深 圳 大 学 实 验 报 告

课程名称 算法设计与分析

项目名称 排序算法性能分析

学　　院 计算机与软件学院

专　　业 软件工程

指导教师 卢亚辉

报 告 人 郑彦薇 学号 2020151022

实验时间 2022/3/1

提交时间 2022/3/8

教务处制

一、实验目的与要求

1、掌握选择排序、冒泡排序、合并排序、快速排序、插入排序算法原理
2、掌握不同排序算法时间效率的经验分析方法，验证理论分析与经验分析的一致性。

二、实验内容与方法

实验内容（问题描述）：

该实验进行的是对选择、冒泡、插入、快速、归并排序这 5 种不同的排序算法的原理进行阐述并根据算法实现原理编写出相应的伪代码和源代码；以及对这五种不同的算法在具体实现过程中，不同的规模下完成一次完整的排序所需的平均时间，对其效率进行分析和比较。

实验方法：

编写 5 种排序算法，分别以规模为 10、100、…、1000000，固定规模，随机产生 20 组数据，进行不同的排序过程，记录排序所需的平均运行时间。取规模以及运行时间对应的对数表示，通过 excel 作图观察运行时间随规模增长趋势，与理论运行效率进行对比。

三、实验步骤与过程

（一）5 种排序的算法原理、实现细节及对应伪代码：

1、选择排序：

原理：从待排序列第一个数开始，在其后面的数中选出最小的数，将该数与第一个数交换，从而实现将最小的数放在第一位；然后从第二个数开始，重复上述过程，实现将第二小的数放在第二位…反复进行上述过程至倒数第二个数（即第 n−1 位），即完成整个待排序列的排序。根据算法原理，可以得到选择排序的时间复杂度为 $O(n^2)$，空间复杂度 $O(1)$。

算法实现细节：将当前待“放入数据”的位置下标记为 i，不断进行 i 对应的数之后 **n−i+1 个数据**的两两比较，把每一次比较中的较小数对应的下标记为 k（即 k 对应的数据为当前要找的最小值，实质是在更新 k 所对应的数值），从而实现访问到最后一位数时，**k 所对应位置的数据为当前最小值**，再将 k 位置对应数据与 i 位置对应数据进行交换即可。重复上述过程直到 i=n−1。

过程演示：

算法原理对应伪代码：

```
SelectSort(n:integer)
    var i,j,k:integer;
    for i←1 to n do
        k←i;
        for j←i+1 to n do
            if key[k]>key[j] do
                k←j;
            j++;
        if k!=i do
            swap(key[k], key[i]);
        i++;
```

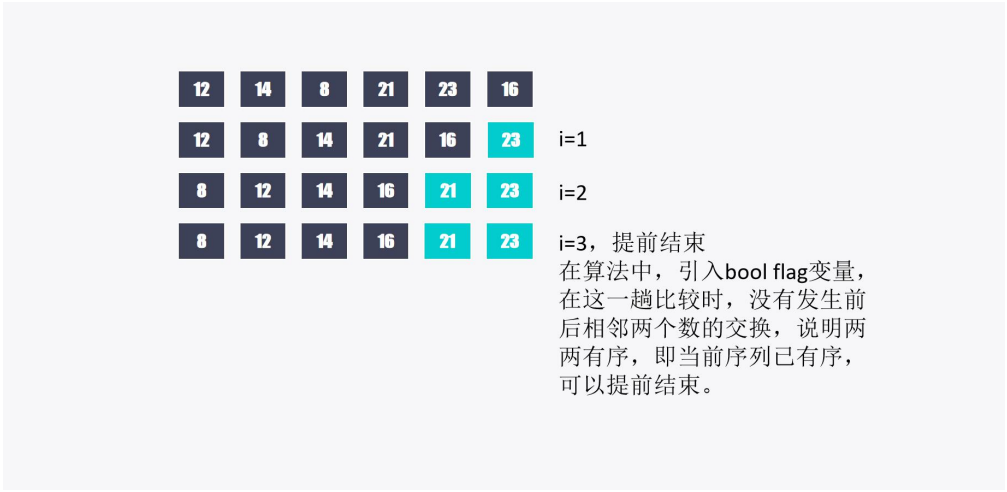
2、冒泡排序：

原理：对 n 个数据进行 n−1 趟排序，从第一个数开始，前后数据两两进行比较，将较大数换在后面，实现当前需要排序的数据中最大的数往下沉。根据算法原理，可以得到冒泡排序的时间复杂度为 $O(n^2)$，空间复杂度 $O(1)$。

算法实现细节：从第一个位置开始（算法中记 i=1，运行过程中 i++），与其后一个数进行比较，若发生逆序，将数据进行交换，从而实现“将较大数往下沉”，直至比较到最后一个数，完成最后一个数与其前一个数的**比较和是否需要交换的判断**后，当前序列中的最后一个数已经为该待排序列的最大值，即实现将最大的数放在最后一个位置。

然后从第一个数到倒数第二个数重复进行上述过程，进行新的一趟冒泡排序，直到最后一趟完成（即最后一趟只有第一个数和第二个数的比较）。

过程演示：



算法原理对应伪代码：

```
BubbleSort(n:integer)
    var i,j:integer;
    flag:boolean;
    flag←true;
    for i←1 to n and flag do
        flag←false;
        for j←1 to n−i+1 do
            if key[j]>key[j+1] do
```

```
                swap(key[j+1], key[j]);
                flag←true;
            j++;
        i++;
```

3、归并排序：

原理：利用分治的思想，合并序列大小从 1 开始，将**前后相邻两个有序序列**合并为一个有序序列。重复进行合并操作，直到只有一个有序序列为止。注意在第一趟时，每一个单独的数据视为一个有序序列。根据算法原理，可以得到归并排序的时间复杂度为 $O(nlogn)$，空间复杂度 $O(n)$。

算法实现细节：在算法中，将每次进行归并的序列**大小记为 step**，用 left 表示前一个序列的第一位，mid 表示前一个序列的最后一位，right 表示后一个序列的最后一位。

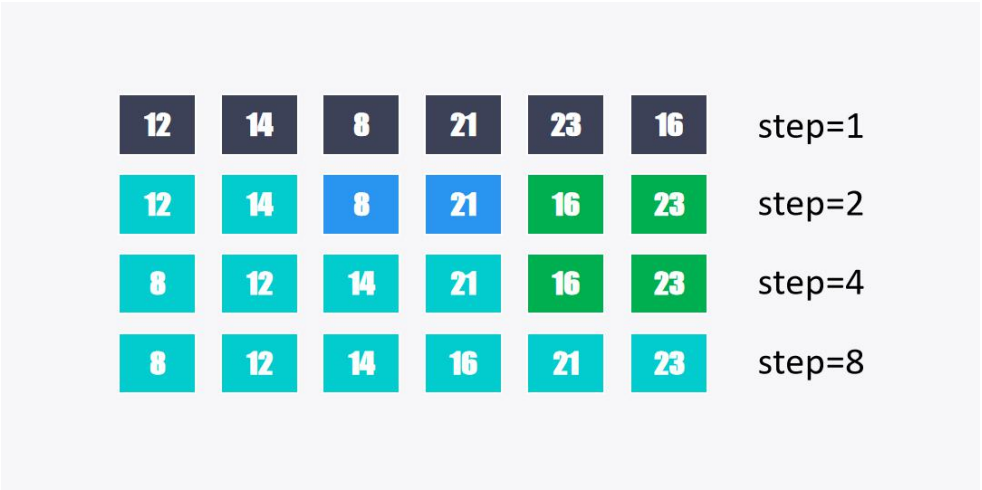
①用 i，j 分别标记当前比较的两个数的下标，其中 **i 的范围从 left~mid，j 的范围从 mid+1~right**，创建临时数组 data

②比较当前 **i，j 所指数，将其中较小的数存入** data，直到所有前一个序列的数或所有后一个序列的数都进行了比较（即 i 或 j 已经达到所属范围的最大值），再将剩余未作比较的数复制到 data 中，即完成一次归并。

③每一次归并完成后，**step 增大为原来的两倍**，这是因为新的一趟排序中，参与合并的有序序列为上一趟合并得到的，有序序列数据的数量会增多。

重复上述过程，直到排列有序。

过程演示：



算法原理对应伪代码：

```
MergeSort(n:integer)
    var left,mid,right,step:integer;
    for step←1 to n do
        for left←1 to n do
            mid←left+step−1;
            if mid>=n do
                break;
            right←left+2*step−1;
            if right>n do
                right←n;
```

```
        var i,j,k,t:integer;
        i←left,j←mid+1,k←left;
        while i<=mid and j<=right do
            if key[i]<=key[j] do
                data[k++]←key[i++];
            else
                data[k++]←key[j++];
        while i<=mid do
            data[k++]←key[i++];
        while j<=right do
            data[k++]←key[j++];
        left←left+2*step;
    step←step*2;
    for t←1 to n do
        key[t] = data[t];
        t++;
```

4、快速排序：

原理：快速排序过程中有一个关键值我们称为为基准记录。快速排序即是按基准记录的数据的大小，遍历整个序列，找到小于基准的数放在序列前部分，大于基准的数放在后部分。基准将整个序列划分为左右子序列，左子序列的所有数据都小于基准值，右子序列的所有数据都大于基准值。再对左右子序列分别进行排序，重复这个过程，直至序列排列有序。根据算法原理，可以得到快速排序的时间复杂度为 $O(nlogn)$，空间复杂度 $O(logn)$。

算法实现细节：对于要进行快速排序的序列中，通常将第一个元素记为基准（在代码中用 p 表示），指针 low 从待排序列第一个数开始，指针 high 从待排序列最后一个数开始，每一趟排序都进行以下过程：

①从 high 指向的数开始，从后往前找第一个比 p（基准）小的数，将其放到 low 指向的位置，low++；

②从 low 指向的数开始，从前往后找第一个比 p（基准）大的数，将其放到 high 指向的位置，high−−；

③重复①②，直到 low=high，将 p 所记录的数据放在 low（即 high）指向的位置，至此，完成一趟快速排序；

对上一趟排序结果以基准划分为前后两个新的子序列进行相同的操作，直到每个子序列只有一个记录为止。

过程演示：

算法原理对应伪代码：

```
QuickSort(low,high:integer)
    var i,j,p:integer;
    i←low;
    j←high;
    p←key[low];
    while low<high do
        while low<high and key[high]>=p do
            high−−;
        if low<high do
            key[low]←key[high];
            low++;
        else
            break;
        while low<high and key[low]<=p do
            low++;
        if low<high
            key[high]←key[low];
            high−−;
    key[low]←p;
    if i<low−1 do
        QuickSort(i,low−1);
    if high+1<j do
        QuickSort(high+1,j);
```

5、插入排序：

原理：每一趟排序将待排序的数据，插入前面已经排好序的序列中，直到所有数据全部插入，即整个序列有序。根据算法原理，可以得到插入排序的时间复杂度为 $O(n^2)$，空间复杂度 $O(1)$。

算法实现细节：①每一趟将**待排序的数据**（在代码中我们用 temp 标记），从**部分已经有序序列的最后一位开始，逐位往前**，分别与 temp 进行比较。

②若部分有序表中当前被比较的数大于 temp，则该数往后挪一位；由于被插入的序列

已经有序，故当出现一个比 temp 小的数时，说明该数前面的数据都小于 temp，此时该数据所在位置的**后一个的位置**即为 temp 要插入的位置。

③将数据插入部分有序表的适当位置上，再对整个序列中未进行插入的数，**重复**这个过程，直到对象全部插入（即整个序列有序）为止。

过程演示：

算法原理对应伪代码：

```
InsertSort(n:integer)
    var i,j,temp:integer
    for i←2 to n do
        temp←key[i];
        for j←i−1 to 1 do
            if temp<key[j] then
                key[j+1] ← key[j];
            else
                break;
            j−−;
        key[j+1]←temp;
        i++;
```

（二）算法测试结果及效率分析：

分别以规模 n=10、100、1000、10000、100000、1000000，随机生成 20 组数据，统计 20 组随机数据在不同排序算法中运行的平均时间。

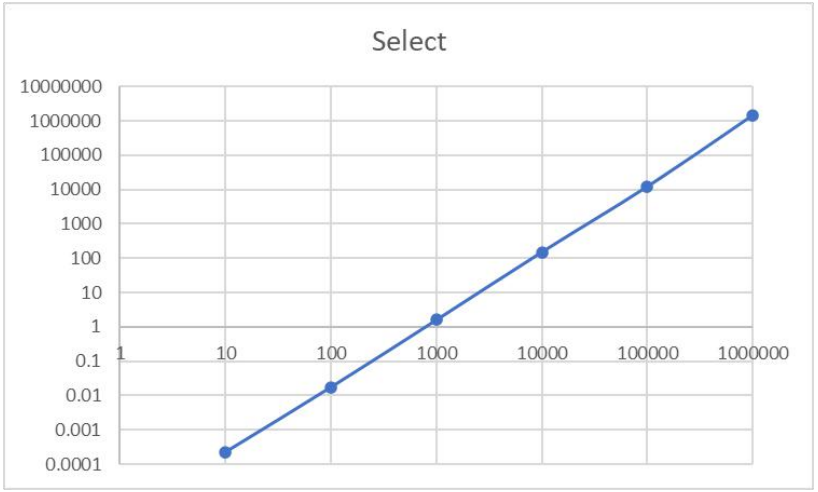
数据统计时，取规模为 10000 的实际运行时间为理论值。

1、选择排序：

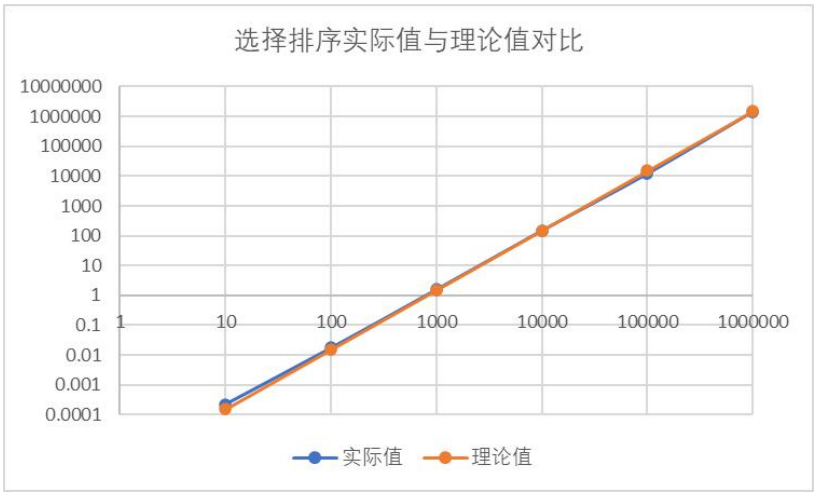
在不同规模下 20 组随机数的平均运行时间、理论运行时间及对数值如下表所示：

	A	B	C	D	E	F
1	规模n	规模对数	平均值	平均值对数	理论值	理论值对数
2	10	1	0.000226	-3.6458916	0.00015	-3.8244882
3	100	2	0.0175	-1.756962	0.01498	-1.8244882
4	1000	3	1.58	0.19865709	1.498	0.17551181
5	10000	4	149.8	2.17551181	149.8	2.17551181
6	100000	5	12110.8	4.08317283	14980	4.17551181
7	1000000	6	1442050	6.15898032	1498000	6.17551181

根据上述表格数据，做出平均运行时间与规模关系如下图所示（其中坐标已取对数表示）:



取 10000 的平均运行时间 149.8ms 为理论值，根据选择排序的时间复杂度:$O(n^2)$可以得到所有规模下的理论运行时间，与实际运行时间对比如下图所示:



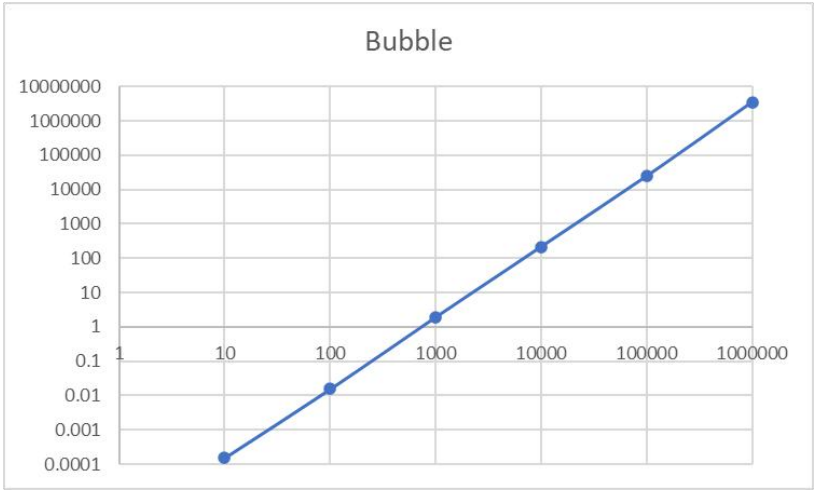
根据上图可以看出，选择排序实际值与理论值分别取对数表示后，其对应的图像基本重合。在规模小于 1000 时，实际值略高于理论值；在规模为 100000 时，实际值略小于理论值。

2、冒泡排序：

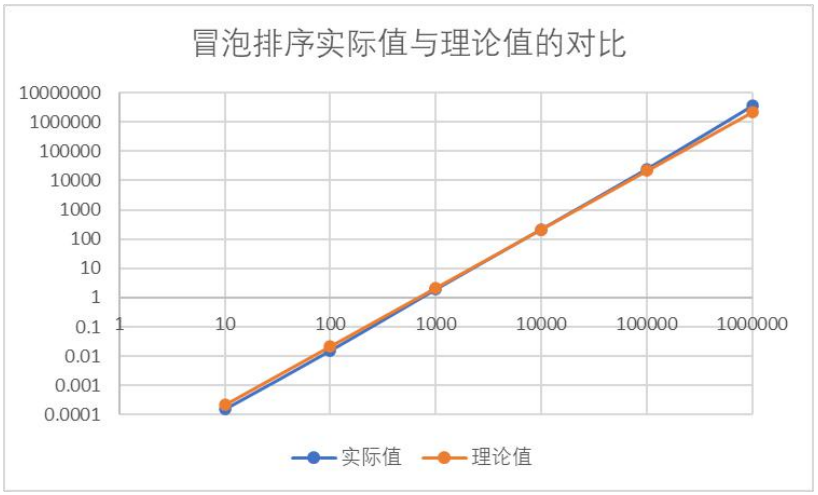
在不同规模下 20 组随机数的平均运行时间、理论运行时间及对数值如下表所示:

	A	B	C	D	E	F
1	规模n	规模对数	平均值	平均值对数	理论值	理论值对数
2	10	1	0.000153	-3.8153086	0.000217	-3.6632402
3	100	2	0.0154	-1.8124793	0.021715	-1.6632402
4	1000	3	1.93	0.28555731	2.1715	0.33675983
5	10000	4	217.15	2.33675983	217.15	2.33675983
6	100000	5	24741.45	4.39342515	21715	4.33675983
7	1000000	6	3504726	6.54465407	2171500	6.33675983

根据上述表格数据，做出平均运行时间与规模关系如下图所示（其中坐标已取对数表示）:



取 10000 的平均运行时间 217.15ms 为理论值，根据冒泡排序的时间复杂度:$O(n^2)$可以得到所有规模下的理论运行时间，与实际运行时间对比如下图所示:



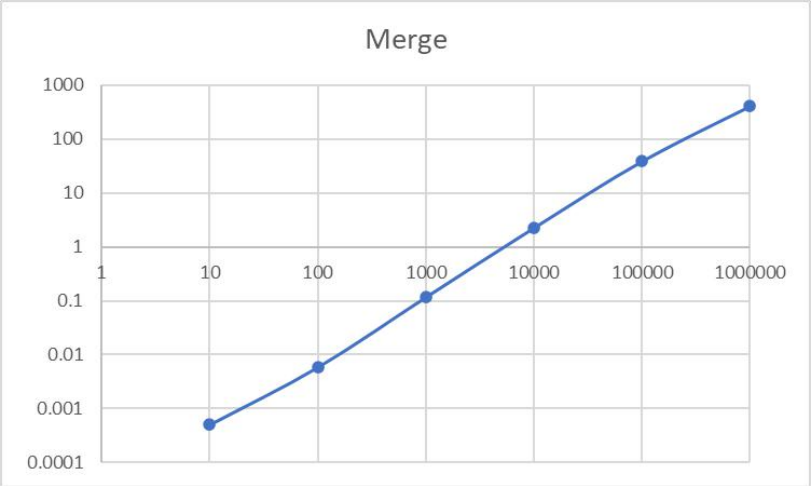
可以看到冒泡排序的实际运行时间与理论运行时间几乎重合，只在规模为 10 和规模为 1000000 时有微小的差别。

3、归并排序：

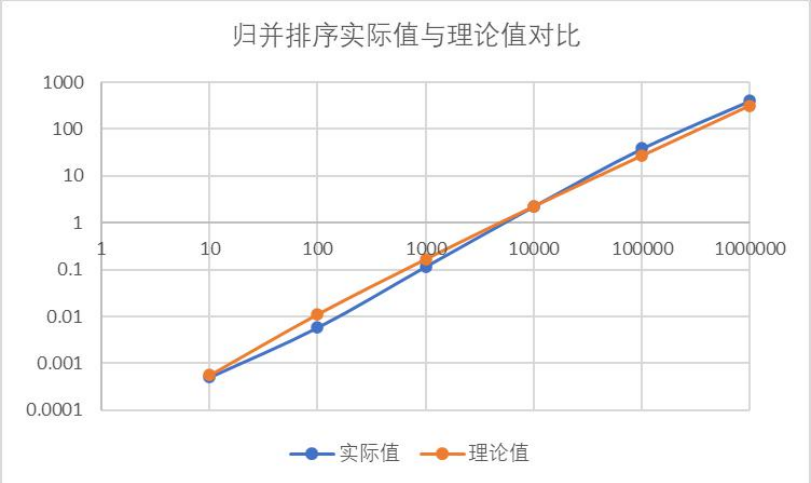
在不同规模下 20 组随机数的平均运行时间、理论运行时间及对数值如下表所示:

	A	B	C	D	E	F
1	规模n	规模对数	平均值	平均值对数	理论值	理论值对数
2	10	1	0.000505	-3.2967086	0.000565	-3.2479516
3	100	2	0.005874	-2.2310661	0.0113	-1.9469216
4	1000	3	0.117155	-0.9312392	0.1695	-0.7708303
5	10000	4	2.26	0.35410844	2.26	0.35410844
6	100000	5	38.55	1.58602438	27.12	1.43328969
7	1000000	6	404.45	2.60686484	316.4	2.50023647

根据上述表格数据，做出平均运行时间与规模关系如下图所示（其中坐标已取对数表示）:



取 10000 的平均运行时间 2.26ms 为理论值，根据归并排序的时间复杂度:$O(n\log_2 n)$ 可以得到所有规模下的理论运行时间，与实际运行时间对比如下图所示:



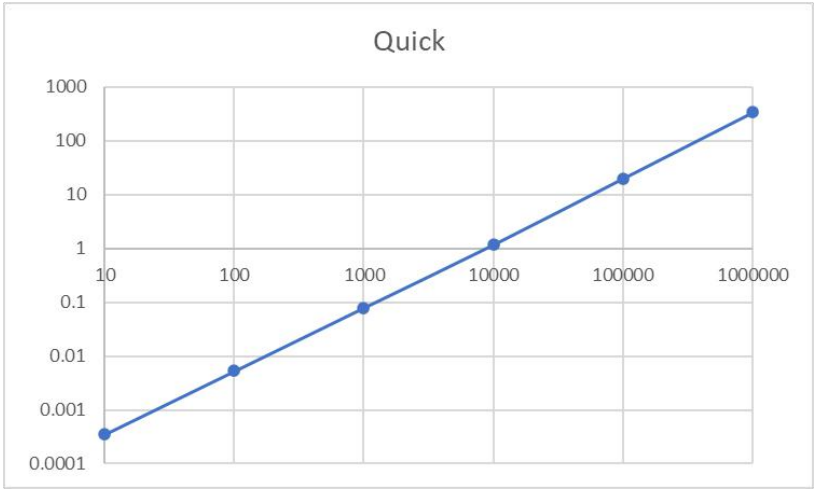
根据上图可以看出，在规模小于 10000 时，实际运行时间略低于理论运行时间，规模大于 10000 时，实际运行时间略高于理论运行时间。这可能是因为规模小时，产生的随机数比较接近，可能会出现待排序数中有相同的数，实际运行时间有所减小；而规模大之后，随机数可取范围也就增大，产生相同的随机数的可能也会降低，且数据值大，从而实际运行时间也有所增加。实验结果也从另一角度验证了时间复杂度 $O(n\log_2 n)$

4、快速排序：

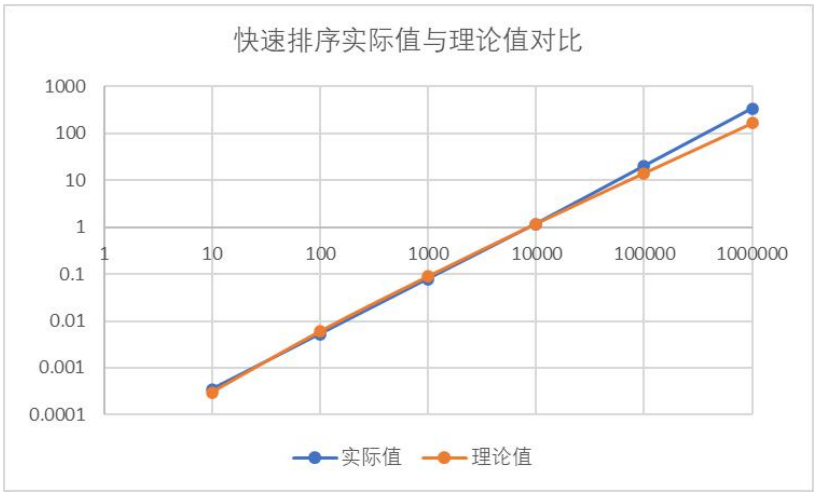
在不同规模下 20 组随机数的平均运行时间、理论运行时间及对数值如下表所示:

	A	B	C	D	E	F
1	规模n	规模对数	平均值	平均值对数	理论值	理论值对数
2	10	1	0.000351	-3.455312	0.000295	-3.5306199
3	100	2	0.00526	-2.2790143	0.005894	-2.2295899
4	1000	3	0.07917	-1.1014394	0.088413	-1.0534839
5	10000	4	1.17885	0.07145855	1.17885	0.07145855
6	100000	5	19.995	1.30092141	14.1462	1.15063979
7	1000000	6	336.3	2.52672687	165.039	2.21758658

根据上述表格数据，做出平均运行时间与规模关系如下图所示（其中坐标已取对数表示）:



取 10000 的平均运行时间 1.17885ms 为理论值，根据快速排序的时间复杂度:$O(n\log_2 n)$ 可以得到所有规模下的理论运行时间，与实际运行时间对比如下图所示:



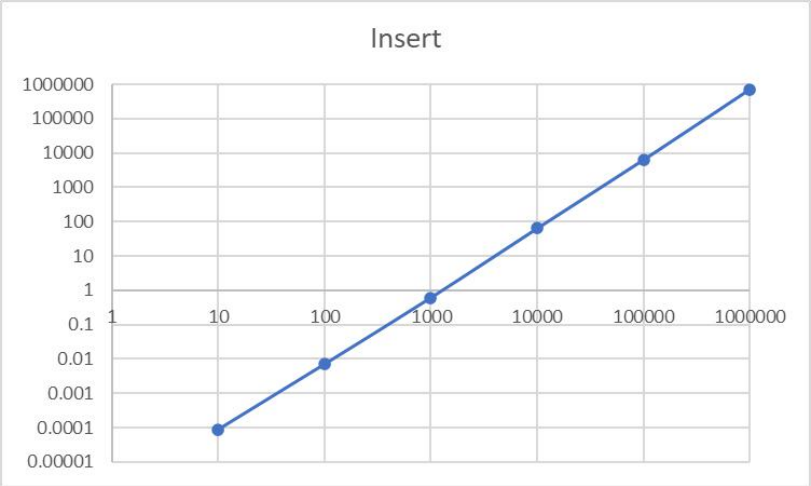
根据上图可以看出，在规模小于 10000 时，实际值与理论值基本重合，在规模大于 10000 时，实际值高于理论值。这可能是因为数据规模大，产生的随机数的无序性更加明显，而理论值时根据时间复杂度得到的结果，故在实际运行过程中所需要的时间会比理论分析时得到的运行时间更长。

5、插入排序：

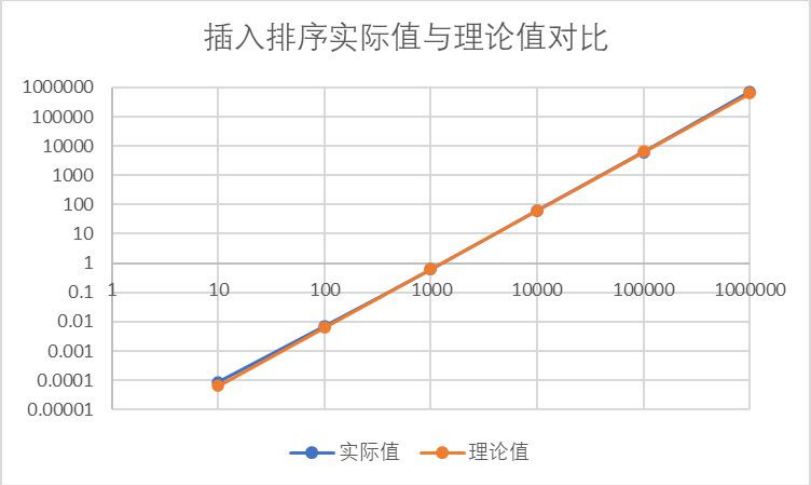
在不同规模下 20 组随机数的平均运行时间、理论运行时间及对数值如下表所示:

	A	B	C	D	E	F
1	规模n	规模对数	平均值	平均值对数	理论值	理论值对数
2	10	1	0.0000885	-4.0530567	0.00006385	-4.1948391
3	100	2	0.0071	-2.1487417	0.006385	-2.1948391
4	1000	3	0.6105	-0.2143143	0.6385	-0.1948391
5	10000	4	63.85	1.8051609	63.85	1.8051609
6	100000	5	6343.05	3.80229813	6385	3.8051609
7	1000000	6	716563.13	5.85525446	638500	5.8051609

根据上述表格数据，做出平均运行时间与规模关系如下图所示（其中坐标已取对数表示）:



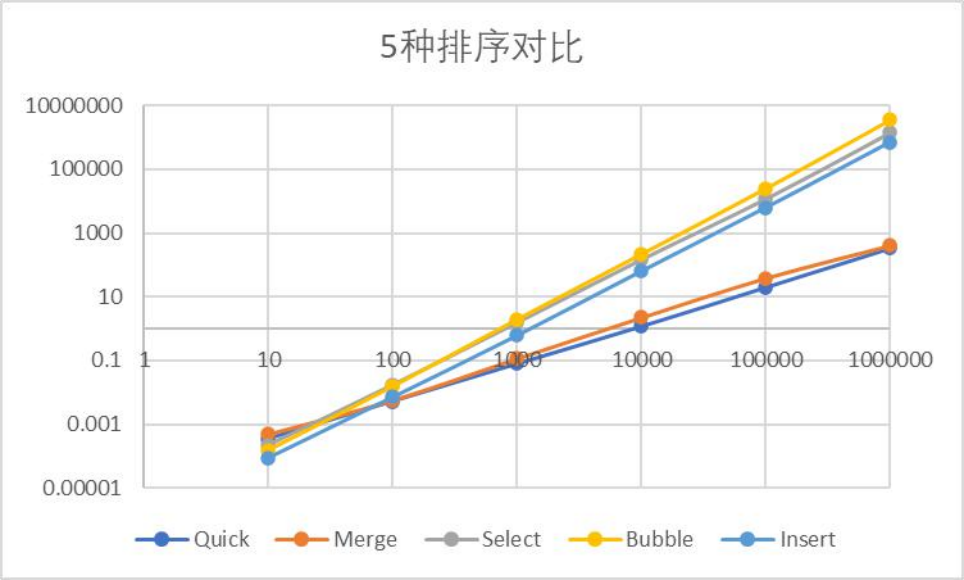
取 10000 的平均运行时间 123.6ms 为理论值，根据插入排序的时间复杂度:$O(n^2)$可以得到所有规模下的理论运行时间，与实际运行时间对比如下图所示:



根据上图可以看到，插入排序的实际运行时间与理论运行时间取对数坐标后，对比差异不大，甚至在规模从 100~10000 时几近重合。

6、五种不同算法的运行效率对比：

将 5 种算法在不同规模下的运行时间进行对比，做出图像如下图所示（其中坐标轴已经全部取对数刻度）:



根据上述结果，可以发现规模大概在 50 之后，快速排序和归并排序的平均运行时间明显小于另外三种排序所需的时间。而在冒泡、选择和插入这三种排序中，又可以观察得，冒泡所需的时间最长，插入在这三种算法中运行速度略快。在规模很小时，5 种排序算法的平均运行时间接近重合。

这也说明，对于规模小的数组，不同的排序算法所需的平均运行时间并没有太大的差别，在实际操作中或编写解决排序问题的代码时，我们可以选择算法简单的来实现小规模的排序；但当规模逐渐增大，快速排序和归并排序在运行速度上的优势就显现出来了，所以当碰到大规模的数组排序时，我们应该结合数据特点，选择快速排序或者归并排序来进行操作。

四、实验结论或体会

在进行实验的过程中，根据所遇到的问题和疑惑归纳出以下总结：

1、在进行快速排序、归并排序的运行时间统计时以及冒泡排序、插入排序、选择排序小规模下的运行时间统计时，由于运行速度快，时间小，在算法中采取重复当前规模同组随机数的排序过程，累计时间，使时间具可记录性，再取平均值从而完成对以上所提运行时间的统计。

2、生成随机数时，由于数值的存储也需要时间，为了使时间更加接近算法时间复杂度所呈现出来的规律，可以根据当前规模限制随机数生成的范围，使不同算法在不同规模下产生的随机数大小数量级相差不会太多，避免这一方面的计时误差。

3、根据冒泡、选择、插入实际平均运行时间的差别以及快速、归并实际平均运行时间的差别可以直到，对于时间复杂度相同的排序算法，在实际处理数据的过程中，运行时间也是有差别的。

4、实际操作时，根据时间复杂度及理论运行时间，我们直到冒泡、插入、选择在进行规模为 1000000 的排序时，每一组所需时间以小时计，尤其是冒泡排序 20 组数据总共的排序时间高达十几个小时。对于这种情况，首先应该保证算法的正确性，以免浪费时间；另外可以将计时代码写在循环内，从而可以时间观察每一组的实际运行时间，其他排序方法也可以利用这种方法，实现对每组数据的实际运行时间进行观察。

5、根据 5 种算法的最终比较，我们也可以得知，在进行不同规模的数组排序时，具体的运行时间有所不同，应当根据规模大小及序列特点选择合适的算法；学会针对不同的序列选择不同的算法。

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。