**本次实验我未尝试在Windows平台上编译，不保证能通过。如有需要，请在Linux平台测试代码。**

**实验2 矩阵链乘与FFT算法**

**王嵩超 PB15000102**

1. **实验要求**

**矩阵链乘**

实现求矩阵链乘问题的算法。随机生成n个矩阵的宽和高，（当然上一个矩阵的高要等于下一个矩阵的宽），用动态规划法求出矩阵链乘问题的最优乘法次序，统计算法运行所需时间 ，画出时间曲线。

**FFT算法**

实现FFT算法。程序输入为两个同样规模的多项式系数列表，将系数列表进行FFT变换，将变化后的点值相乘，再进行FFT反变换，得到相乘后多项式的系数。

1. **实验环境**

编译环境：Lubuntu 17.10 GCC 6.3

机器内存：16G

时钟主频：2.60GHz

1. **实验过程**

**实验过程本质上仍为写代码、调bug、编译、调bug的反复。**

**代码的修改、调试过程可在github查看：**

<https://github.com/songchaow/Algorithm-Labs/commits/master>

1. **编译步骤**
2. g++ matrix.cpp -std=c++11 -O3
3. g++ fft.cpp -std=c++11 -O3
4. **实验设计**

**此节简单地把代码行为描述一遍。**

**本次实验分为两个项目，两个项目的源代码均为独立的cpp文件。**

**Matrix.cpp：规划矩阵链乘程序的源文件。**

要运行的多趟n值存储在一个数组中。每次循环使用数组中的一个n值。运行链乘算法前，矩阵的宽高值先被计算好。

int run(int n,std::vector<unsigned int> &random\_list,unsigned long long\*\* &m, int\*\* &s)用来计算矩阵链乘次序。通过三层循环的动态规划方法，找出最小相乘次数和分割位置，存于整型数组m和s中。

调用完run后，将用printMatrix打印矩阵，以及printOptimalSequenceWrapper打印最优链乘次序。printOptimalSequenceWrapper调用printOptimalSequence，而printOptimalSequence使用递归算法输出括号。

**Fft.cpp：多项式相乘程序的源文件**

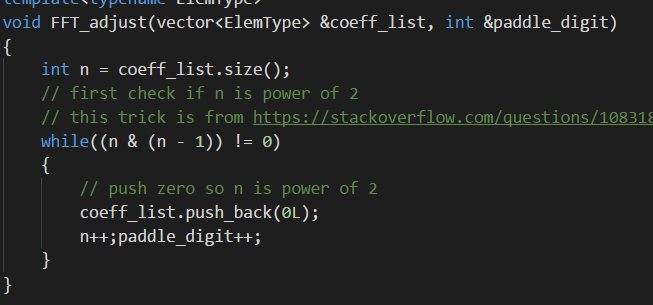
和Matrix.cpp一样，main()函数的每次循环都使用数组中的一个n值。使用RandomGenerator类生成随机数。创建这个类是为了能够每次调用都能生成n个随机数，并保证下次调用时能接着生成，而不是从头生成。

如果n不是2的次方，则在随机数填充零(这相当于更高次项的系数)，直到系数数量为2的n次方。同时在paddle\_digit变量中记录填充零的个数，以便在输出计算结果时将多余的0去掉。

当扩充完系数，使得两个多项式的系数数量均为2的n次方时，再将系数数量加倍(同样是扩充0)。开始FFT变换。结果保存至std::vector<std::complex<long double>> val\_list1, val\_list2。

将val\_list1和val\_list2的元素按位相乘，再进行FFT反变换。根据paddle\_digit去掉多余的0，输出。

1. **实验关键代码截图（结合文字说明）**



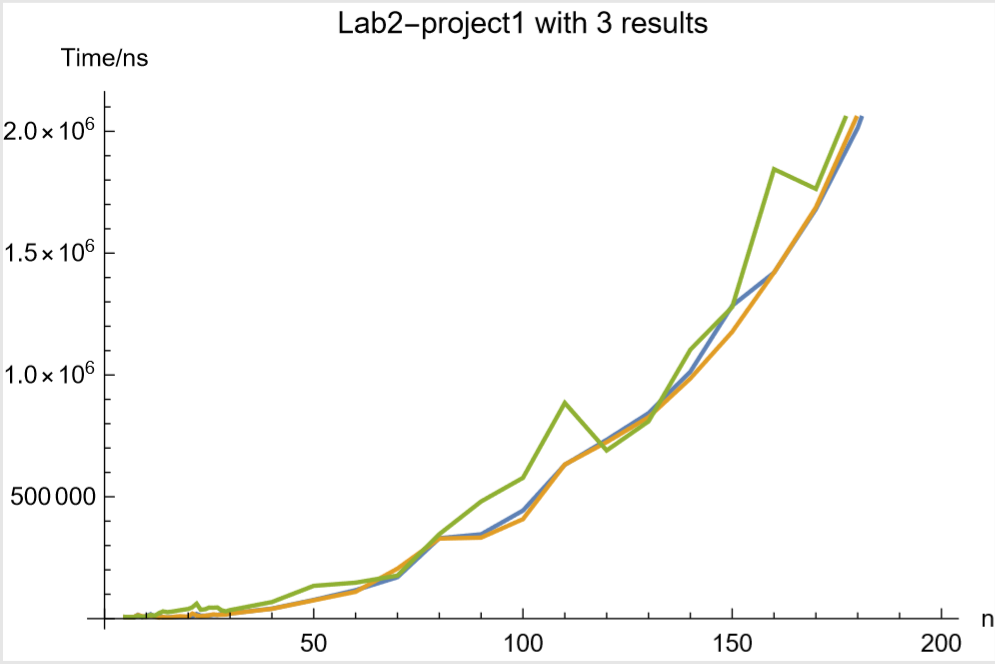
这是用于将系数列表扩增为2的次方数量的函数。

其中n&(n-1)!=0用来判断n是不是2的多次幂。这用到了来自stackoverflow的回答：

<https://stackoverflow.com/questions/108318/whats-the-simplest-way-to-test-whether-a-number-is-a-power-of-2-in-c>

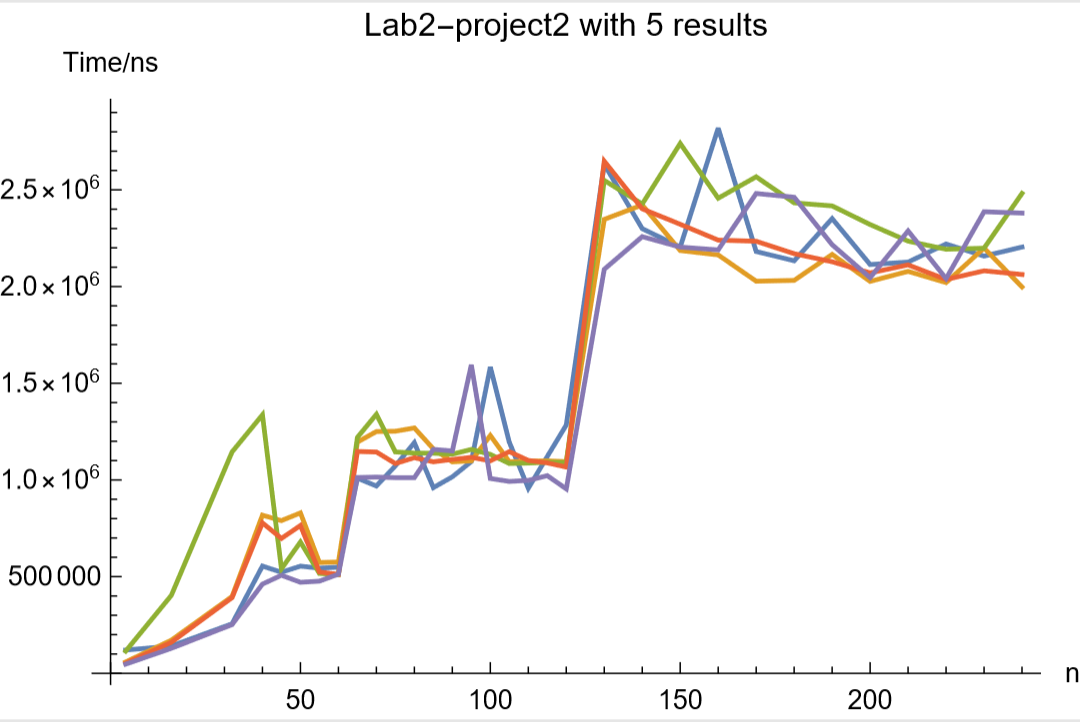
1. **实验结果、分析（结合相关数据图表分析）**

矩阵链乘的耗时与n的关系(运行3次)：

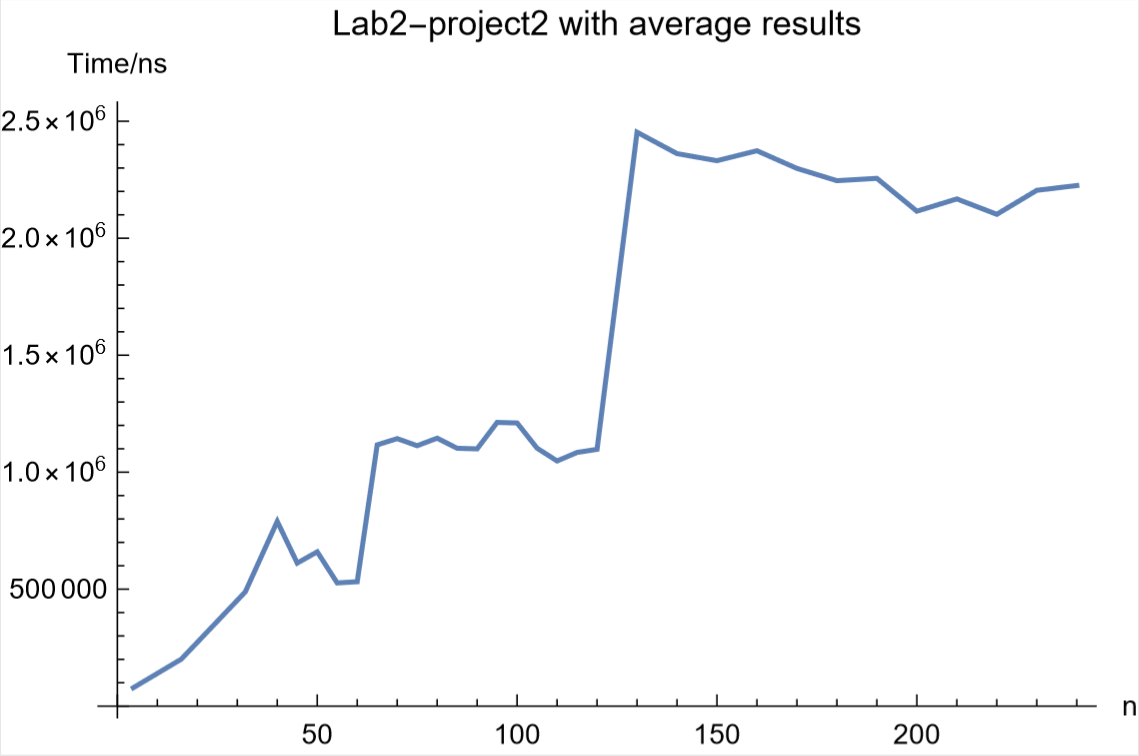


观察曲线的走势，可推出时间复杂度为，符合课本结论。

多项式相乘耗时与n的关系(运行5次)



多项式相乘耗时(平均值)与n的关系(运行5次)



因为真正用于运行FFT算法的vector长度都是2的次方，故在32-64，64-128，128-256这些区间内时间大致为常数。根据理论上的的时间复杂度，n每增加一倍时，会增加略多于一倍的值，与曲线的变化规律相符。

1. **实验心得**

本次实验我再次熟悉了动态规划方法和FFT算法。能更熟练地使用C++模板和类，以及接触了C++ Standard Library中的complex类，以进行复数的运算。