排序实验

1. 实验要求
   - 实验 1：排序 n 个元素，元素为随机生成的长为 1..16 的字符串（字符串均为英文小写字母），n 的取值为：3^2，3^4，3^6，3^8，3^{10}，3^{12}；

     算法：直接插入排序，堆排序，归并排序，快速排序，SHELL 排序（5 个增量，分别是：1，3，7，15，21）。
   - 实验 2：排序 n 个元素，元素为随机生成的 1 到 65535 之间的整数，n 的取值为：3^3，3^5，3^7，3^9，3^{11}，3^{13}；

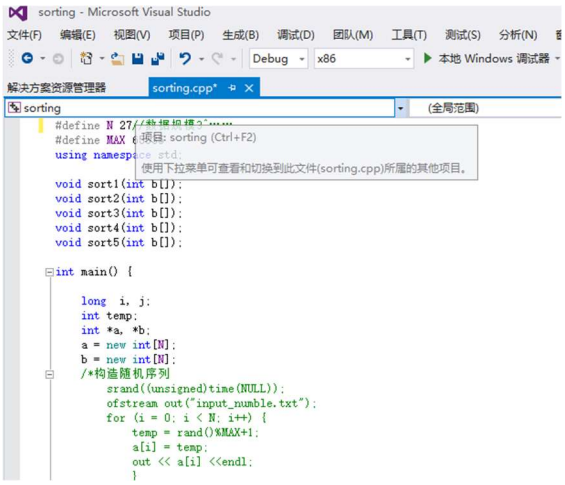
     算法：冒泡排序，快速排序，归并排序，　基数排序，计数排序。
   - 字符串大小判断标准：
   - 1.首先按字符串长度进行排序（短字符串在前，长字符串在后）。
   - 2.对长度相同的字符串，按字母顺序进行排序。

2. 实验环境

编译环境：Microsoft Visual Studio 2015
机器内存：8G
时钟主频：2.5GHz

3. 实验过程
   - a)创建随机数据文件 input_numble.txt 和 input_sting.txt
   - b)input：
     - 输入文件中每行一个随机数据，总行数大于等于 3^{13}
     - 顺序读取 N 个数据，进行排序
   - C)output：
     - 每种算法建立一个子文件夹，其输出结果数据导出到其对应子文件下面
       - result_n.txt：排序结果的数据(N 为数据规模的指数)，每个数据规模一个输出文件
       - time.txt：运行时间效率的数据，六个规模的时间结果都写到同个文件

   实验截图：



4. 实验关键代码

以 ex2 为例

1、使用rand函数构造随机序列

```
srand((unsigned)time(NULL));
ofstream out("input_numble.txt");
for (i = 0; i < N; i++) {
    temp = rand()%MAX+1;
    a[i] = temp;
    out << a[i] <<endl;
    }
```

2、从文件获取数据

```
ifstream in;
in.open("input_numble.txt");
for (i = 0; i < N; i++) {
    in>>a[i];
}
in.close();
```

3、计算排序算法的时间

使用 windows 系统的计时 api.

```
#include<windows.h>
LARGE_INTEGER li;
long long f_time;
ofstream timeout("time.txt");
QueryPerformanceCounter(&li);       //计算算法效率时间
        long long f_start = li.QuadPart;
        sort(b);
        QueryPerformanceCounter(&li);
        long long f_end = li.QuadPart;
        f_time = (f_end - f_start);
        timeout << "冒泡排序 time=" << f_time << endl;
        ofstream out("冒泡排序 result.txt");
```

QueryPerformanceCounter() 这个函数返回高精确度性能计数器的值.但是QueryPerformanceCounter()确切的精确计时的最小单位是与系统有关的,所以,本实验用相对时间长度比较各算法的性能效率。

4、排序算法实现见源代码

5. 实验结果、分析

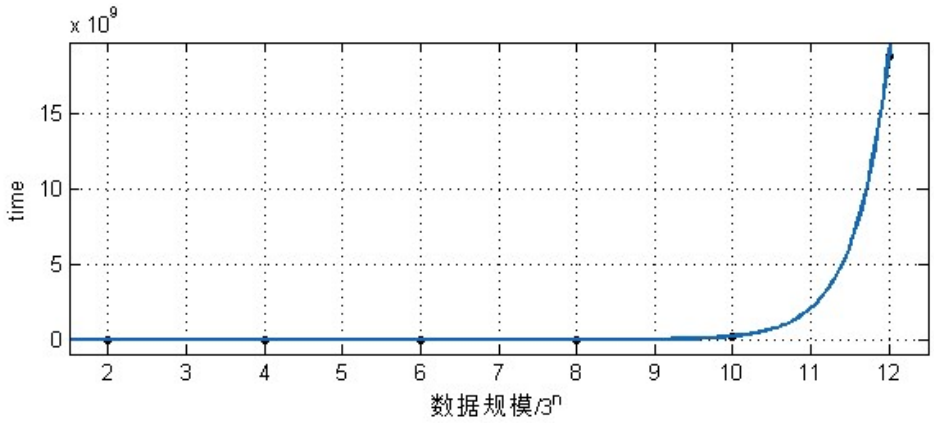
Ex1:各排序算法用时：

数据规模3^n	数据规模实际值	SHELL排序	归并排序	快速排序	直接插入排序	堆排序
2	9	10	45	20	11	27
4	81	646	480	289	585	516
6	729	38500	12166	6870	44690	7375
8	6561	3140608	655240	36160	4712046	144227
10	59049	231587948	267780594	474772	233133558	986233
12	531441	18824415874	28334380614	7372113	19032713726	10733293

各排序算法得到的结果存储在 output 下的文件夹。

1、SHELL 排序

算法渐进性能 T(N)=〇(N*N)=〇(3^2n).



General model:

f(x) = a*3^(2*x)

Coefficients (with 95% confidence bounds):

a = 0.06665 (0.06665, 0.06666)

Goodness of fit:

SSE: 7.334e+11

R-square: 1

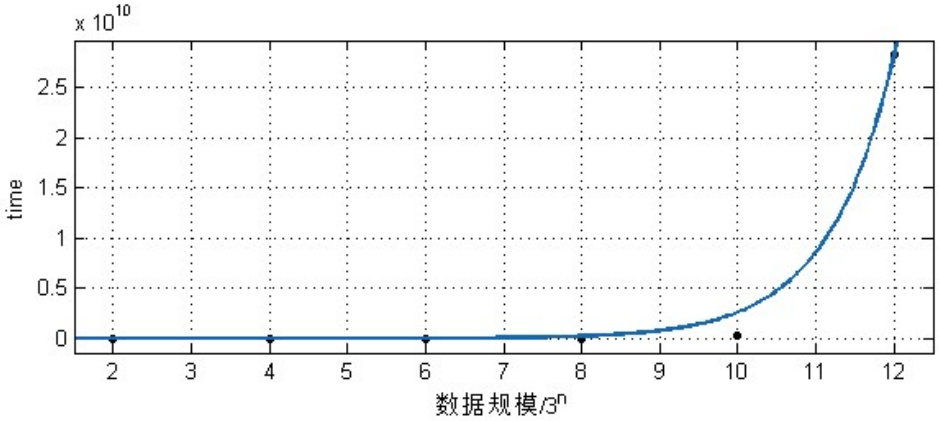
Adjusted R-square: 1

RMSE: 3.83e+05

实验结果的拟合曲线和最坏情况下的算法渐进性能符合的较好。

2、归并排序

算法渐进性能 T(N)= θ (NlgN)= θ (n*3^n)



General model:

f(x) = a*x*3^x

Coefficients (with 95% confidence bounds):

a = 4409 (3986, 4832)

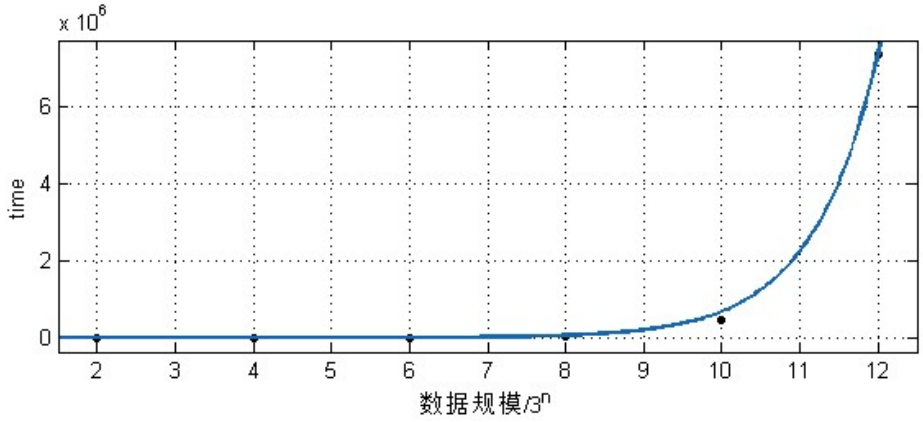
Goodness of fit:

SSE: 5.556e+18
R-square: 0.9917
Adjusted R-square: 0.9917
RMSE: 1.054e+09

实验结果的拟合曲线和算法渐进性能符合的较好。

3、快速排序

算法渐进性能 T(N)=θ(NlgN)=θ(n*3^n)



General model:

f(x) = a*x*3^x

Coefficients (with 95% confidence bounds):

a = 1.153 (1.116, 1.19)

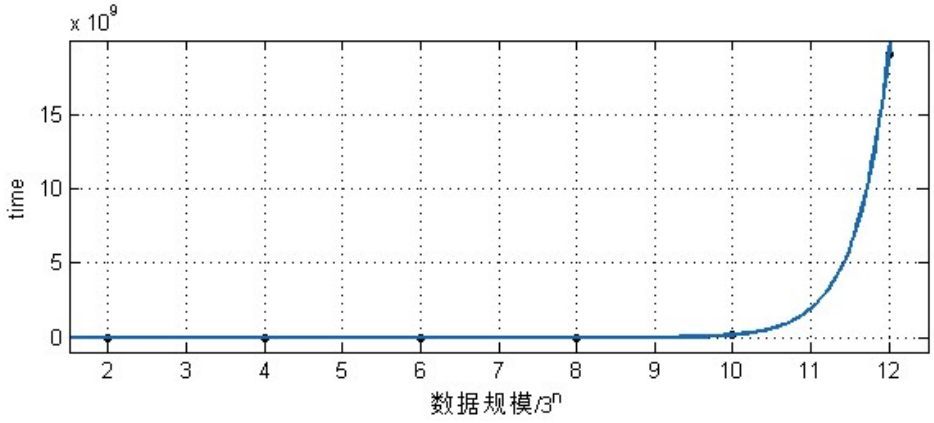
Goodness of fit:

SSE: 4.342e+10
R-square: 0.999
Adjusted R-square: 0.999
RMSE: 9.319e+04

实验结果的拟合曲线和算法渐进性能符合的非常好。

4、插入排序

算法渐进性能 T(N)=θ(N*N)=θ(3^2n)。



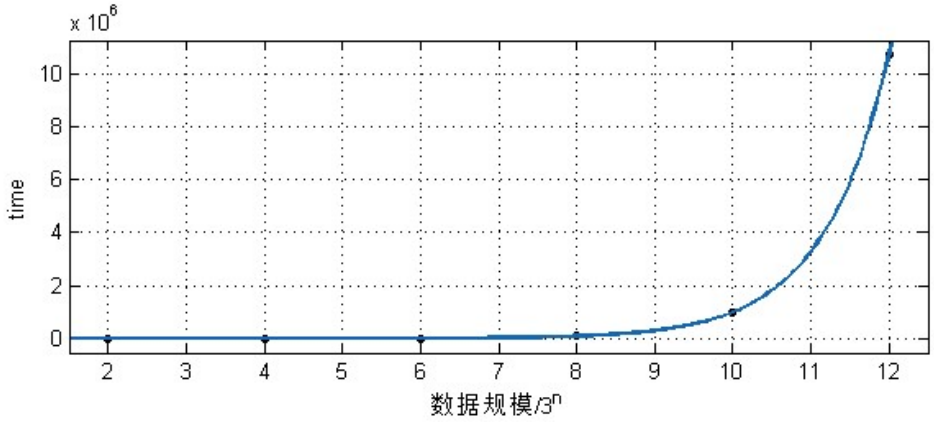
General model:

f(x) = a*3^(2*x)

Coefficients (with 95% confidence bounds):
a = 0.06739 (0.06738, 0.0674)

Goodness of fit:
SSE: 6.659e+12
R-square: 1
Adjusted R-square: 1
RMSE: 1.154e+06Ex2:
实验结果的拟合曲线和算法渐进性能符合的非常好。

5、堆排序
算法渐进性能 T(N)=〇(N+NlgN)=〇(3^n+cn3^n)≈〇(n*3^n)



General model:
f(x) = a*x*3^(x)
Coefficients (with 95% confidence bounds):
a = 1.683 (1.673, 1.693)

Goodness of fit:
SSE: 3.181e+09
R-square: 1
Adjusted R-square: 1
RMSE: 2.522e+04
实验结果的拟合曲线和最坏情况下的算法渐进性能符合的较好。

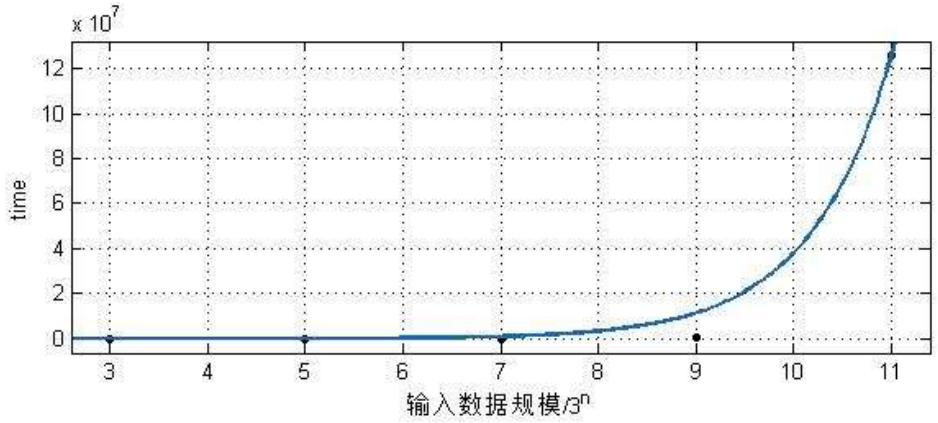
总结：横向比较，在不同的数据规模下，快速排序的性能最好。

Ex2:各排序算法用时：

数据规模 (3^n)	数据规模实际值	归并排序	基数排序	计数排序	快速排序	冒泡排序
3	27	280	77	1179	10	8
5	243	642	469	1222	90	351
7	2187	27274	4029	1328	1075	29308
9	19683	356745	29239	1712	9539	2941081
11	177147	125758386	279402	9969	112357	261151158
13	1594323		2569062	91346	1348595	

各排序算法得到的结果存储在 output 下的文件夹。
由于归并排序和冒泡在数据规模 3^13 时运行时间过长，故没有计算在内。
1、归并排序

算法渐进性能 T(N)=θ(NlgN)=θ(n*3^n)

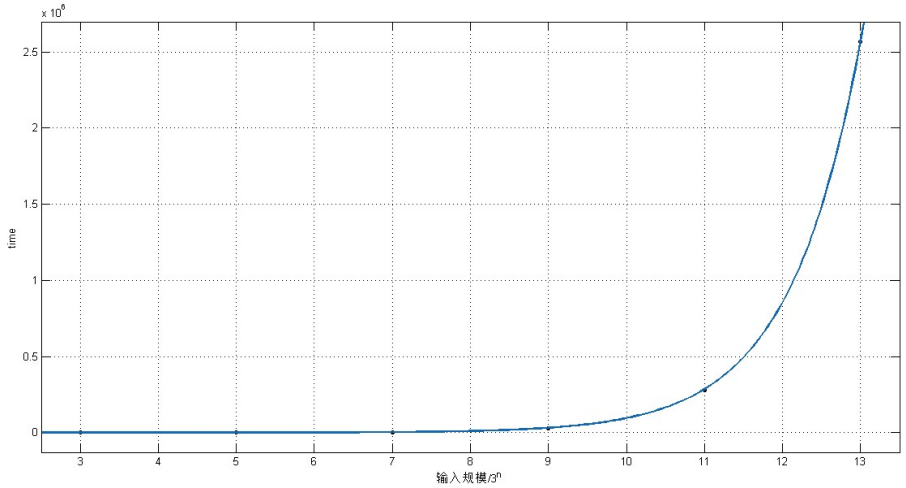


General model:
f(x) = a*x*3^x
Coefficients (with 95% confidence bounds):
a = 64.02 (56.17, 71.88)
Goodness of fit:
SSE: 1.226e+14
R-square: 0.9903
Adjusted R-square: 0.9903
RMSE: 5.536e+06

实验结果的拟合曲线和算法渐进性能符合的较好。
2、基数排序

算法渐进性能 T(N)=θ(d(N+k))=θ(d*(3^n+k))。由于数据取的是 1 到

65535 之间的整数，按高位补 0 作比较，d=5,k=10;所以 T(N)=θ(3^n)。



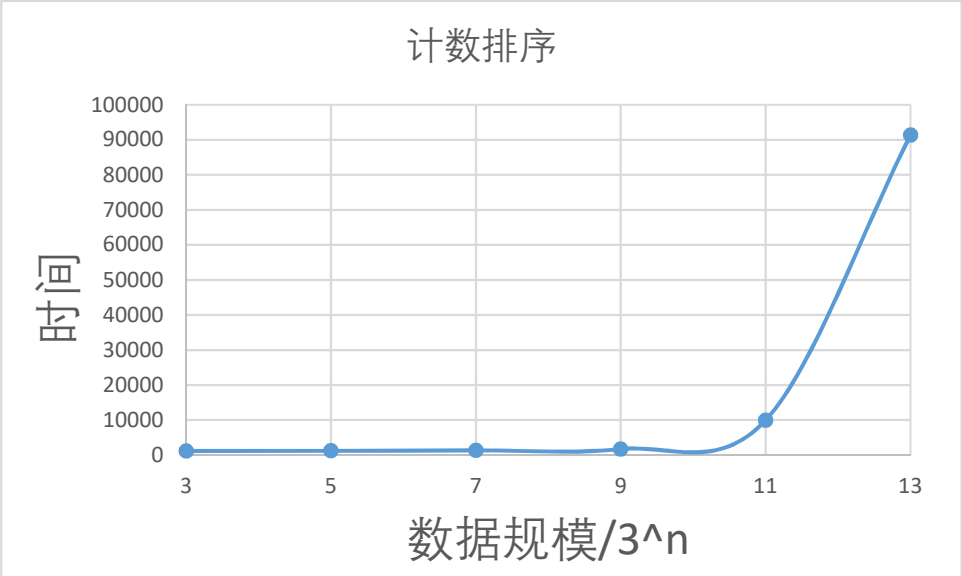
```
General model:
     f(x) = a*3^x
Coefficients (with 95% confidence bounds):
       a =       1.611  (1.606, 1.616)

Goodness of fit:
  SSE: 4.251e+07
  R-square: 1
  Adjusted R-square: 1
RMSE: 2916
```

实验结果的拟合曲线和算法渐进性能符合的非常好。

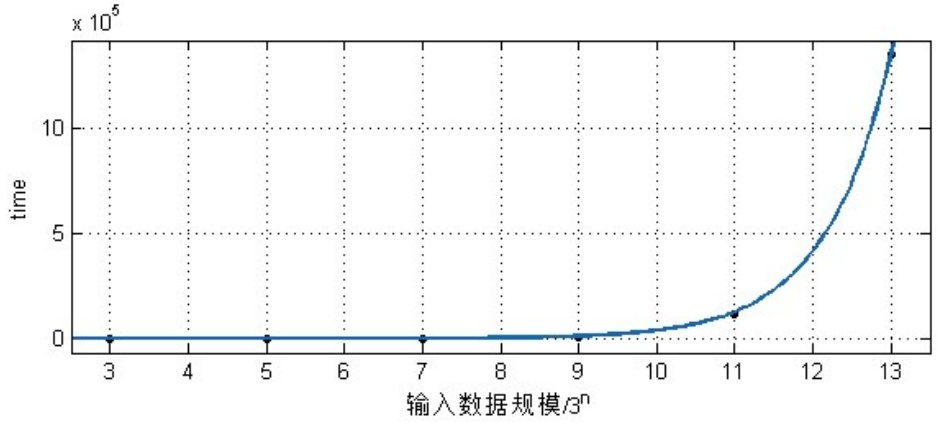
3、计数排序

算法渐进性能 T(N)= θ (3^n+k)，由于数据取的是 1 到 65535 之间的整数，这里 k 应取 65535。在数据规模<=65535 时（n<=7），算法用时趋近于〇(k);当数据规模大于>65535 时(n>=9)，算法用时趋近于 θ (3^n)。表现在图上即是先线性增长，后指数增长（以 n 为横坐标）。



4、快速排序

算法渐进性能 T(N)=θ(NlgN)=θ(n*3^n)



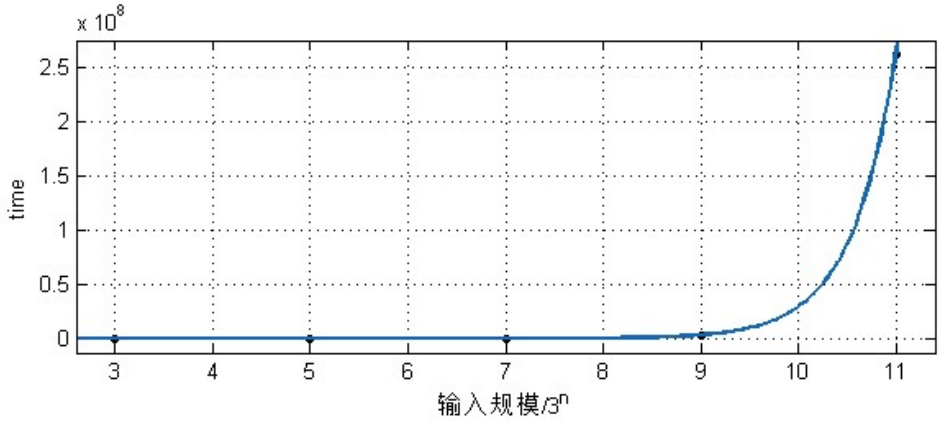
```
General model:
     f(x) = a*x*3^x
Coefficients (with 95% confidence bounds):
       a =       0.065   (0.0642, 0.0658)

Goodness of fit:
  SSE: 2.104e+08
  R-square: 0.9999
  Adjusted R-square: 0.9999
RMSE: 6487
```

实验结果的拟合曲线和算法渐进性能符合的非常好。

5、冒泡排序

算法渐进性能 T(N)= θ (N*N)= θ (3^2n)



General model:

f(x) = a*3^(2*x)

Coefficients (with 95% confidence bounds):

a = 0.008322 (0.008309, 0.008334)

Goodness of fit:

SSE: 8.019e+10

R-square: 1

Adjusted R-square: 1

RMSE: 1.416e+05

实验结果的拟合曲线和算法渐进性能符合的非常好。

总结：就整体来看，当数据规模较小时（n<=7）,快速排序的性能最好；当数据规模较大时（n>=8）,计数排序的性能最好。

6. 实验心得

通过本次实验，我熟悉了不同算法的使用及其时间复杂度，大致感受了各算法在不同数据规模上的优劣。

其中，值得注意的是，归并排序的在大规模数据上的用时比同级别算法复杂度的快速排序和堆排序慢很多。仔细思考，归并排序在运行时需要不断递归调用并为左右子数组分配临时空间。所以，虽然归并排序减少了比较次数，但在调用函数，分配空间，合

并左右子数组话费了较多的时间。另外，归并排序是稳定的排序，而快速排序不稳定，这是归并排序比快速排序优势的地方。

由于计数 k 的需要，计数排序在大规模（远大于 k）的数据性能最好，但不适合小规模（远小于 k）的数据。基数排序在小规模数据时优于计数排序，然而在大规模数据时，由于需要多轮排序，性能又比计数排序差了。

Shell 排序算法复杂度应在 o(N)和〇(N^N)之间，大致上，它的性能优于直接插入排序。但对于大规模数据而言（n>8），27 开始的增量序列并不能显著改善性能。整体渐进性能趋于〇(N^N)。