华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析	**年级**：22 级	**上机实践成绩**：
指导教师：金澈清	**姓名**：石季凡	
上机实践名称：Strassen 算法	**学号**：10225501403	**上机实践日期**：2023.3.9
上机实践编号：No.2	**组号**：1-403	

一、目的

1．熟悉算法设计的基本思想

2．掌握 Strassen 算法的基本思想，并且能够分析算法性能

二、内容与设计思想

1. 设计一个随机数矩阵生成器，输入参数包括 N, s, t；可随机生成一个大小为 N*N、数值范围在[s, t]之间的矩阵。
2. 编程实现普通的矩阵乘法；
3. 编程实现 Strassen’s algorithm；
4. 在不同数据规模情况下（数据规模 N=2^4, 2^8, 2^9, 2^10, 2^11）下，两种算法的运行时间各是多少；
5. 思考题：修改 Strassen’s algorithm，使之适应矩阵规模 N 不是 2 的幂的情况；
6. 改进后的算法与 2 中的算法在相同数据规模下进行比较。

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

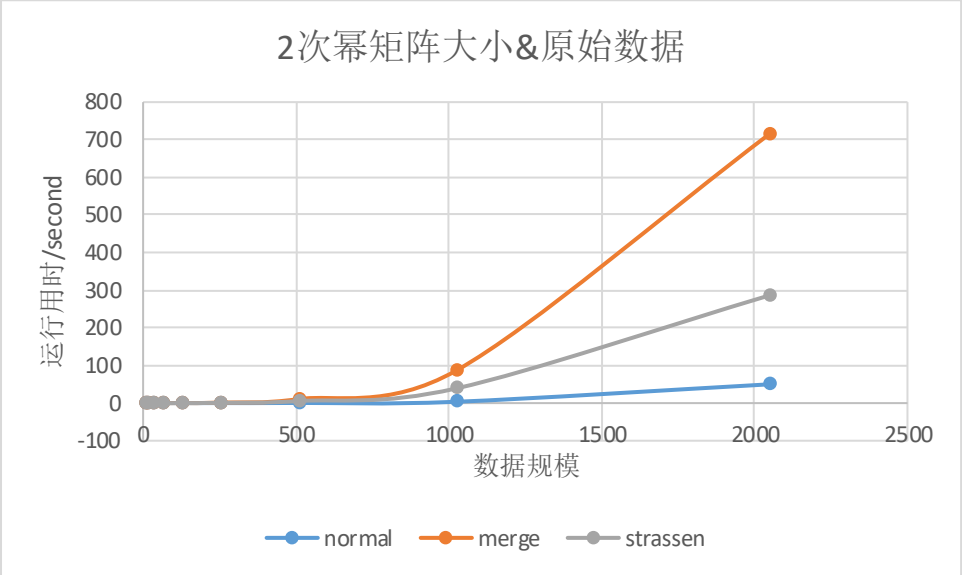
1. 写出矩阵生成器、矩阵乘法、Strassen’s algorithm 的源代码；
2. 分别画出各个实验结果的折线图

五、总结

壹、结果以及分析：

2 的幂的矩阵长宽大小情况下的运算结果

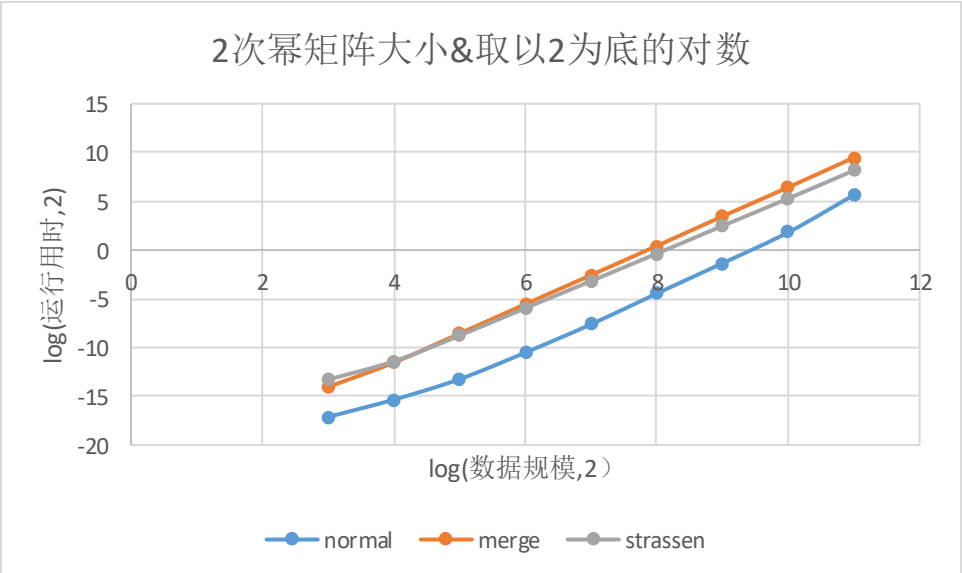
注：两种递归算法均已做修改，均可运算长宽为非 2 次幂的矩阵，但不会影响该情况下的正常结果，详见代码



具体数据如下：

	normal	merge	strassen
8	0.000007	0.000059	0.0001
16	0.000024	0.000333	0.000366
32	0.000103	0.002641	0.002278
64	0.00068	0.021191	0.016017
128	0.00507	0.16832	0.109677
256	0.045396	1.333312	0.773291
512	0.389524	10.9536	5.549977
1024	3.510603	86.38385	38.94071
2048	49.89473	714.8727	286.1481

同时为了更直观的看到数据规模与运算时间的关系，我对输入矩阵的大小与运行时间均取了以 2 为底的对数，得到如下图表：



具体数据如下

	normal	merge	strassen
3	-17.1242	-14.0489	-13.2877
4	-15.3466	-11.5522	-11.4159
5	-13.2451	-8.5647	-8.77802
6	-10.5222	-5.5604	-5.96425
7	-7.6238	-2.57072	-3.18867
8	-4.46129	0.415014	-0.37092
9	-1.36022	3.453333	2.472482
10	1.811719	6.43269	5.283207
11	5.640816	9.481543	8.160618

由运行所得的数据分析，普通的行列相乘算法在运行时间上最为优秀，对另外两种算法为“降维打击”，同时普通算法的运行时间大致符合 O（n^3）。再看到 strassen 算法，其基数较普通的归并拆分算法略大，因此在矩阵大小比较小的情况下，运行时间不如普通的归并拆分算法，但是当 n>=2^5 时，strassen 算法由于其时间复杂度为 O（n^lg7）,较普通的递归拆分算法的时间复杂度 O（n^3）小，因此时间短于普通的归并拆分算法。而且由于普通的行列相乘的时间复杂度也为 O（n^3）,所以可以猜测，当 n 足够大时，strassen 算法所用的时间会短于普通的行乘列算法。（由于运行时间是在太长，没有条件进行验证）

同时，函数图像取对数后，在趋势上更为明显，基本都为线性增长，根据数据可运算得三种算法的线性回归方程：

Normal： $\hat{y}$=-27.0239 + 2.8728 x

Strassen： $\hat{y}$=-22.1415 + 2.7331 x

Merge： $\hat{y}$ =-23.2785 + 2.9681 x

将普通算法的线性回归方程与 strassen 算法的方程联立解得

x=34.94917680744452

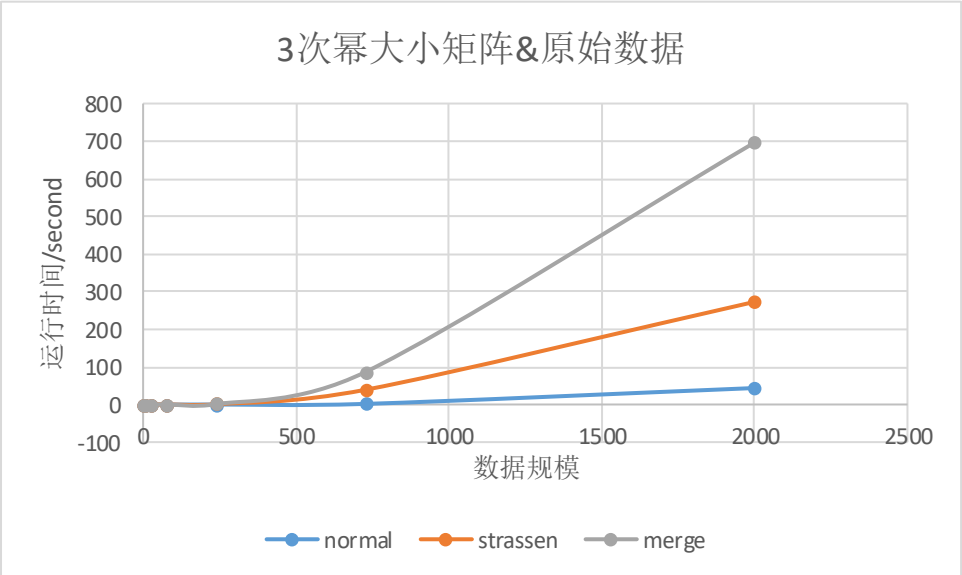
y=73.37809513242662

由此可见，这三种算法的时间复杂度基本符合理论。

3 的幂的矩阵长宽大小情况下的运算结果

为了对这三种算法对于非 2 次幂的大小矩阵乘法的运算进行分析，同时需要作出取对数图像便于分析时间复杂度，我选择了 3 的幂作为测试矩阵的大小，原因为其是除 1 以外，距离 2 最近的和 2 互质的数，但是由于 3^7 刚好略大于 2048，由于我的改进后的算法是将矩阵用 0 补为 2 次幂矩阵，因此为了最后的数据不会飙升得过于离谱，我最后一组矩阵的大小取的是 2000*2000。最终我得到了如下的数据与图表。

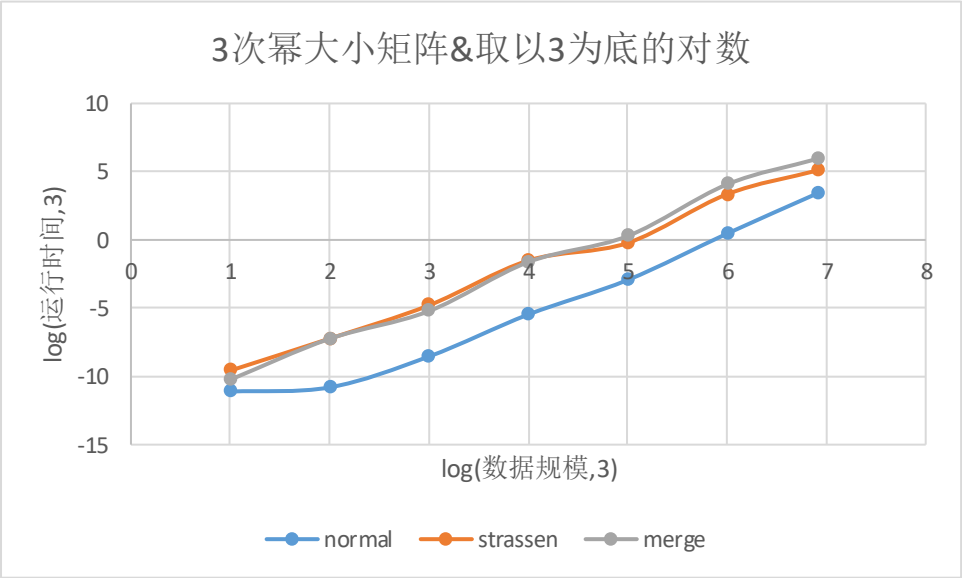
注：普通行列算法并不需要补 0，而是直接可以运算。



具体数据如下

	normal	strassen	merge
3	0.000005	0.000027	0.000013
9	0.000007	0.000355	0.000351
27	0.000084	0.005166	0.00325
81	0.002472	0.190198	0.168691
243	0.039734	0.783333	1.365601
729	1.654043	39.40944	87.075047
2000	43.77312	272.87528	696.34825

同样为了方便观察，我将运行时间与数据规模都以 3 为底取了对数，得到图表如下



具体数据如下

	normal	strassen	merge
1	-11.1104	-9.57542	-10.2407
2	-10.8042	-7.23039	-7.2407
3	-8.54232	-4.79301	-5.21485
4	-5.46392	-1.51071	-1.61994
5	-2.93602	-0.22228	0.283626
6	0.458053	3.344224	4.06583
6.91864	3.439812	5.105545	5.958289

对于三次幂大小的矩阵运算，普通的行列算法依然表现出了碾压级的优势，而与 2 次幂情况类似，strassen 算法在 n=5 时超过了普通的分块算法，并在后续结果中保持。

同样我们也可以计算一下三者的线性回归方程

Normal： $\hat{y}$ =-15.2987 + 2.5836 x

Strassen：$\hat{y}$ =-12.1521 + 2.5138 x

Merge： $\hat{y}$ =-13.0254 + 2.7641 x

由此可见，结合图像，我们可以知道以补 0 法完善的算法不适合取对数后建立线性回归模型，方程难以分析出有用的信息。

贰、心得以及改进：

这次用 c 语言实现 strassen 算法时，最开始我想二级指针作为数组容器，来作为 strassen 函数的返回类型，但是在接下来的实现过程中内存不断出现泄漏，经过了助教学长的指导修改了一个明显的错误类型，即直接在调用乘法函数时，以加减法函数作为其参数，这会导致这部分空间被 malloc 以后无法被释放，修改后经检验还是有内存泄漏，于是换成了 void 类型的函数，提前申请好结果存放的空间，利于对于内存的管理，最终完成了 c 语言对于 strassen 算法的实现。

同时，由于写代码时复制粘贴类似内容过于多，导致最开始没有实现，因此以后对于复制的内容一定要修改好变量。

这次实验的进行锻炼了我的细心程度，加强了我对于递归的理解。