于数据集的变量把控以及对于测试的情况分析（上文均有），在此前并不是很熟悉。但在经过本次实验后，我也知道了如何去测试，如何去设计测试，以及如何用限定变量的方法来分析变量的影响度。

**Q4. 没有考虑到多样例测试**

导致测试文件名是在程序中写死，每次调用程序测试不同数据集都需要重新编译运行。这类情况以后一定要避免。

# 四、总结

## 4.1 时间和空间复杂度分析

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Algorithm\_1 | Algorithm\_2 | Algorithm\_3 | Algorithm\_4 |
| 时间复杂度 | O(n^3) | O(n^2) | O(n\*logn) | O(n) |
| 空间复杂度 | O(1) | O(1) | O(n) | O(1) |

对于这4个算法来说，空间复杂度应该是确定的，但时间复杂度反映出来的依旧有值得探讨的地方：

算法三采用了比较复杂的数据结构。所以再数据规模小的时候，还不一定能跑过算法一算法二。但是数据规模变大之后，这一点开销就会比数据规模带来的影响更小了。

## 4.2 实验经验总结

1. 了解了优化算法的重要性：在数据量庞大的情况下，不同的时间复杂度会导致极大的效率差别。

2. 算法需要循序渐进。在这四种算法中，我认为1-2-4是循序渐进的：

* 建立在顺序读取枚举计算的算法一上，通过小子段推大子段这一技巧，使得时间复杂度降了一个量级；
* 在顺序读取的思路中，我们限定了一个已经推出的强结论，使得算法再次降了一个复杂度。
* 分治算法反而像是插进来的，它不是顺序读取的思路。它属于分治思想，但由于时间复杂度所以将它放在算法3。

3. 算法复杂度测试的基本流程：

* 生成不同规模不同性质的数据集，对算法进行全方面的测试。
* 比较不同算法的复杂度，尤其在大规模数据集下的时间，分析优劣。（在本次实验中，分治法在很大程度上有一部分数据结构的开销，所以小规模数据中算法3甚至不如算法1算法2）