

**数据结构课程设计**

**实验报告**



题 目 大整数运算

学 号 202103151211

姓 名 江银

班 级 计算机实验班01

提交日期 2022.12.25

**目 录**

[一． 实验的题目和要求 - 3 -](#_Toc123161536)

[1.1题目描述 - 3 -](#_Toc123161537)

[1.2 实验要求 - 3 -](#_Toc123161538)

[二． 设计思路 - 3 -](#_Toc123161539)

[2.1系统总体设计 - 3 -](#_Toc123161540)

[2.1.1 系统的数据结构 - 3 -](#_Toc123161541)

[2.1.2 系统的技术思路 - 4 -](#_Toc123161542)

[2.2 系统功能设计 - 4 -](#_Toc123161543)

[2.3 类的设计 - 4 -](#_Toc123161544)

[2.3.1 双向链表类 - 4 -](#_Toc123161545)

[2.3.2 大整数类 - 6 -](#_Toc123161546)

[2.4 主程序的设计 - 13 -](#_Toc123161547)

[三． 调试分析 - 14 -](#_Toc123161548)

[3.1 技术难点分析 - 15 -](#_Toc123161549)

[3.1.1 快速傅里叶变换 - 15 -](#_Toc123161550)

[3.2 调试错误分析 - 15 -](#_Toc123161551)

[3.2.1 文件读写中输出文件的覆盖问题 - 15 -](#_Toc123161552)

[3.2.2 除法运算与减法运算耦合问题 - 16 -](#_Toc123161553)

[3.2.3 减法运算的返回值问题 - 16 -](#_Toc123161554)

[四． 测试结果分析 - 17 -](#_Toc123161555)

[4.1 测试常规数据 - 17 -](#_Toc123161556)

[4.1.1 控制台输入输出测试 - 17 -](#_Toc123161557)

[4.1.2 文件输入输出测试 - 19 -](#_Toc123161558)

[4.2 测试边界数据 - 20 -](#_Toc123161559)

[4.2.1 非法数据测试 - 20 -](#_Toc123161560)

[4.2.2 极端数据测试 - 20 -](#_Toc123161561)

[4.3 测试性能 - 23 -](#_Toc123161562)

[五． 附录 - 24 -](#_Toc123161563)

1. **实验的题目和要求**

## 1.1题目描述

密码学分为两类密码：对称密码和非对称密码。对称密码主要用于数据的加/ 解密，而非对称密码则主要用于认证、数字签名等场合。非对称密码在加密和解密时，是把 加密的数据当作一个大的正整数来处理，这样就涉及到大整数的加、减、乘、除和指数运算 等，同时，还需要对大整数进行输出。请采用相应的数据结构实现大整数的加、减、乘、除 和指数运算，以及大整数的输入和输出。

## 1.2 实验要求

1. 要求采用链表来实现大整数的存储和运算，不允许使用标准模板类的链表类(list)和函数。同时要求可以从键盘输入大整数，也可以文件输入大整数，大整数可以输出至显示器，也可以输出至文件。大整数的存储、运算和显示，可以同时支持二进制和十进制， 但至少要支持十进制。大整数输出显示时，必须能清楚地表达出整数的位数。测试时， 各种情况都需要测试，并附上测试截图；要求测试例子要比较详尽，各种极限情况也要 考虑到，测试的输出信息要详细易懂，表明各个功能的执行正确。

1. 要求大整数的长度可以不受限制，即大整数的十进制位数不受限制，可以为十几位的整 数，也可以为 500 多位的整数，甚至更长；大整数的运算和显示时，只需要考虑正的大整数。如果可能的话，请以秒为单位显示每次大整数运算的时间。
2. 要求采用类的设计思路，不允许出现类以外的函数定义，但允许友元函数。主函数中只 能出现类的成员函数的调用，不允许出现对其它函数的调用。
3. 要求采用多文件方式：.h 文件存储类的声明，.cpp 文件存储类的实现，主函数 main 存 储在另外一个单独的 cpp 文件中。如果采用类模板，则类的声明和实现都放在.h 文件中。
4. 不强制要求采用类模板，也不要求采用可视化窗口；要求源程序中有相应注释。
5. **设计思路**

## 2.1系统总体设计

## 2.1.1 系统的数据结构

系统要实现的目标是使用链表完成大整数的各种运算，我们需要实现两个类，一个链表类和一个大整数类。为了后续具体操作的方便，链表使用双向链表，而大整数类通过链表类来储存大整数本身，并且需要实现大整数的加减乘除幂的基本操作，同时也需要一些比如比较、交换的辅助操作。

## 2.1.2 系统的技术思路

系统总体需要完成加法、减法、乘法、除法、指数运算五种基本操作。

首先是加法和减法运算。通过模拟竖式的算法进行编写，总体是O(n)的线性时间复杂度，和数据规模属于同一级别，优化空间不大，算法本身较为理7想。

乘法运算的实现一般来说有三种。第一种是直接模拟竖式进行运算，时间复杂度为O(n^2)，实现难度较低，而且在数据规模较小的时候效率也比较可观；第二种是分治乘法，即Karatsuba乘法，通过递归实现O(n^1.585)的时间复杂度；第三种是使用快速傅里叶变换(FFT)算法，算法本身是用来计算多项式卷积，但是可以通过根据十进制处理进位，来进行高精度乘法的运算，算法复杂度很优秀，为O(nlogn)，但是由于常数比较大，在处理小规模数据时无法体现其优越性。在本系统中，使用了第一种和第三种算法进行实现。

除法运算的实现一般来说也有三种。第一种是直接模拟竖式进行运算，每一次对除数进行试减，时间复杂度为O(n^2)，并且带有一定常数，但是可以同时求出商和余数；第二种是二分法，收敛次数是O(n)，最坏的时间复杂度也为O(n^2)；第三种是牛顿迭代法，这种算法有O(logn)的收敛速度，是相当快的，如果结合FFT进行迭代时乘法的运算，可以达到O(nlogn)的整除时间复杂度。但是牛顿迭代法需要高精度浮点数的支持，代码实现难度较大。在本系统种，使用了第一种算法进行实现。

指数运算使用快速幂。系统同时提供了分别与两种乘法算法结合的幂运算算法。

## 2.2 系统功能设计

该系统主要分为链表类和大整数类两个部分。下表展示了大整数类的各个功能模块。

|  |  |  |
| --- | --- | --- |
| **系统模块** | **实现功能** | **说明** |
| 运算部分 | 实现加法、减法、乘法、除法、指数运算 | 使用相应的成员函数实现 |
| 辅助运算部分 | 实现交换、比较、判零的成员函数 | 方便运算部分具体算法的实现 |
| 输入输出部分 | 从控制台和文件输入输出大整数的内容 | 提供了多种输入输出的成员函数 |

## 2.3 类的设计

下面展示了双向链表类和大整数类的具体设计。

## 2.3.1 双向链表类

双向链表类主要实现了一些链表数据结构的基本操作。链表类是这个系统中容器的角色，是大整数类进行实现的基础。

|  |  |  |
| --- | --- | --- |
| **实现功能** | **说明** | **具体代码** |
| 链表的结点类设计 | 分别有前驱指针、后继指针、结点值等属性。 |  |
| 链表的初始化 | 空链表由两个结点组成，长度为0。 |  |
| 字符串转换成链表 | 大整数的操作中需要用到将读入的字符串转化成链表。代码中的MAX\_LEN是原定的将所有大整数的位数取成固定值以进行取模的用途，现MAX\_LEN为0，固定不进行取模。 |  |
| 链表在头部和尾部添加元素 | 添加元素的同时要将链表长度加1，且仅有这个操作可以增加链表的长度。 |  |
| 返回链表的长度以及链表的判空 | 链表的长度即为len。 |  |
| 链表元素的输出 | 在控制台的输出 |  |
| 在文件中的输出，文件打开为app模式，在原来的文件内容上追加。 |  |

## 2.3.2 大整数类

大整数类是这个系统的核心部分，实现了加法、减法、乘法、除法、指数运算。

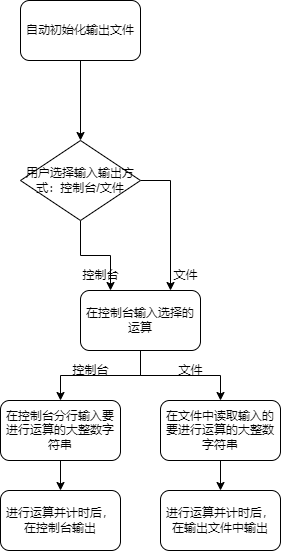
|  |  |  |
| --- | --- | --- |
| **实现功能** | **说明** | **具体代码** |
| 加法运算 | 加法运算部分采用了模拟加法竖式运算的算法。首先将两个参与加法运算的大整数的高位用0补齐。然后从低位开始两两相加，依次进位，保存到新的大整数res中。最高位若有进位，将最后的进位加入到res中去。时间复杂度为线性的O(n)。 |  |
| 减法运算 | 减法运算部分采用了模拟减法竖式运算的算法。首先将两个参与减法运算的大整数的高位用0补齐。然后用cmp函数判断被减数和减数的大小，得到最终运算结果的正负号。如果被减数更小，则需要交换两个数，得到大数减小数的运算结果。整个函数的返回值分为正负号和绝对值。最后交换回来是为了接下来的除法运算中的操作。在算法中，用borrow表示借位。时间复杂度为O(n)。 |  |
| 比较两个大整数之间的大小 | 在执行cmp这个成员函数之前，确保了参与运算的两个大整数的位数相同，且均为正数。从最高位开始比较，如果两数该位不同，就可以判断出它们的大小并退出。如果第一个数较大，就返回1；如果第二个数较大，就返回-1；如果两者相等，则返回0。时间复杂度为O(n)。 |  |
| 交换两个大整数 | 在执行swap这个成员函数之前，确保了参与运算的两个大整数的位数相同，且均为正数。因此将两个大整数对应的每一位直接进行交换。时间复杂度为O(n)。 |  |
| 朴素乘法运算 | 朴素乘法运算采用模拟乘法竖式运算。从卷积的角度来看，乘积得到的结果最多为n+m位。从低位看起，其第i位的结果是所有满足i=j+k的第一个数的第j位和第二个数的第k位的乘积之和。得到卷积形式后，从低位到高位，模10来进行进位操作。时间复杂度为O(n^2)级别。 |  |
| 除以2的运算 | 这是一个脱胎于高精度除法的除以2运算，主要为快速幂服务。模拟竖式运算，从高位开始依次一位一位进行除以2的操作。时间复杂度为线性的O(n)。 |  |
| 朴素的快速幂运算 | 使用快速幂来进行大整数的指数运算。即用分治的思想将幂指数每次对半拆成两部分相乘的结果进行计算。 |  |
| 使用FFT的快速幂运算 | 和朴素的快速幂运算类似，将上面内置的朴素乘法运算改为了采用快速傅里叶变换的乘法运算。 |  |

高精度除法和使用FFT的高精度乘法较为复杂，现结合流程图详细介绍。

|  |  |
| --- | --- |
| **实现功能** | 高精度除法 |
| **具体代码** |  |
| **程序流程图** |  |
| **说明** | 模拟除法竖式进行试减，每次试减完判断是否够减，具有一定的常数。除此之外每一次操作的复杂度为O(n^2)。 |
| **实现功能** | 高精度乘法 |
| **具体代码** |  |
| **程序流程图** |  |
| **说明** | 将多项式系数形式转化成点表示形式的时间复杂度为O(logn)，将点表示形式转化回系数形式的时间复杂度也是O(logn)，而点表示形式的相乘只需要O(n)的复杂度。因为涉及很多赋值等操作，所以算法的常数很大，在数据规模较小的时候显得效率不及普通的卷积算法。 |

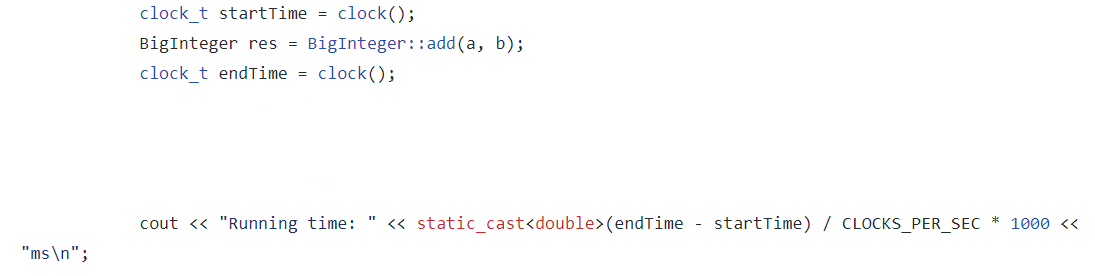
## 2.4 主程序的设计

主函数主要实现了用户交互，分别实现了控制台输入输出、文件输入输出、计算运行时间等功能。



图表 1：主函数的程序流程图

具体用来计算运行时间的函数也非常简单。记录运行算法前后的时间，最后输出前后的时间差，即为算法运行的总时长。



图表 2：计算运行时间

1. **调试分析**

## 3.1 技术难点分析

## 3.1.1 快速傅里叶变换

快速傅里叶变换是一种优雅、高效的卷积算法，在提高高精度乘法的效率上起到了不可磨灭的作用，将原本O(n^2)的高精度乘法运算的运行效率提高了几个数量级，但算法本身具有一定的难度。

根据范德蒙德行列式的性质等，可以得到一个n次多项式可以由n+1个点所唯一确定。而这n+1个点，选择的是复平面中大于这个数量的最接近的二次幂等分点。这些等分点有一些特殊的性质，比如对称性等。如果用这些点进行运算，可以在O(n)的复杂度完成用点表示的卷积运算（多项式乘法）。

快速傅里叶变换的具体操作就是将通过用系数表示的多项式转化成用点表示的多项式。首先有一个比特逆序的概念，即将01串比如“001”逆序转换成“100”，这里有一个公式：rev[i] = ((rev[i>>1]>>1))|((i&1)<<(bit-1))。然后我们通过交换把系数排成比特逆序的顺序，然后进行核心的变换部分。较为普通的变换方式是直接递归，但是递归的效率并没有这么高，所以采用更好的迭代方式。至此就完成了快速傅里叶变换部分。

得到点表示形式后，进行相乘以后就是逆变换的过程。可以证明发现，通过简单的操作可以再次通过快速傅里叶变换得到用系数表示的多项式。至此只需要将得到的系数按照模10进行进位就可以得到高精度乘法运算的结果了。

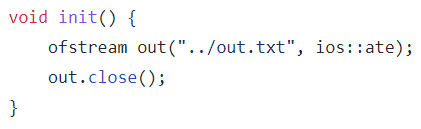
## 3.2 调试错误分析

## 3.2.1 文件读写中输出文件的覆盖问题

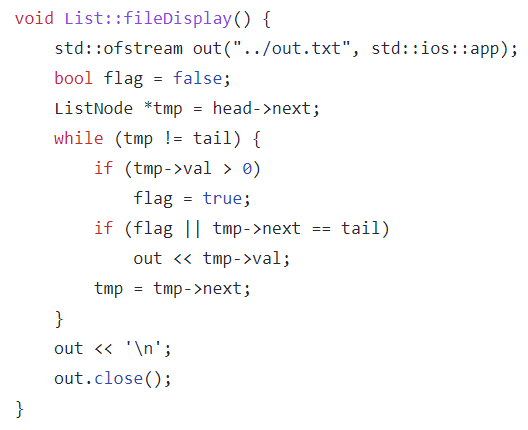
本系统要实现文件的输入输出功能。在编写完文件功能的时候进行初步测试，发现每次进行输入输出，都会把之前所有的结果覆盖掉，导致用户的交互性很差。

解决方法：在每次开始运行系统后，将输出的文件中的内容全部清空。之后每次的得到运算的结果，放入输出文件后，都在原来的基础上进行追加。

实现代码：



图表 3：初始化输出文件



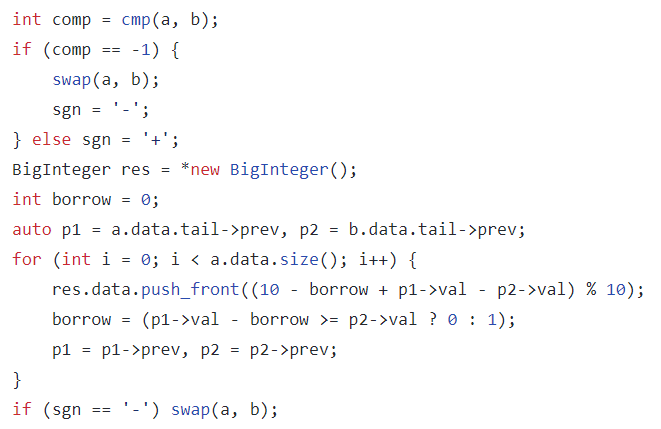
图表 4：文件输出

## 3.2.2 除法运算与减法运算耦合问题

在实现完除法运算后，发现试减过程中出现了错误。经过对变量的跟踪，发现是在不够减的情况下出现了错误。经过进一步排查，是在减法运算中出现了一些当时未曾预料的错误。

解决方法：减法运算中针对被减数比减数更小的情况下，采用了交换作差的方式。由于参数是引用的形式，所以值相应的发生了交换，导致被减数和减数实际发生了交换。因此在完成减法运算后，如果之前进行了交换，就把它交换回来。

实现代码：



图表 5：除法运算和减法运算耦合问题

## 3.2.3 减法运算的返回值问题

减法运算要处理正负号的问题，第一次写的时候，如果判断是负数的话，就直接进行输出。但是考虑到要和除法运算进行耦合，以及文件输入输出的问题，不得不把符号位保留下来进一步进行处理。

解决方法：用pair类型，解决函数保存和返回两个值的问题。同样的，除法中要返回商和余数两个变量值，也通过pair类型进行解决。

实现代码：



图表 6：减法运算的返回值问题

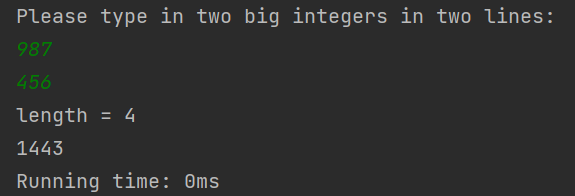
1. **测试结果分析**

## 4.1 测试常规数据

首先用常规的数据来测试五种运算。以下全部通过和python代码“print(eval(input()))”的运行结果进行对比来测试结果的正确性。

## 4.1.1 控制台输入输出测试

首先测试加法运算的结果：



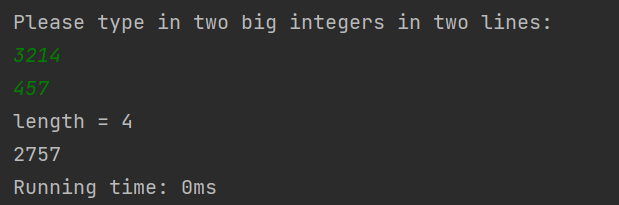
图表 7：加法运算测试结果1



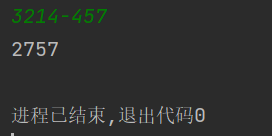
图表 8：加法运算python运行结果1

可以看到加法运算结果在常规数据下正确。

然后试着测试减法运算的结果：



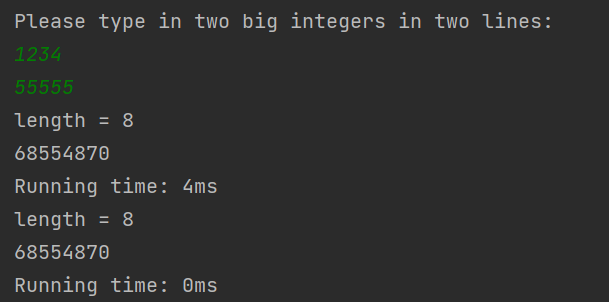
图表 9：减法运算测试结果1



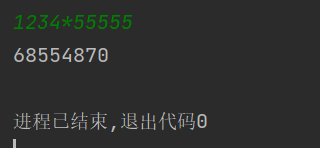
图表 10：减法运算python运行结果1

可以看到减法运算结果在常规数据下正确。

然后我们试试两种算法实现的高精度乘法。



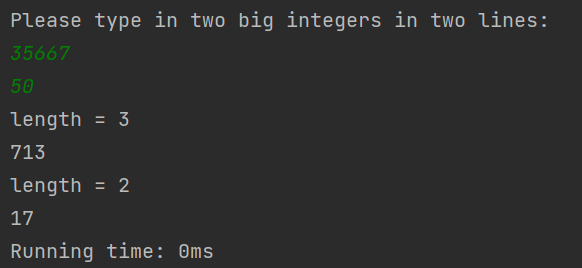
图表 11：乘法运算测试结果1



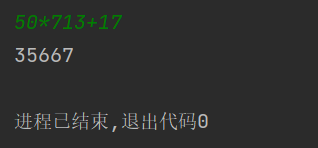
图表 12：乘法运算python运行结果1

同样的经过验证，乘法的常规数据下测试结果也是正确的。两个乘法算法第一个显示的结果是快速傅里叶变换，由于有多次数组拷贝的操作，所以在小数据情况下，效率不及朴素算法。

接着是除法运算。



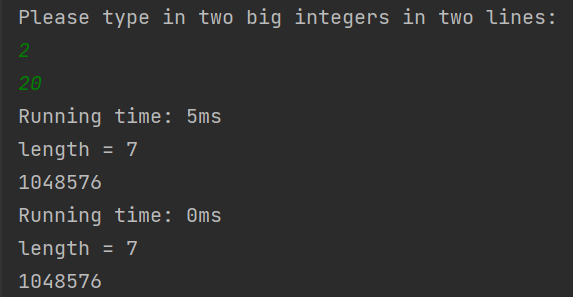
图表 13：除法运算测试结果1



图表 14：除法运算python运行结果1

经测试，得到的商和余数均正确。

最后是分别用两种高精度乘法实现的高精度求幂运算。



图表 15：指数运算测试结果1



图表 16：指数运算python运行结果1

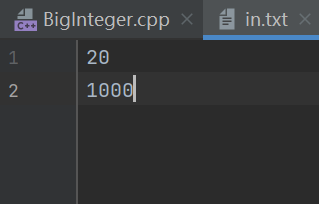
两种算法得到的指数运算结果也都是正确的。

至此，我们测试了常规的较小数据下系统的正确性，是可靠的。

## 4.1.2 文件输入输出测试

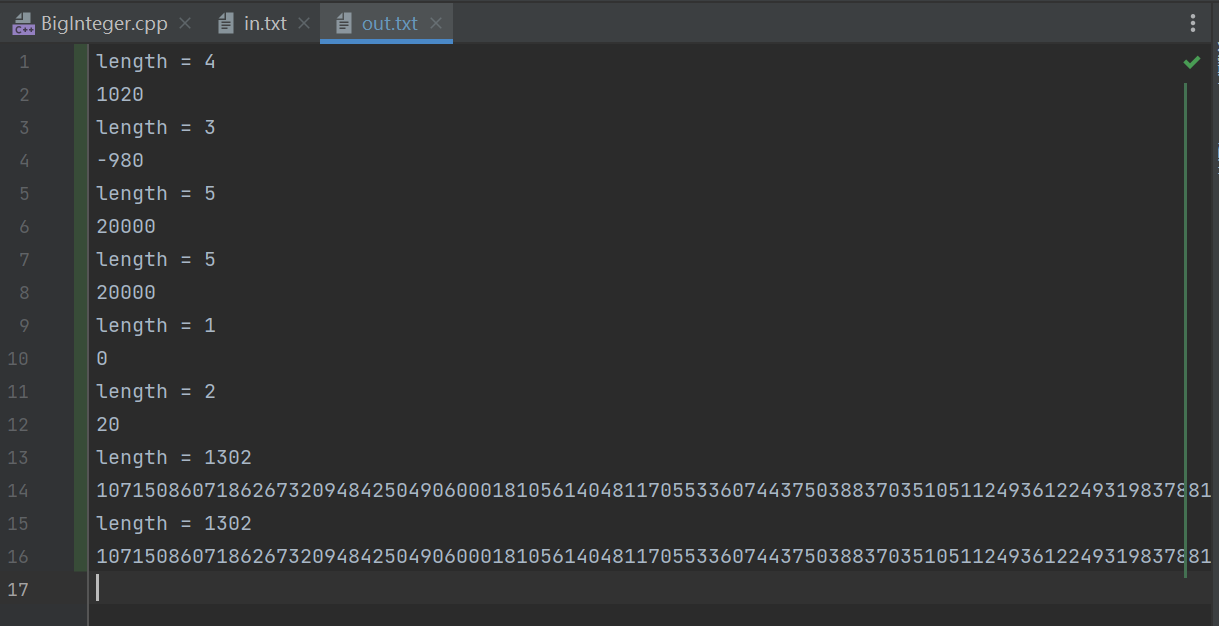
现在来测试文件输入输出功能。

在in.txt文件中写入以下内容：



图表 17：in.txt的内容

然后在控制台进行交互，依次执行加法、减法、乘法、除法、指数五种运算并输出到out.txt中。



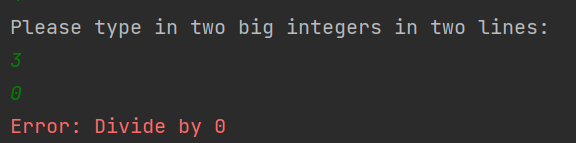
图表 18：out.txt的内容

然后在out.txt中得到了如下的内容，返回了大整数的长度而结果的值。可以看到结果符合预期。而这部分测试主要重在测试文件的功能，因此不对具体的数值正确性作进一步验证。

## 4.2 测试边界数据

## 4.2.1 非法数据测试

在大整数运算中有一些运算是非法的。比如除数为0或者0的0次幂的情况，这几种情况我们采用了特殊的方法来解决和规避。具体测试步骤与结果如下：



图表 19：除数为0的情况

返回错误信息，除数为0。



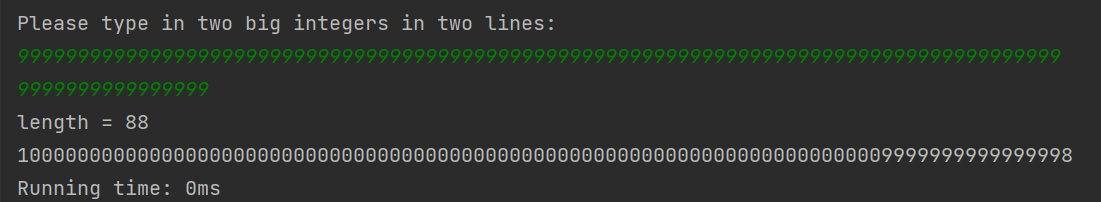
图表 20：0的0次幂的情况

返回错误信息。

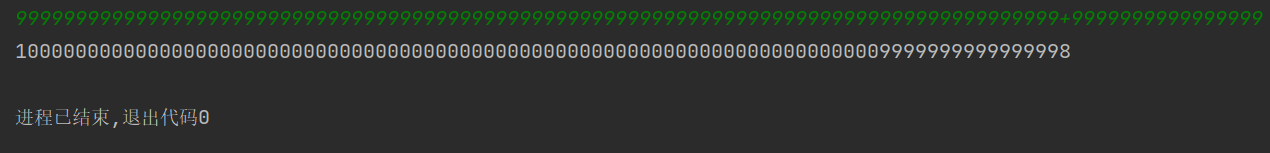
## 4.2.2 极端数据测试

这里测试一些数据较大的情况。

加法运算：

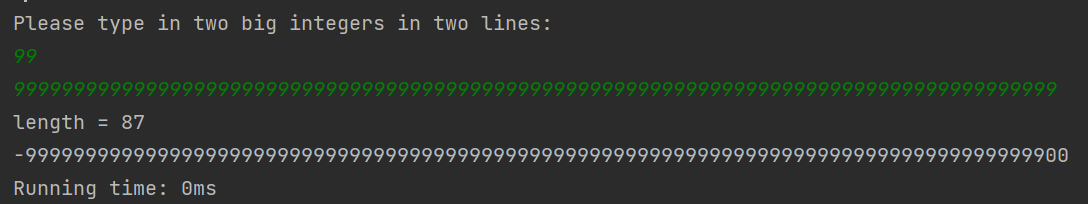


图表 21：加法运算测试结果2

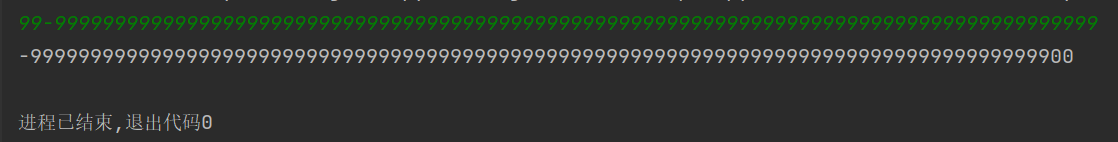


图表 22：加法运算python运行结果2

减法运算有被减数小于减数的情况，测试结果如下。

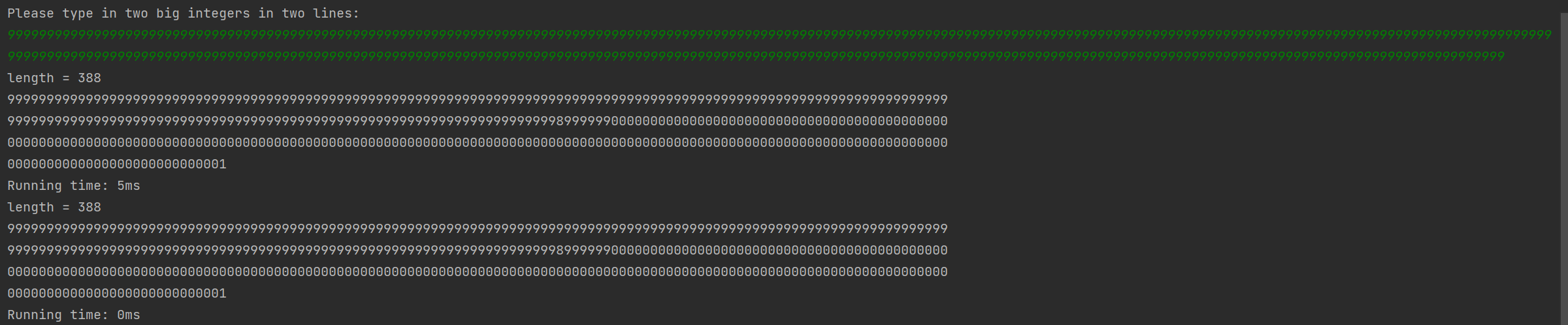


图表 23：减法运算测试结果2



图表 24：减法运算python运行结果2

乘法运算：

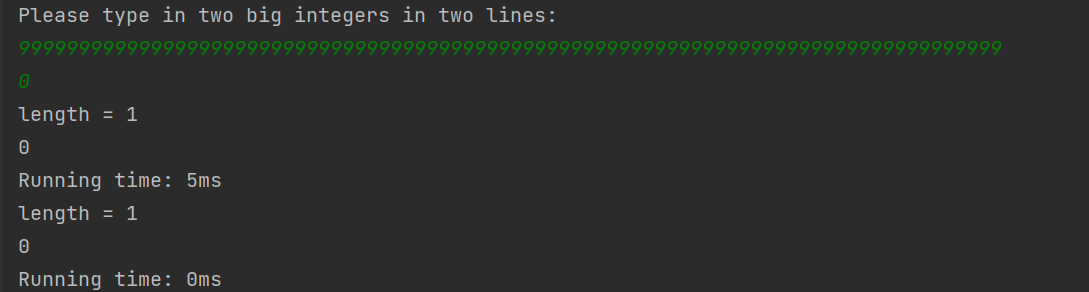


图表 25：乘法运算测试结果2

经过python运算得到的结果为：

9999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999899999900000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

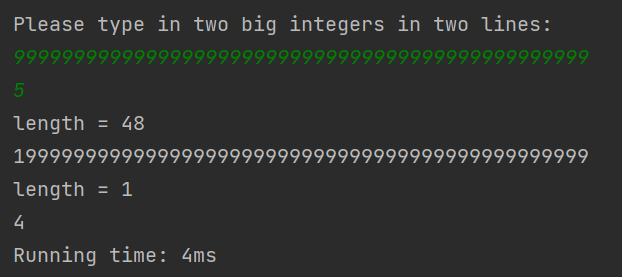
根据运行时间可知，这种数量级情况下快速傅里叶变换仍然无法体现其效率的优越性。



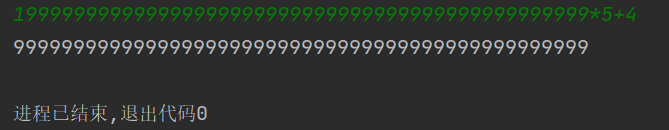
图表 26：乘法运算测试结果3

经过测试，运算结果正确。

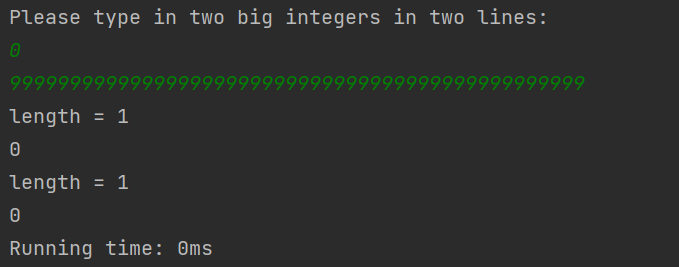
除法运算：



图表 27：除法运算测试结果2



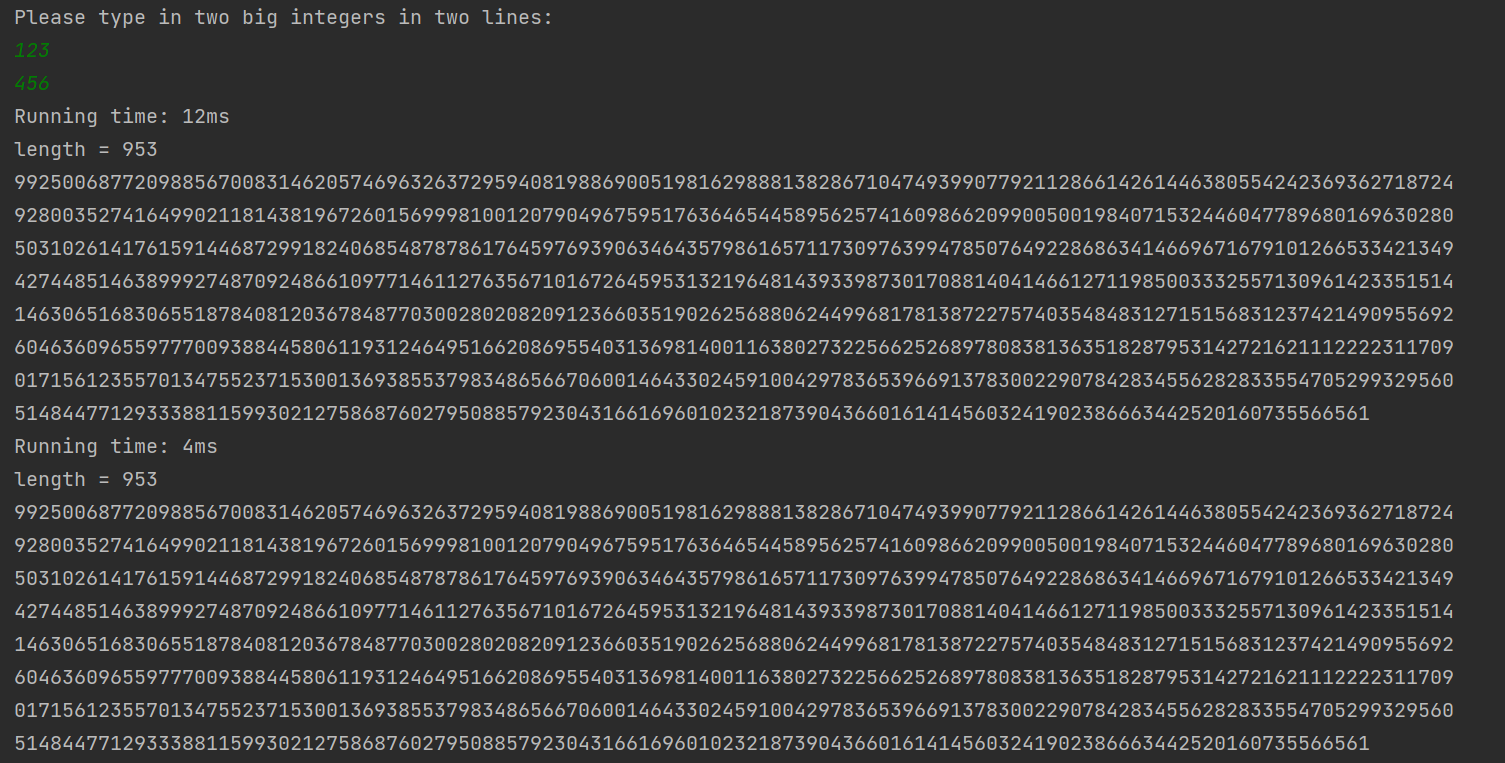
图表 28：除法运算python运行结果2



图表 29：除法运算测试结果3

经过测试，运行结果正确。

指数运算：



图表 30：指数运算测试结果2

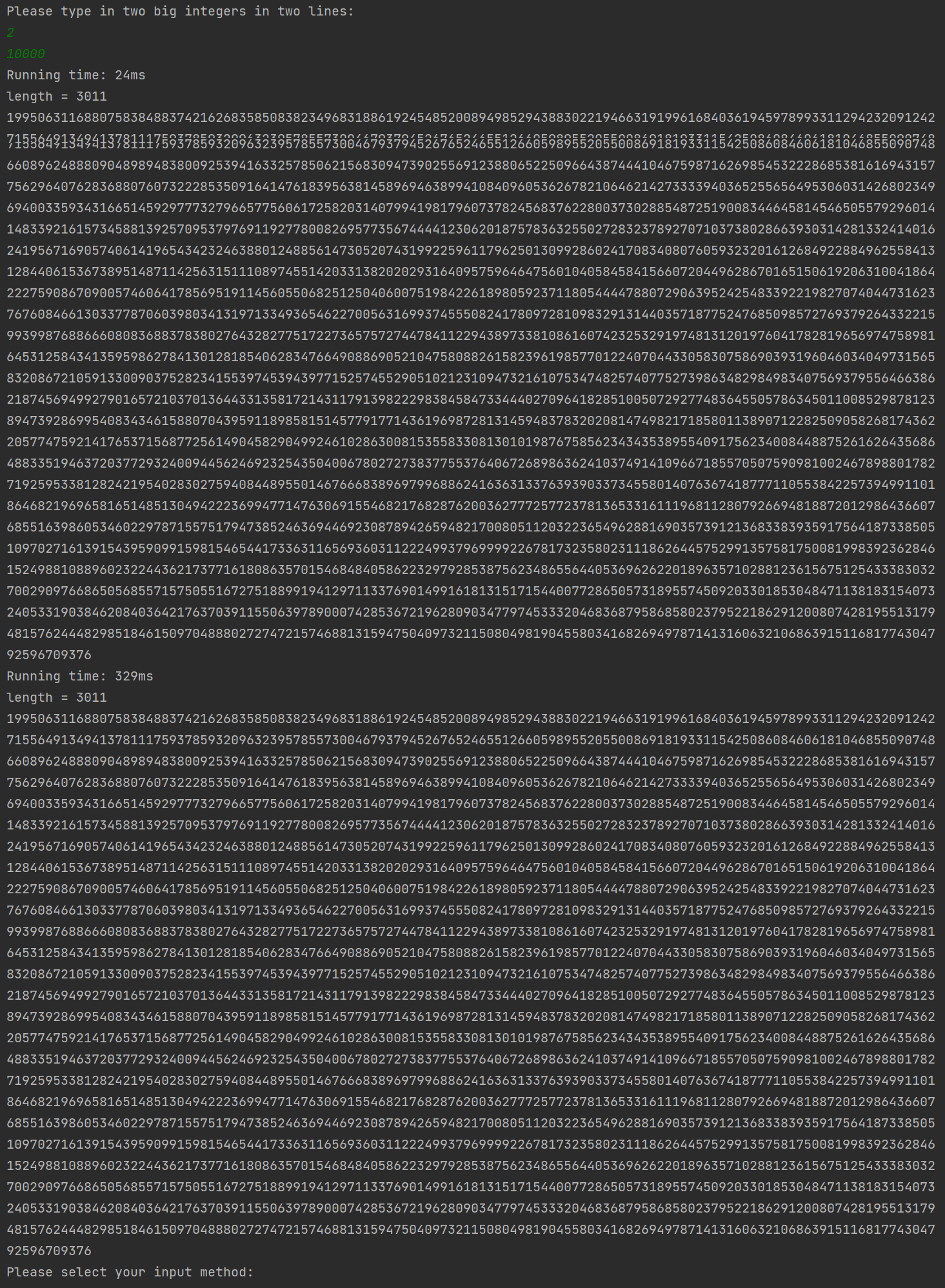
根据python运行出来的结果，答案为：

99250068772098856700831462057469632637295940819886900519816298881382867104749399077921128661426144638055424236936271872492800352741649902118143819672601569998100120790496759517636465445895625741609866209900500198407153244604778968016963028050310261417615914468729918240685487878617645976939063464357986165711730976399478507649228686341466967167910126653342134942744851463899927487092486610977146112763567101672645953132196481439339873017088140414661271198500333255713096142335151414630651683065518784081203678487703002802082091236603519026256880624499681781387227574035484831271515683123742149095569260463609655977700938844580611931246495166208695540313698140011638027322566252689780838136351828795314272162111222231170901715612355701347552371530013693855379834865667060014643302459100429783653966913783002290784283455628283355470529932956051484477129333881159930212758687602795088579230431661696010232187390436601614145603241902386663442520160735566561

根据比对，答案是正确的。但是可以观察到在这个数量级下，快速傅里叶变换的算法仍然不占优。

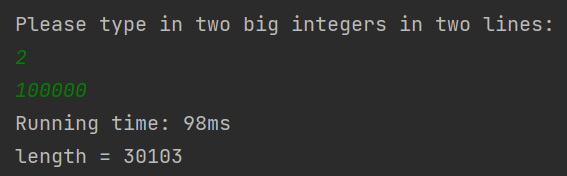
## 4.3 测试性能

根据前面的一些测试结果，发现在不高的数量级下，快速傅里叶变换并没有想象中好的效果。而指数运算的数量级比较大，因此针对指数算法对两种算法的性能进行测试。根据前面的测试，我们认为算法是正确的可靠的，因此我们注重测试性能本身。

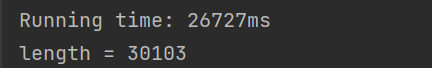


图表 31：测试性能1

我们得到运行2^10000时，快速傅里叶变换算法可以在24ms给出答案，而朴素算法需要329ms，差了十多倍。



图表 32：测试性能2

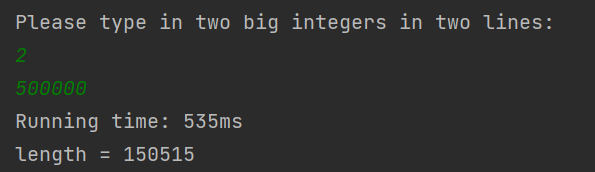


图表 33：测试性能3

然后接着测试了2^100000时的结果，快速傅里叶变换得出答案的时间是98ms，朴素算法用了26727ms，差了大约270倍。根据输出的大整数位数结果，得到2^100000的位数是30103位，可见朴素算法在这种数量级的运算已经是相当吃力了。

接着我们只测试快速傅里叶变换的性能。

测试2^500000的结果。

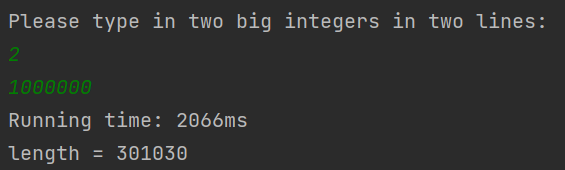


图表 34：测试性能4

得到长度为150515，运行时长为535ms。

继续测试时我们发现事先定义的数组不够大，因此我们临时将数组开到10000000的数量级。

测试2^1000000的结果。



图表 35：测试性能

可以看到在接近两秒的时间里，可以完成2^100000的运算。当然在这个过程中，由于结果的位数达到了十万的数量级，因此数组拷贝的过程会消耗不少时间。总体来讲，采用快速傅里叶变换的算法达到了相当高的效率。

1. **附录**

|  |  |
| --- | --- |
| List.h | #ifndef BIGINTEGER\_FOR\_DATA\_STRUCTURE\_LIST\_H #define BIGINTEGER\_FOR\_DATA\_STRUCTURE\_LIST\_H  #include<iostream> #include<algorithm>  class List { public:  List();   ~List();   List(const List &);   List(const std::string &str);   const int &front();   const int &back();   int size() const;   bool empty() const;   void push\_front(const int &);   void push\_back(const int &);   void pop\_front();   void pop\_back();   void clear();   void remove(const int &);   List &operator=(const List &);   void displayLen(std::string);   void display();   void displayAll();   void fileDisplay();   friend class BigInteger;  private:  class ListNode {  public:  int val;  ListNode \*prev, \*next;   ListNode(const int &v) : val(v), next(nullptr), prev(nullptr) {}   ListNode() : next(nullptr), prev(nullptr) {}  };   int len;  ListNode \*head;  ListNode \*tail; };  #endif //BIGINTEGER\_FOR\_DATA\_STRUCTURE\_LIST\_H |
| List.cpp | #ifndef BIGINTEGER\_FOR\_DATA\_STRUCTURE\_LIST\_CPP #define BIGINTEGER\_FOR\_DATA\_STRUCTURE\_LIST\_CPP  #include "List.h" #include <fstream>  #define MAX\_LEN 0  List::List() : head(new ListNode()), tail(new ListNode()), len(0) {  tail->prev = head;  head->next = tail; }  List::List(const List &rhs) {  head = new ListNode();  tail = new ListNode();  tail->prev = head;  head->next = tail;  len = 0;  ListNode \*tmp = rhs.head->next;  while (tmp != rhs.tail) {  push\_back(tmp->val);  tmp = tmp->next;  } }  List::List(const std::string &str) {  head = new ListNode();  tail = new ListNode();  tail->prev = head;  head->next = tail;  len = 0;  for (int i = 0; i < std::max(MAX\_LEN, (int) str.length()); i++)  if (i >= str.length()) push\_front(0);  else push\_back(int(str[i]) - 48); }  List::~List() {  clear();  delete head;  delete tail; }  const int &List::front() {  if (!empty())  return head->next->val; }  const int &List::back() {  if (!empty())  return tail->prev->val; }  int List::size() const {  return len; }  bool List::empty() const {  if (len == 0)  return true;  else  return false; }  void List::push\_front(const int &elem) {  auto \*tmp = new ListNode(elem);  tmp->prev = head;  head->next->prev = tmp;  tmp->next = head->next;  head->next = tmp;  len++; }  void List::push\_back(const int &elem) {  auto \*tmp = new ListNode(elem);  tmp->next = tail;  tail->prev->next = tmp;  tmp->prev = tail->prev;  tail->prev = tmp;  len++; }  void List::pop\_front() {  if (empty())  return;  ListNode \*tmp = head->next;  tmp->next->prev = head;  head->next = tmp->next;  delete tmp;  len--; }  void List::pop\_back() {  if (empty())  return;  ListNode \*tmp = tail->prev;  tmp->prev->next = tail;  tail->prev = tmp->prev;  delete tmp;  len--; }  void List::clear() {  while (!empty())  pop\_back(); }  void List::remove(const int &elem) {  if (empty())  return;  ListNode \*tmp = head->next;  while (tmp != tail) {  if (tmp->val == elem) {  ListNode \*p = tmp;  tmp->prev->next = tmp->next;  tmp->next->prev = tmp->prev;  tmp = tmp->next;  delete p;  len--;  } else  tmp = tmp->next;  } }  List &List::operator=(const List &rhs) {  if (this == &rhs) return \*this;  clear();  ListNode \*tmp = rhs.head->next;  while (tmp != rhs.tail) {  push\_back(tmp->val);  tmp = tmp->next;  }  return \*this; }  void List::displayLen(std::string io) {  bool flag = false;  int length = 0;  ListNode \*tmp = head->next;  while (tmp != tail) {  if (tmp->val > 0)  flag = true;  if (flag || tmp->next == tail)  length++;  tmp = tmp->next;  }  if (io == "1")  std::cout << "length = " << length << std::endl;  else {  std::ofstream out("../out.txt", std::ios::app);  out << "length = " << length << '\n';  out.close();  } }  void List::display() {  bool flag = false;  ListNode \*tmp = head->next;  while (tmp != tail) {  if (tmp->val > 0)  flag = true;  if (flag || tmp->next == tail)  std::cout << tmp->val;  tmp = tmp->next;  }  std::cout << std::endl; }  void List::displayAll() {  ListNode \*tmp = head->next;  while (tmp != tail) {  std::cout << tmp->val;  tmp = tmp->next;  }  std::cout << std::endl; }  void List::fileDisplay() {  std::ofstream out("../out.txt", std::ios::app);  bool flag = false;  ListNode \*tmp = head->next;  while (tmp != tail) {  if (tmp->val > 0)  flag = true;  if (flag || tmp->next == tail)  out << tmp->val;  tmp = tmp->next;  }  out << '\n';  out.close(); }  #endif //BIGINTEGER\_FOR\_DATA\_STRUCTURE\_LIST\_CPP |
| BigInteger.h | #ifndef BIGINTEGER\_FOR\_DATA\_STRUCTURE\_BIGINTEGER\_H #define BIGINTEGER\_FOR\_DATA\_STRUCTURE\_BIGINTEGER\_H  #include "List.h" #include <cmath> #include <cstring>  struct Complex {  double x, y;   Complex operator+(const Complex &t) const {  return {x + t.x, y + t.y};  }   Complex operator-(const Complex &t) const {  return {x - t.x, y - t.y};  }   Complex operator\*(const Complex &t) const {  return {x \* t.x - y \* t.y, x \* t.y + y \* t.x};  } };  class BigInteger { public:  BigInteger();   BigInteger(const std::string &str);   static int cmp(const BigInteger &a, const BigInteger &b);   static void swap(BigInteger &a, BigInteger &b);   static BigInteger add(BigInteger &a, BigInteger &b);   static std::pair<char, BigInteger> sub(BigInteger &a, BigInteger &b);   static BigInteger mul(const BigInteger &a, const BigInteger &b);   static BigInteger exmul(const BigInteger &a, const BigInteger &b);   static void FFT(Complex para[], int);   static std::pair<BigInteger, BigInteger> div(const BigInteger &a, const BigInteger &b);   static BigInteger div2(BigInteger &a);   static BigInteger exp(BigInteger &a, BigInteger &b);   static BigInteger exexp(BigInteger &a, BigInteger &b);   static bool isZero(BigInteger &a);   void display();   void displayAll();   void fileDisplay();   void displayLen(std::string);  private:  List data; };  #endif //BIGINTEGER\_FOR\_DATA\_STRUCTURE\_BIGINTEGER\_H |
| BigInteger.cpp | #ifndef BIGINTEGER\_FOR\_DATA\_STRUCTURE\_BIGINTEGER\_CPP #define BIGINTEGER\_FOR\_DATA\_STRUCTURE\_BIGINTEGER\_CPP  #include "BigInteger.h"  BigInteger::BigInteger() : data() { }  BigInteger::BigInteger(const std::string &str) {  data = \*new List(str); }  int BigInteger::cmp(const BigInteger &a, const BigInteger &b) {  auto p1 = a.data.head->next, p2 = b.data.head->next;  for (int i = 0; i < a.data.size(); i++) {  if (p1->val > p2->val) return 1;  if (p1->val < p2->val) return -1;  p1 = p1->next, p2 = p2->next;  }  return 0; }  void BigInteger::swap(BigInteger &a, BigInteger &b) {  auto p1 = a.data.head->next, p2 = b.data.head->next;  for (int i = 0; i < a.data.size(); i++) {  int tmp = p1->val;  p1->val = p2->val;  p2->val = tmp;  p1 = p1->next, p2 = p2->next;  } }  BigInteger BigInteger::add(BigInteger &a, BigInteger &b) {  BigInteger res = \*new BigInteger();  int lenA = a.data.size(), lenB = b.data.size();  for (int i = 0; i < lenA - lenB; i++)  b.data.push\_front(0);  for (int i = 0; i < lenB - lenA; i++)  a.data.push\_front(0);  int carry = 0;  auto p1 = a.data.tail->prev, p2 = b.data.tail->prev;  for (int i = 0; i < a.data.size(); i++) {  res.data.push\_front((p1->val + p2->val + carry) % 10);  carry = (p1->val + p2->val + carry) / 10;  p1 = p1->prev, p2 = p2->prev;  }  if (carry > 0) res.data.push\_front(carry);  return res; }  std::pair<char, BigInteger> BigInteger::sub(BigInteger &a, BigInteger &b) {  char sgn;  int lenA = a.data.size(), lenB = b.data.size();  for (int i = 0; i < lenA - lenB; i++)  b.data.push\_front(0);  for (int i = 0; i < lenB - lenA; i++)  a.data.push\_front(0);  int comp = cmp(a, b);  if (comp == -1) {  swap(a, b);  sgn = '-';  } else sgn = '+';  BigInteger res = \*new BigInteger();  int borrow = 0;  auto p1 = a.data.tail->prev, p2 = b.data.tail->prev;  for (int i = 0; i < a.data.size(); i++) {  res.data.push\_front((10 - borrow + p1->val - p2->val) % 10);  borrow = (p1->val - borrow >= p2->val ? 0 : 1);  p1 = p1->prev, p2 = p2->prev;  }  if (sgn == '-') swap(a, b);  return {sgn, res}; }  BigInteger BigInteger::mul(const BigInteger &a, const BigInteger &b) {  BigInteger res = \*new BigInteger();  for (int i = 0; i < a.data.size() + b.data.size(); i++)  res.data.push\_back(0);  auto base = res.data.tail->prev;  auto p1 = a.data.tail->prev;  for (int i = 0; i < a.data.size(); i++) {  auto p = base;  auto p2 = b.data.tail->prev;  for (int j = 0; j < b.data.size(); j++) {  p->val += p1->val \* p2->val;  p = p->prev;  p2 = p2->prev;  }  p1 = p1->prev;  base = base->prev;  }  int carry = 0;  auto p = res.data.tail->prev;  for (int i = 0; i < res.data.size(); i++) {  carry += p->val;  p->val = carry % 10;  carry /= 10;  p = p->prev;  }  return res; }  int arr[300010]; int rev[300010], bit, tot; Complex Ca[300010], Cb[300010];  BigInteger BigInteger::exmul(const BigInteger &a, const BigInteger &b) {  BigInteger res = \*new BigInteger();  memset(arr, 0, sizeof arr);  memset(rev, 0, sizeof rev);  memset(Ca, 0, sizeof Ca);  memset(Cb, 0, sizeof Cb);  auto p1 = a.data.tail->prev;  auto p2 = b.data.tail->prev;  int n = 0, m = 0;  while (p1 != a.data.head) {  Ca[n++].x = p1->val;  p1 = p1->prev;  }  while (p2 != b.data.head) {  Cb[m++].x = p2->val;  p2 = p2->prev;  }  n--, m--;  bit = 0;  while ((1 << bit) < n + m + 1) bit++;  tot = 1 << bit;  for (int i = 0; i < tot; i++)  rev[i] = ((rev[i >> 1] >> 1)) | ((i & 1) << (bit - 1)); //比特逆序  FFT(Ca, 1), FFT(Cb, 1);  for (int i = 0; i < tot; i++) Ca[i] = Ca[i] \* Cb[i];  FFT(Ca, -1);  int k = 0;  for (int i = 0, t = 0; i < tot || t; i++) {  t = t + (int) (Ca[i].x / tot + 0.5);  arr[k++] = t % 10;  t /= 10;  }  while (k > 1 && !arr[k - 1]) k--;  for (int i = k - 1; i >= 0; i--)  res.data.push\_back(arr[i]);  return res; }  const double PI = acos(-1);  void BigInteger::FFT(Complex para[], int inv) {  for (int i = 0; i < tot; i++)  if (i < rev[i])  std::swap(para[i], para[rev[i]]);  for (int mid = 1; mid < tot; mid \*= 2) {  auto w1 = Complex({cos(PI / mid), inv \* sin(PI / mid)});  for (int i = 0; i < tot; i += mid \* 2) {  auto wk = Complex({1, 0});  for (int j = 0; j < mid; j++, wk = wk \* w1) {  auto x = para[i + j], y = wk \* para[i + j + mid];  para[i + j] = x + y, para[i + j + mid] = x - y;  }  }  } }  std::pair<BigInteger, BigInteger> BigInteger::div(const BigInteger &a, const BigInteger &b) {  BigInteger rem = \*new BigInteger(a); //余数  BigInteger tmp = \*new BigInteger(b); //除数辅助变量  BigInteger quo = \*new BigInteger(); //商  auto p = rem.data.head->next;  auto q = tmp.data.head->next;  for (int i = 0; i < a.data.size(); i++) {  if (p->val > 0 || rem.data.size() == 1) break;  p = p->next;  rem.data.pop\_front();  } //被除数（余数）去除前面的0  for (int i = 0; i < b.data.size(); i++) {  if (q->val > 0 || tmp.data.size() == 1) break;  q = q->next;  tmp.data.pop\_front();  } //除数去除前面的0  int d = rem.data.size() - tmp.data.size();  for (int i = 0; i < d; i++)  tmp.data.push\_back(0);  //填充除数后面的0，使之跟被除数相同  for (int i = 0; i < d + 1; i++) {  int val = 0;  while (true) {  auto res = sub(rem, tmp);  if (res.first == '+') {  rem = \*new BigInteger(res.second);  val++;  } else break;  }  tmp.data.pop\_back();  tmp.data.push\_front(0);  quo.data.push\_back(val);  }  if (quo.data.empty()) quo.data.push\_back(0);  return {quo, rem}; }  BigInteger BigInteger::div2(BigInteger &a) {  BigInteger res = \*new BigInteger();  auto p = a.data.head->next;  int r = 0;  bool flag = false;  for (int i = 0; i < a.data.size(); i++) {  r = r \* 10 + p->val;  if (r / 2 > 0 || i == a.data.size() - 1) flag = true;  p = p->next;  if (!flag) continue;  res.data.push\_back(r / 2);  r %= 2;  }  return res; }  BigInteger BigInteger::exp(BigInteger &a, BigInteger &b) {  BigInteger res = \*new BigInteger();  BigInteger two = \*new BigInteger();  res.data.push\_back(1);  two.data.push\_back(2);  while (!isZero(b)) {  if (b.data.tail->prev->val & 1) res = mul(res, a);  a = mul(a, a);  b = div(b, two).first;  }  return res; }  BigInteger BigInteger::exexp(BigInteger &a, BigInteger &b) {  BigInteger res = \*new BigInteger();  res.data.push\_back(1);  while (!isZero(b)) {  if (b.data.tail->prev->val & 1) res = exmul(res, a);  a = exmul(a, a);  b = div2(b);  }  return res; }  bool BigInteger::isZero(BigInteger &a) {  auto p1 = a.data.head->next;  while (p1 != a.data.tail) {  if (p1->val != 0) return false;  p1 = p1->next;  }  return true; }  void BigInteger::display() {  data.display(); }  void BigInteger::displayAll() {  data.displayAll(); }  void BigInteger::fileDisplay() {  data.fileDisplay(); }  void BigInteger::displayLen(std::string io) {  data.displayLen(io); }  #endif //BIGINTEGER\_FOR\_DATA\_STRUCTURE\_BIGINTEGER\_CPP |
| Main.cpp | #include "BigInteger.h" #include <ctime> #include <fstream>  using namespace std;  void init() {  ofstream out("../out.txt", ios::ate);  out.close(); }  int main() {  init();  while (true) {  cout << "Please select your input method:\npress 1: keyboard\notherwise: file\n";  string io;  getline(cin, io);  cout << "Please enter the operation method you select:\n1.addition\n2.subtraction\n3.multiplication\n"  "4.division\n5.exponentiation\n0.EXIT\n";  string opt;  getline(cin, opt);  if (opt == "1") {  string s, t;  if (io == "1") {  cout << "Please type in two big integers in two lines:\n";  getline(cin, s);  getline(cin, t);  } else {  ifstream in("../in.txt");  getline(in, s);  getline(in, t);  in.close();  }  BigInteger a(s), b(t);  clock\_t startTime = clock();  BigInteger res = BigInteger::add(a, b);  clock\_t endTime = clock();  res.displayLen(io);  if (io == "1")  res.display();  else res.fileDisplay();  cout << "Running time: " << static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC \* 1000 << "ms\n";  } else if (opt == "2") {  string s, t;  if (io == "1") {  cout << "Please type in two big integers in two lines:\n";  getline(cin, s);  getline(cin, t);  } else {  ifstream in("../in.txt");  getline(in, s);  getline(in, t);  in.close();  }  BigInteger a(s), b(t);  clock\_t startTime = clock();  auto res = BigInteger::sub(a, b);  clock\_t endTime = clock();  res.second.displayLen(io);  if (res.first == '-') {  if (io == "1") cout << '-';  else {  ofstream out("../out.txt", ios::app);  out << '-';  out.close();  }  }  if (io == "1")  res.second.display();  else res.second.fileDisplay();  cout << "Running time: " << static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC \* 1000 << "ms\n";  } else if (opt == "3") {  string s, t;  if (io == "1") {  cout << "Please type in two big integers in two lines:\n";  getline(cin, s);  getline(cin, t);  } else {  ifstream in("../in.txt");  getline(in, s);  getline(in, t);  in.close();  }  BigInteger a(s), b(t);  clock\_t startTime = clock();  BigInteger res = BigInteger::exmul(a, b);  clock\_t endTime = clock();  res.displayLen(io);  if (io == "1")  res.display();  else res.fileDisplay();  cout << "Running time: " << static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC \* 1000 << "ms\n";  startTime = clock();  res = BigInteger::mul(a, b);  endTime = clock();  res.displayLen(io);  if (io == "1")  res.display();  else res.fileDisplay();  cout << "Running time: " << static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC \* 1000 << "ms\n";  } else if (opt == "4") {  string s, t;  if (io == "1") {  cout << "Please type in two big integers in two lines:\n";  getline(cin, s);  getline(cin, t);  } else {  ifstream in("../in.txt");  getline(in, s);  getline(in, t);  in.close();  }  BigInteger a(s), b(t);  if (BigInteger::isZero(b)) {  cerr << "Error: Divide by 0\n";  continue;  } else {  clock\_t startTime = clock();  pair<BigInteger, BigInteger> res = BigInteger::div(a, b);  clock\_t endTime = clock();  res.first.displayLen(io);  if (io == "1")  res.first.display();  else res.first.fileDisplay();  res.second.displayLen(io);  if (io == "1")  res.second.display();  else res.second.fileDisplay();  cout << "Running time: " << static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC \* 1000 << "ms\n";  }  } else if (opt == "5") {  string s, t;  if (io == "1") {  cout << "Please type in two big integers in two lines:\n";  getline(cin, s);  getline(cin, t);  } else {  ifstream in("../in.txt");  getline(in, s);  getline(in, t);  in.close();  }  BigInteger a(s), b(t), c(s), d(t);  if (BigInteger::isZero(a) && BigInteger::isZero(b)) {  cerr << "Error: 0 ^ 0 undefined\n";  continue;  } else {  clock\_t startTime = clock();  BigInteger res = BigInteger::exexp(a, b);  clock\_t endTime = clock();  cout << "Running time: " << static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC \* 1000 << "ms\n";  res.displayLen(io);  if (io == "1")  res.display();  else res.fileDisplay();  startTime = clock();  res = BigInteger::exp(c, d);  endTime = clock();  cout << "Running time: " << static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC \* 1000 << "ms\n";  res.displayLen(io);  if (io == "1")  res.display();  else res.fileDisplay();  }  } else if (opt == "0") {  break;  } else {  cout << "请重新输入!\n";  }  }  return 0; } |