深 圳 大 学 实 验 报 告

课程名称：算法设计与分析

实验名称：多种排序算法的算法实现及性能比较

学院：计算机与软件学院 专业：计算机科学与技术

报告人：张跃 学号：**2015160180** 班级：**5**

同组人：无

指导教师：杨烜

实验时间：**2017/9/15——2017/9/16**

实验报告提交时间：

教务处制

一．实验目的

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．实验步骤与结果

实验总体思路：

1. 使用命令行参数改变每次排序测试所使用的排序算法、数据规模和随机种子值。
2. 性能测试方法使用获取执行排序函数执行前后的时钟时间，通过时间相减得到排序算法运行时间。
3. 为了控制无关因素，生成随机样本时通过控制随机种子值一致使每组样本数据一致。并且本次实验算法在**云服务器环境**上运行，排除减少硬件的无关负载的影响，使实验结果更有说服力。
4. 使用 shell 和 Python 脚本帮助自动化运行测试程序并统计数据。

各排序算法的实现及实验结果：

（注 1：以下代码为 C++代码。）

（注 2：图中显示的时间单位均为秒，图表中耗时均为平均消耗时间。）

（注 3：为更好的演示排序效果，使用命令行参数设置排序方法，数据量为 20000，随机种子值设置为 1。）

1、 选择排序

1.1 算法思路

每次挑选待排序数组的最值，与前置元素（在已排好序的数组末端）交换位置，然后继续挑选剩余元素的最值并重复操作。

1.2 代码实现

如图 1 所示。

```
// 选择排序
void SortingFunctions::selectionSort(int array[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        int min = i;
        // 内循环，每次循环时找出比array[min]小的元素，将其坐标赋给min作为新的最小元素值
        for (int j = i + 1; j < n; j++)
        {
            if (array[j] < array[min])
                min = j;
        }
        swap(array[i], array[min]);
    }
}
```

图 1 选择排序 C++实现

1.3 运行结果

如图 2 所示。20000 数据量用时 0.559405s。

```
Yorkson@Yorksons-Mac ~/Documents/Algorithms/Experience-1/src master ● ./main 1 20000 1
Selection sorting:
Data size: 20000 Time cost: 0.559405s
```

图 2 选择排序运行结果

1.4 运行速度测试及数据统计

以待排序数组的大小 n 为输入规模，固定 n，随机产生 20 组测试样本，统计排序算法在 20 个样本上的**平均运行时间**，n = 10000，20000，30000，40000，50000，以及 n = 100，1000，10000，100000，1000000

将数据绘制成图表如图 3 和图 4。

数据规模	10000	20000	30000	40000	50000
算法用时(s)	0.17033665	0.6984899	1.581282	2.8249735	4.389343
数据规模	100	1000	10000	100000	1000000
算法用时(s)	0.0000528	0.00183915	0.17521625	17.61875	1736.591

图 3 选择排序测试数据

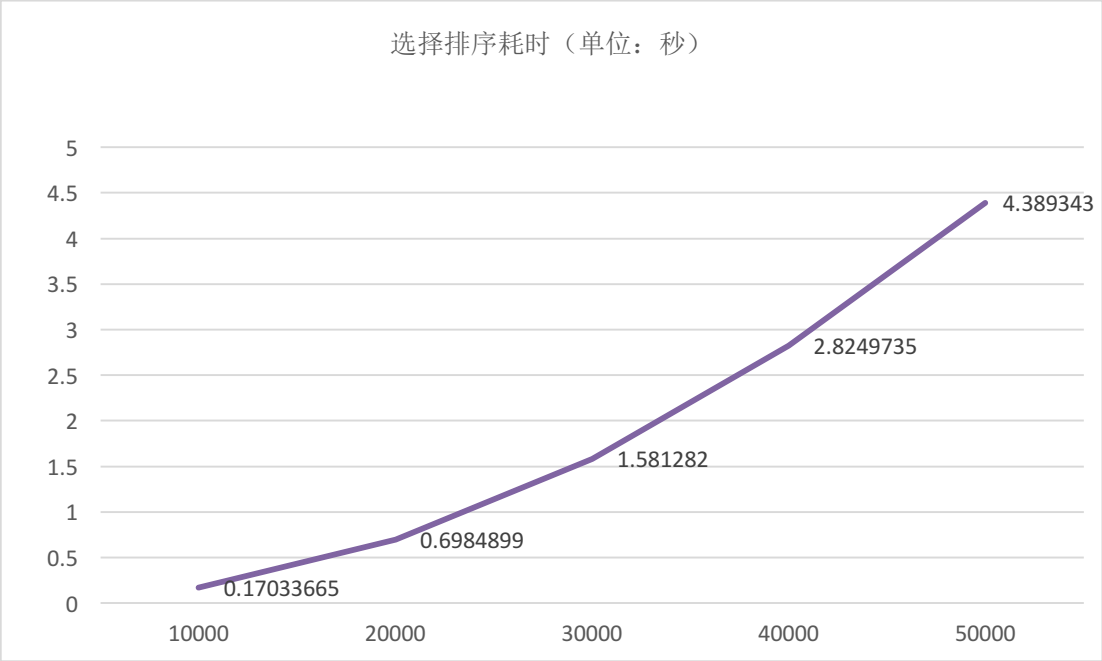

图 4 选择排序测试数据（10000 至 50000 数据量）

将 100 到 1000000 数据量的测试数据中的耗时取对数后描点绘线。得到的图表如图 5 所示。

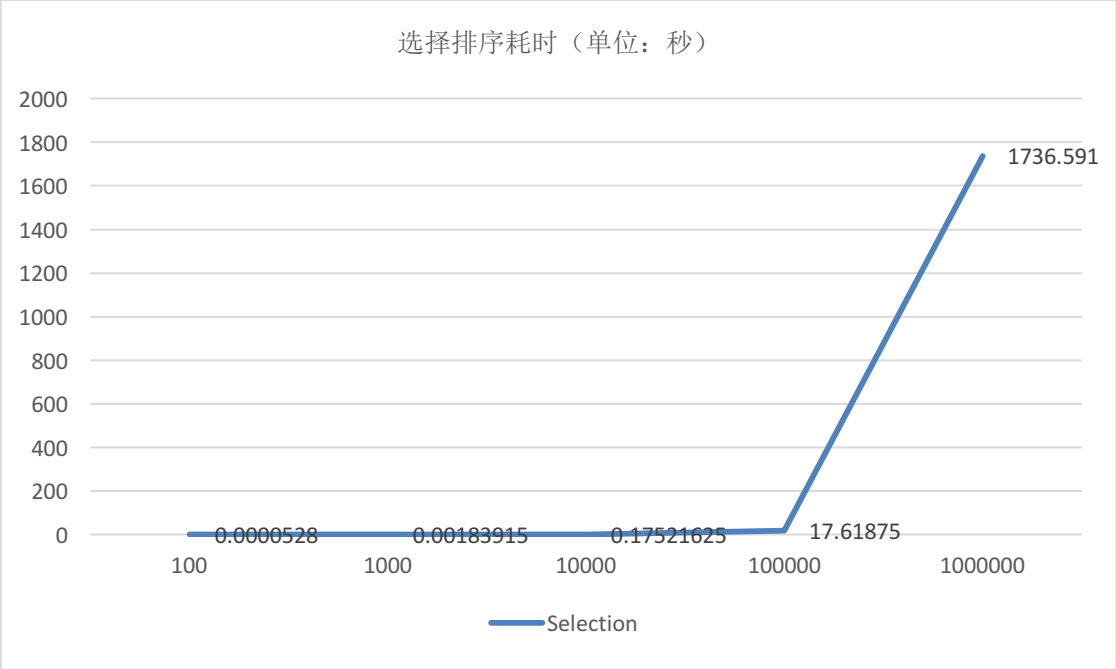

图 5 选择排序测试数据（100 至 1000000 数据量）

2、 冒泡排序

2.1 算法思路

冒泡排序通过多次重复两两比较每对相邻的元素，并按某种顺序交换他们，最终把数据排序。一直重复下去。算法的过程中每个循环的待排序数组中的最值就像一个个“气泡”冒到顶端。

2.2 代码实现

如图 6 所示。

```cpp
// 冒泡排序
void SortingFunctions::bubble(int array[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = n - 1; j > 0; j--)
        {
            if (array[j] < array[j - 1])
                swap(array[j], array[j - 1]);
        }
    }
}
```

图 6 冒泡排序 C++实现

2.3 运行结果

如图 7 所示。20000 数据量用时 1.877s。

```
Yorkson@Yorksons-Mac ~/Documents/Algorithms/Experience-1/src master ./main 2 20000 1
Bubble sorting:
Data size: 20000 Time cost: 1.877s
```

图 7 冒泡排序运行结果

2.4 运行速度测试及数据统计

以待排序数组的大小 n 为输入规模，固定 n，随机产生 20 组测试样本，统计排序算法在 20 个样本上的**平均运行时间**，n = 10000，20000，30000，40000，50000，以及 n = 100，1000，10000，100000，1000000

将数据绘制成图表如图 8 和图 9。

数据规模	10000	20000	30000	40000	50000
算法用时(s)	0.8901906	3.523429	7.87541	14.291035	22.093355
数据规模	100	1000	10000	100000	1000000
算法用时(s)	0.00011535	0.00759215	0.8799935	87.676805	9021.8885

图 8 冒泡排序测试数据

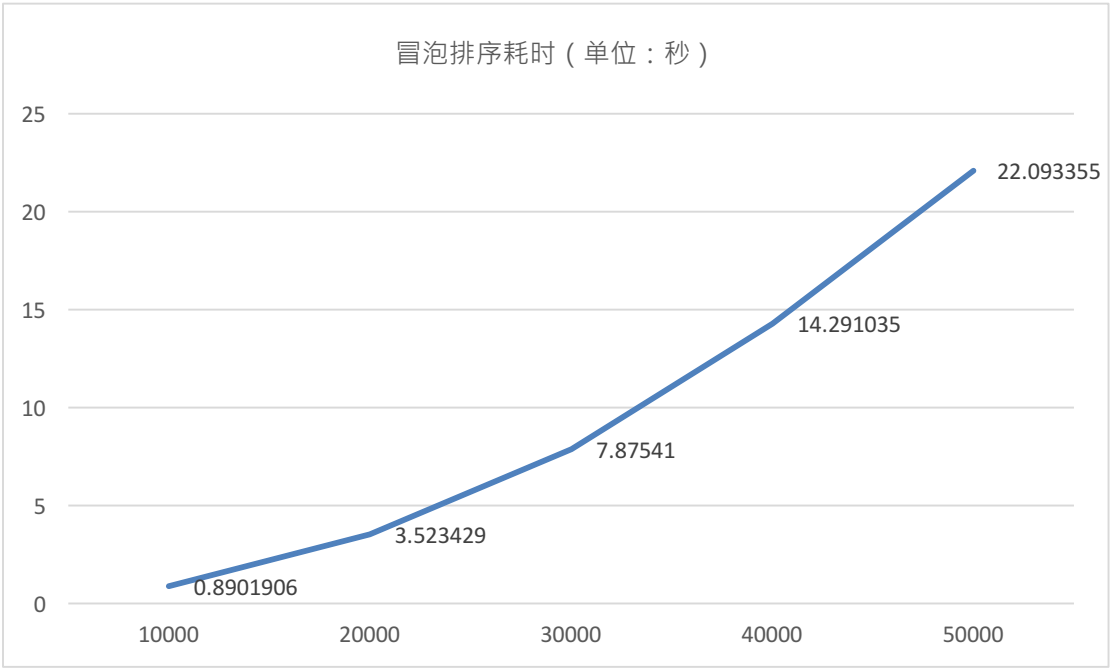

图 9 冒泡排序测试数据（10000 至 50000 数据量）

将 100 到 1000000 数据量的测试数据中的耗时取对数后描点绘线。得到的图表如图 10 所示。

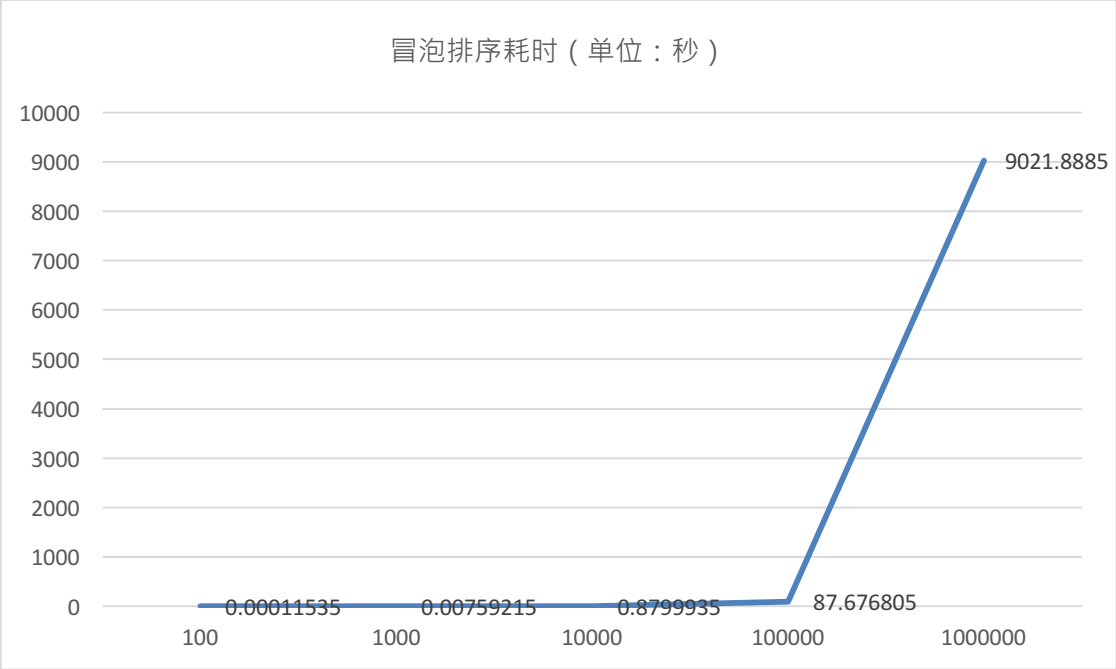

图 10　冒泡排序测试数据（100 至 1000000 数据量）

3、 合并排序

3.1　算法思路

归并排序基于分治的思想，将一个待排序序列分成两个长度相等的子序列，为每个子序列排序，然后再将两个序列合并起来。

3.2 代码实现

如图 11 所示。

```
// 归并排序
void SortingFunctions::mergeSort(int array[], int temp[], int left, int right)
{
    int middle = (left + right) / 2;
    if (left >= right)
        return;
    mergeSort(array, temp, left, middle);
    mergeSort(array, temp, middle + 1, right);
    for (int i = left; i <= right; i++)
        temp[i] = array[i];
    // 分前后两部分
    int j = left;
    int k = middle + 1;
    for (int curr = left; curr <= right; curr++)
    {
        if (j == middle + 1)
            array[curr] = temp[k++];
        else if (k > right)
            array[curr] = temp[j++];
        else if (temp[j] < temp[k])
            array[curr] = temp[j++];
        else
        {
            array[curr] = temp[k++];
        }
    }
}
```

图 11　归并排序 C++实现

3.3 运行结果

如图 12 所示。20000 数据量用时 0.004269s。

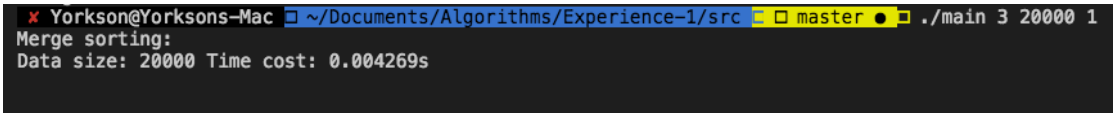

图 12 归并排序运行结果

3.4 运行速度测试及数据统计

以待排序数组的大小 n 为输入规模，固定 n，随机产生 20 组测试样本，统计排序算法在 20 个样本上的**平均运行时间**，n = 10000，20000，30000，40000，50000，以及 n = 100，1000，10000，100000，1000000，将数据绘制成图表如图 13 和图 14 。

数据规模	10000	20000	30000	40000	50000
算法用时(s)	0.00210685	0.00452215	0.00662955	0.0112717	0.01204225
数据规模	100	1000	10000	100000	1000000
算法用时(s)	0.0000447	0.0001811	0.0024956	0.02646335	0.2959137

图 13 归并排序测试数据

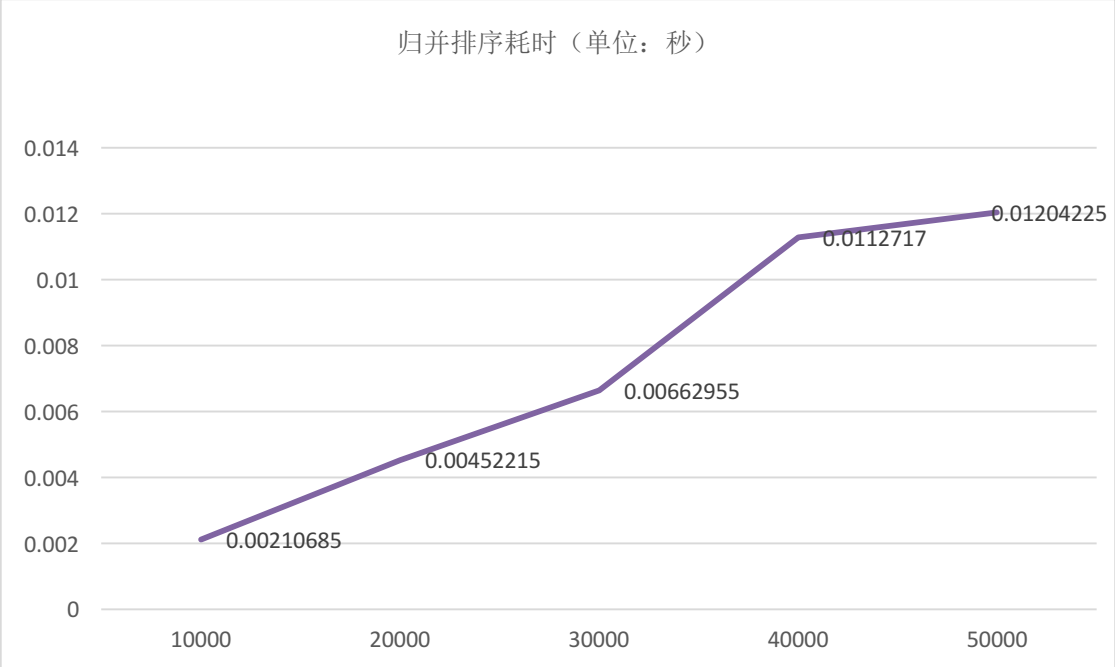

图 14 归并排序测试数据（10000 至 50000 数据量）

将 100 到 1000000 数据量的测试数据中的耗时取对数后描点绘线。得到的图表如图 15 所示。

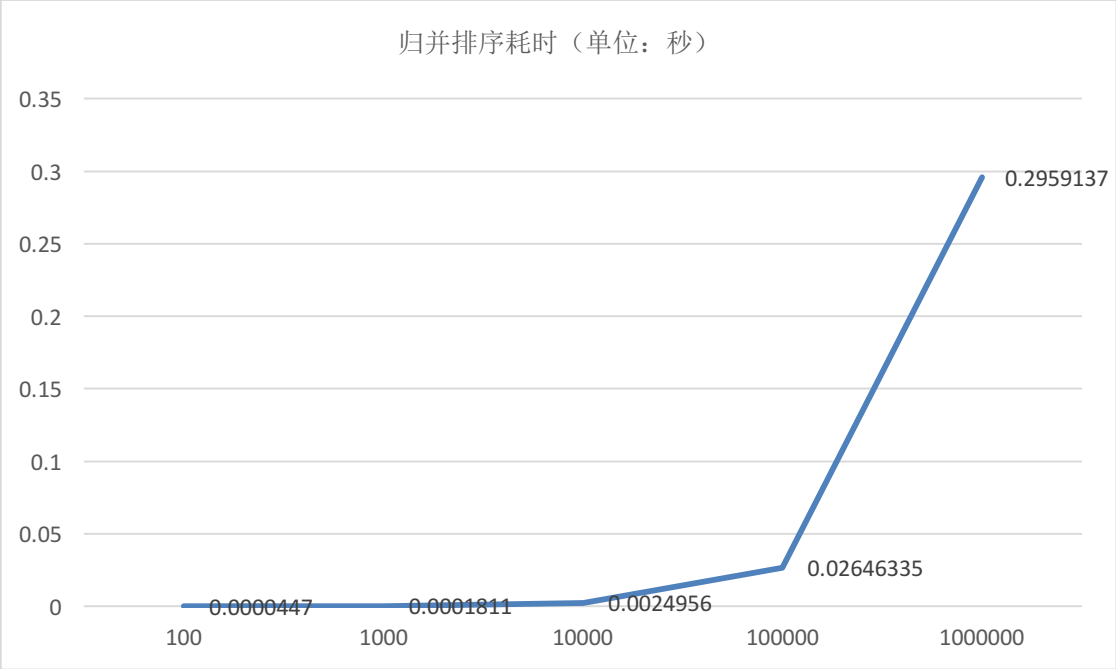

图 15　归并排序测试数据（100 至 1000000 数据量）

4、 快速排序

4.1　算法思路

（1）从数据列表中，选择一个元素，称为枢轴值

（2）重新排序列表，把所有数值小于枢轴的元素排到基准之前，所有数值大于基准的排基准之后(相等的值可以有较多的选择)。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区(partition)操作；

（3）分别递归排序较大元素的子列表和较小的元素的子列表。当列表元素个数<=1 时，递归结束。

4.2　代码实现

如图 16 所示。

```cpp
// 快速排序的一次循环子函数
// 将轴值放到数组的适当的位置
int SortingFunctions::partition(int array[], int left, int right)
{
    int pivot = (left + right) / 2;
    int tmp = array[pivot];
    swap(array[pivot], array[right]); // 把轴值放到最右
    int i = left;
    int j = right;
    while (1)
    {
        // 左边指针i向右移动，直到找到一个大于轴值tmp的值
        while (1)
        {
            // 如果i与j相遇则确定轴值位置，将当前下标返回
            if (i == j)
            {
                array[i] = tmp;
                return i;
            }
            // 若轴值左边元素大于轴值，则与轴值右边j下标元素互换
            if (array[i] > tmp)
            {
                array[j] = array[i];
                j--;
                break;
            }
            i++;
        }
        // 右边指针j向左移动，直到找到一个小于轴值tmp的值
        while (1)
        {
            // 如果i与j相遇则确定轴值位置，将当前下标返回
            if (i == j)
            {
                array[j] = tmp;
                return j;
            }
            // 若轴值右边元素小于轴值，则与轴值坐边i下标元素互换
            if (array[j] < tmp)
            {
                array[i] = array[j];
                i++;
                break;
            }
            j--;
        }
    }
}

// 快速排序
void SortingFunctions::quickSort(int array[], int left, int right)
{
    if (right <= left)
        return;
    int pivot = SortingFunctions::partition(array, left, right);
    quickSort(array, left, pivot - 1);
    quickSort(array, pivot + 1, right);
}
```

图 16　快速排序 C++实现

4.3 运行结果

如图 17 所示。数据量 20000 用时 0.002996s。

```
✘ Yorkson@Yorksons-Mac ~/Documents/Algorithms/Experience-1/src master ● ./main 4 20000 1
Quick sorting:
Data size: 20000 Time cost: 0.002996s
```

图 17 快速排序运行结果

4.4 运行速度测试及数据统计

以待排序数组的大小 n 为输入规模，固定 n，随机产生 20 组测试样本，统计排序算法在 20 个样本上的**平均运行时间**，n = 10000，20000，30000，40000，50000，以及 n = 100，1000，10000，100000，1000000，将数据绘制成图表如图 18 和图 19。

数据规模	10000	20000	30000	40000	50000
算法用时(s)	0.0014632	0.00371065	0.00477245	0.0070987	0.0087756
数据规模	100	1000	10000	100000	1000000
算法用时(s)	0.0000352	0.00017485	0.00141265	0.0185158	0.2124974

图 18 快速排序测试数据

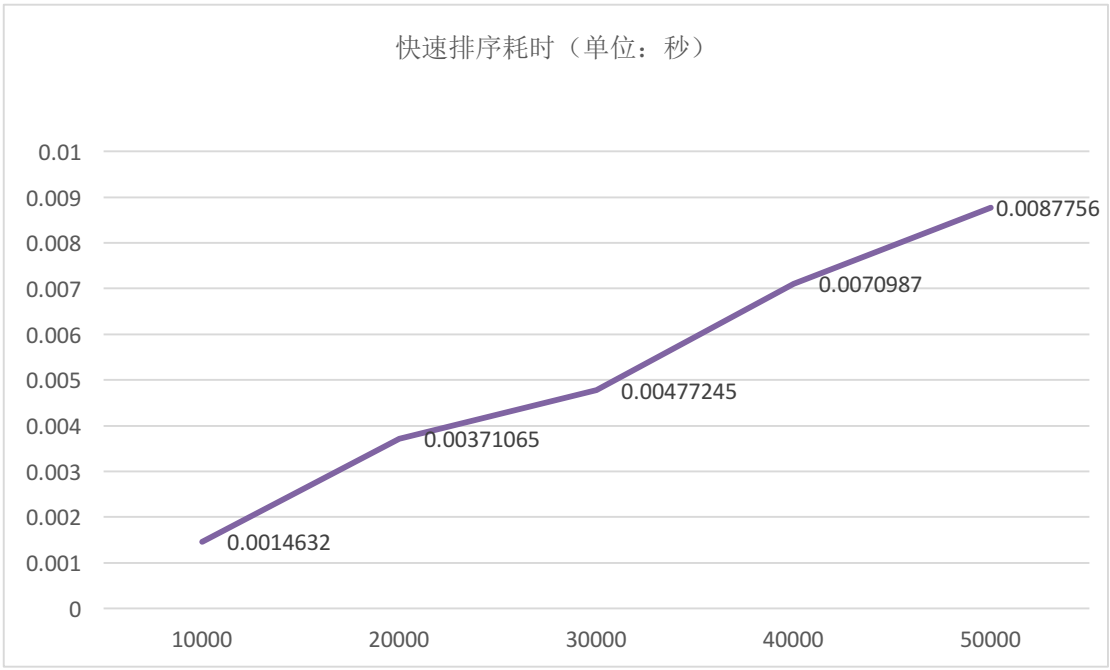

图 19 快速排序测试数据（10000 至 50000 数据量）

将 100 到 1000000 数据量的测试数据中的耗时取对数后描点绘线。得到的图表如图 20 所示。

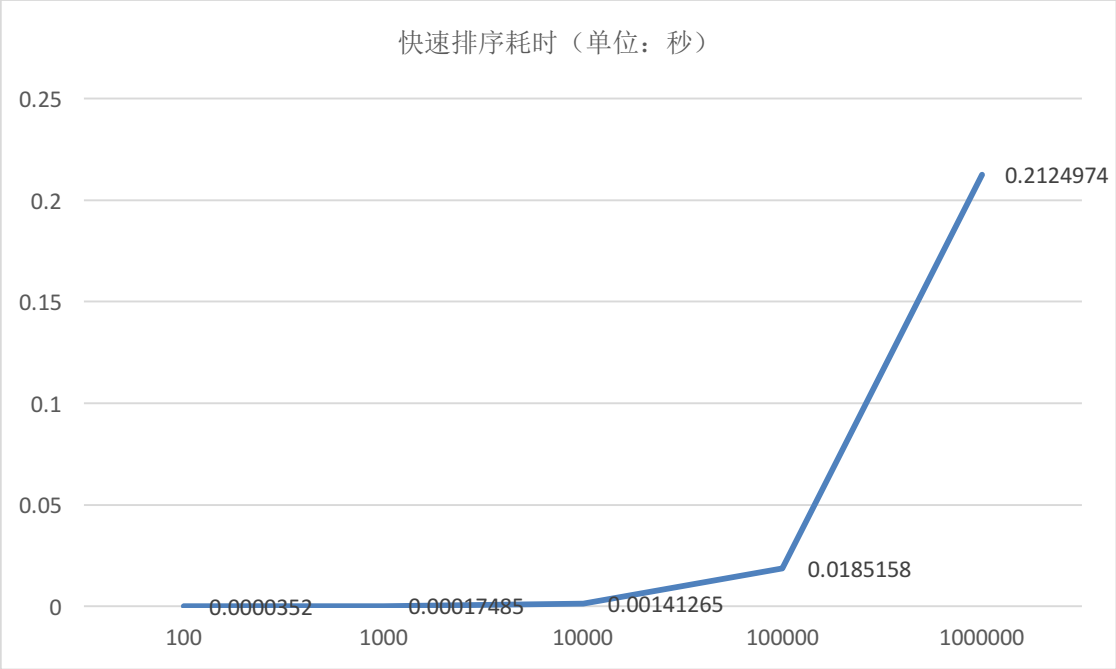

图 20　快速排序测试数据（100 至 1000000 数据量）

5、 插入排序

5.1 算法思路

插入排序逐个处理待排序的记录，每个新纪录都与前面已排序的子序列进行比较，并找到相应的位置进行插入。可归纳为以下步骤：

（1）设待排序数组为 a[0, 1, …, n-1]，开始时，令 a[0]成为 1 个有序序列（即将第一个元素看成已经被排序），未排序的为 a[1..n-1]。令 i=1；

（2）将 a[i]插入当前的有序区 a[0,…,i-1]中，形成 a[0,…,i]的有序区间。

（3）i++，重复第二步直到 i==n-1，排序完成。

5.2 代码实现

如图 21 所示。

```
// 插入排序
void SortingFunctions::insertSort(int array[], int n)
{
    for (int i = 1; i < n; i++)
    {
        for (int j = i; j > 0; j--)
        {
            if (array[j] < array[j - 1])
                swap(array[j - 1], array[j]);
            else
                break;
        }
    }
}
```

图 21　插入排序 C++实现

5.3 运行结果

如图 22 所示。

```
✘ Yorkson@Yorksons-Mac ~/Documents/Algorithms/Experience-1/src master ● ./main 5 20000 1
Insert sorting:
Data size: 20000 Time cost: 0.657911s
```

图 22 插入排序运行结果

5.4 运行速度测试及数据统计

以待排序数组的大小 n 为输入规模，固定 n，随机产生 20 组测试样本，统计排序算法在 20 个样本上的**平均运行时间**，n = 10000，20000，30000，40000，50000，以及 n = 100，1000，10000，100000，1000000，将数据绘制成图表如图 23 和图 24。

数据规模	10000	20000	30000	40000	50000
算法用时(s)	0.410771	1.6665865	3.79498	6.6102355	10.3750545
数据规模	100	1000	10000	100000	1000000
算法用时(s)	0.00008245	0.00405375	0.3780098	40.885905	4260.82

图 23 插入排序测试数据

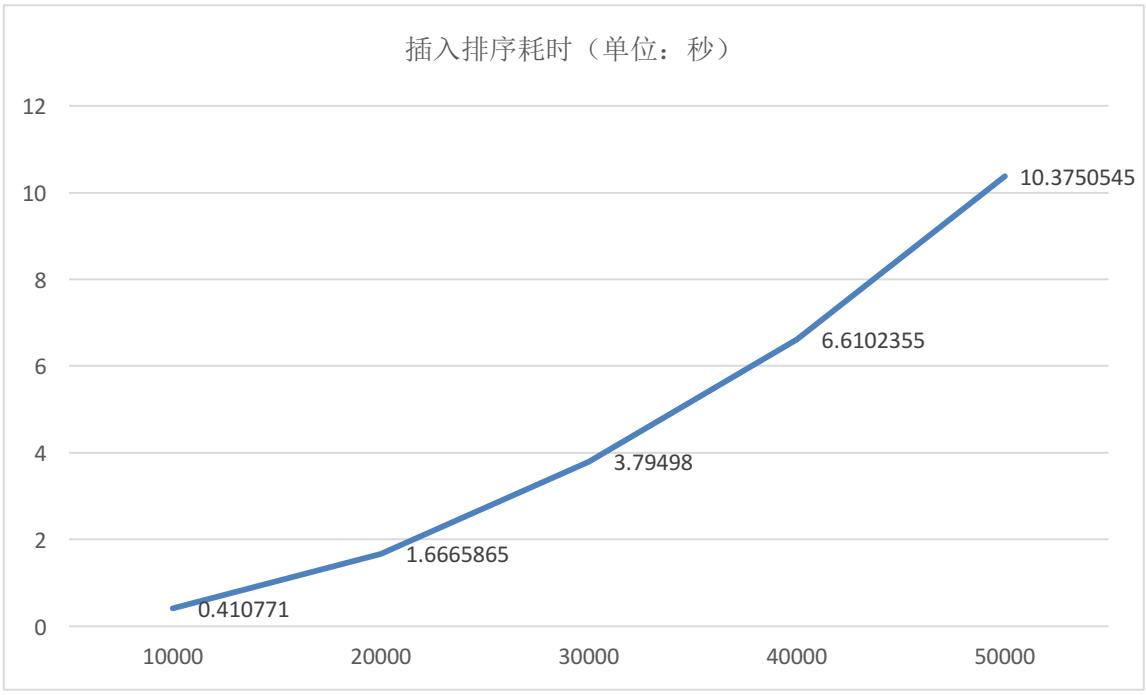

图 24 插入排序测试数据（10000 至 50000 数据量）

将 100 到 1000000 数据量的测试数据中的耗时取对数后描点绘线。得到的图表如图 25 所示。

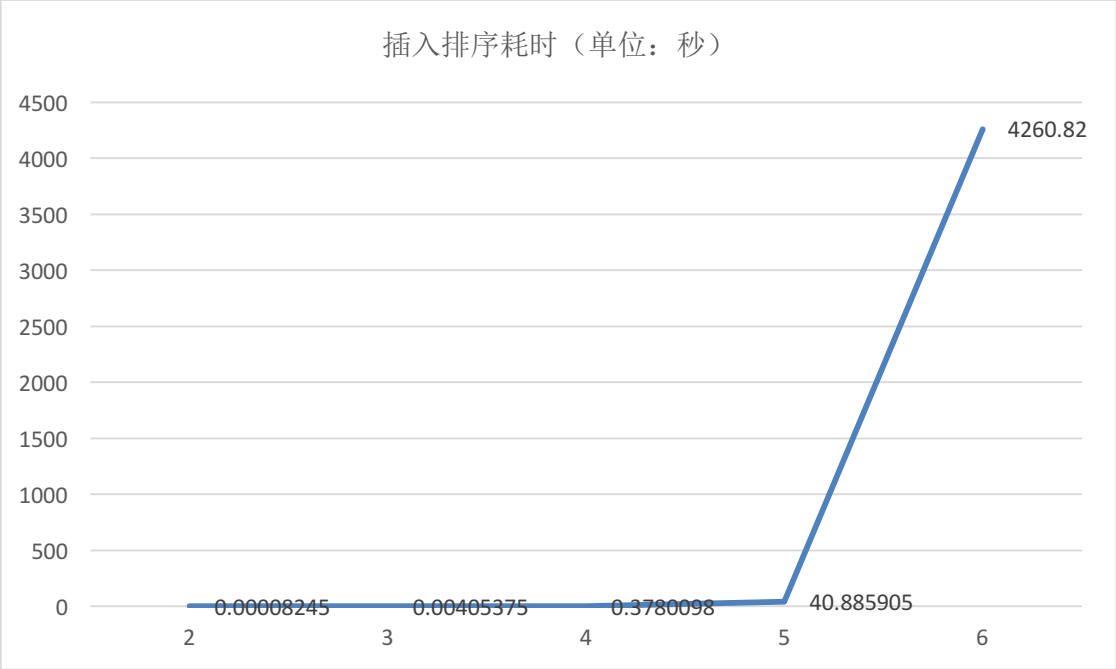

图 25　插入排序测试数据（100 至 1000000 数据量）

三．实验分析

五种排序算法中，大致可以分为两类，平均时间复杂度为 O（n2）的和 O（nlogn）的

其中选择排序、冒泡排序、插入排序平均复杂度均为 O（n^2），快速排序、归并排序均为 O（nlogn）复杂度。

O（n^2）：

将选择排序、冒泡排序、插入排序三种排序的测试结果描点绘制在同一图表，如图所示。

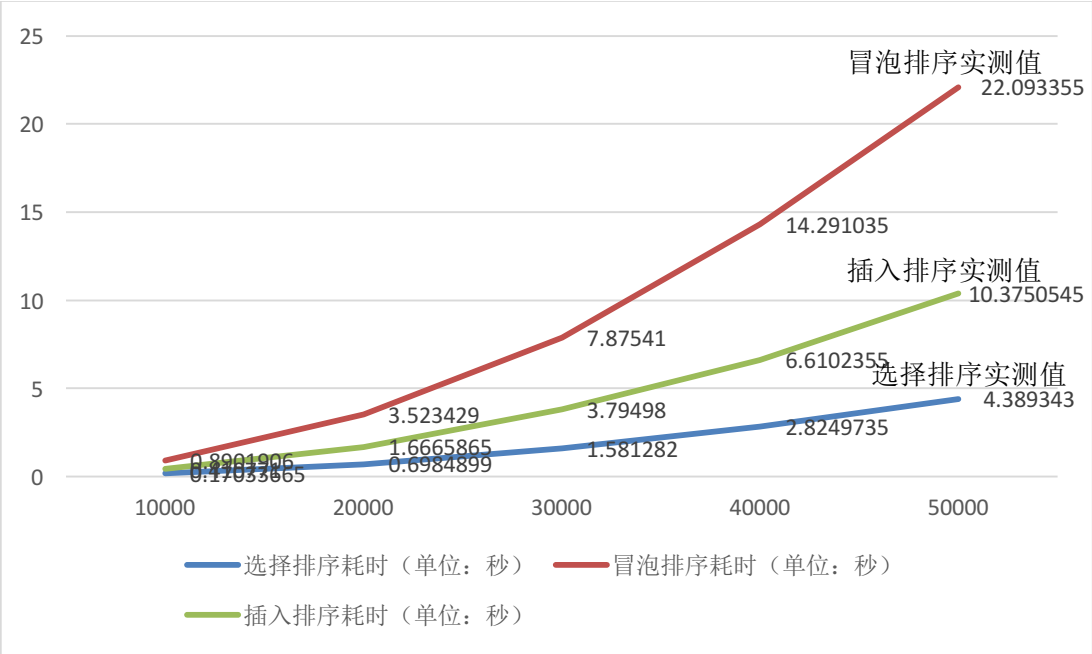

图 26 复杂度为 O（n^2）的选择排序、冒泡排序、插入排序测试结果

然后进行理论对比分析。联系理论与实际进行对比，算法耗时与数据规模的关系可以用函数 $T = n^2$ 表示，横坐标增加 10000，即 n 扩大 n 倍，则 T 扩大 n^2 倍。以 n = 10000 为基准，计算理论耗时，制作表格如下图所示。

	10000	20000	30000	40000	50000
冒泡理论值	0.8901906	3.5607624	8.0117154	14.2430496	22.254765
插入理论值	0.410771	1.643084	3.696939	6.572336	10.269275
选择理论值	0.17033665	0.6813466	1.53302985	2.7253864	4.25841625

图 27　计算理论值

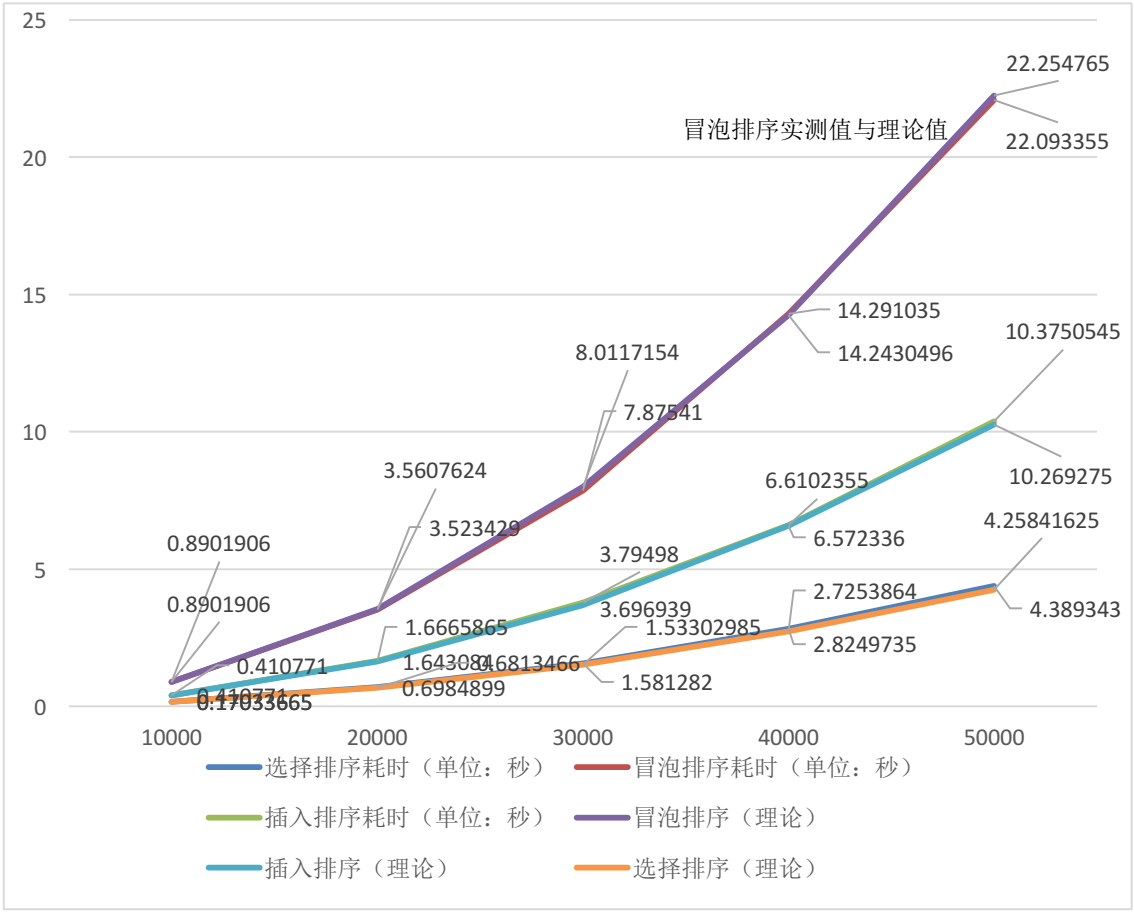

图 28　三种算法的理论值与实测值比较（以 10000 数据量为基准）

由图可以得出三种算法的理论耗时与实际耗时十分贴合。与 n 的关系为二次函数，这与理论复杂度是相符合的。

将横坐标改写成 10 的 k 次方的形式，k = 2，3，4，5，6。如图 29 所示。

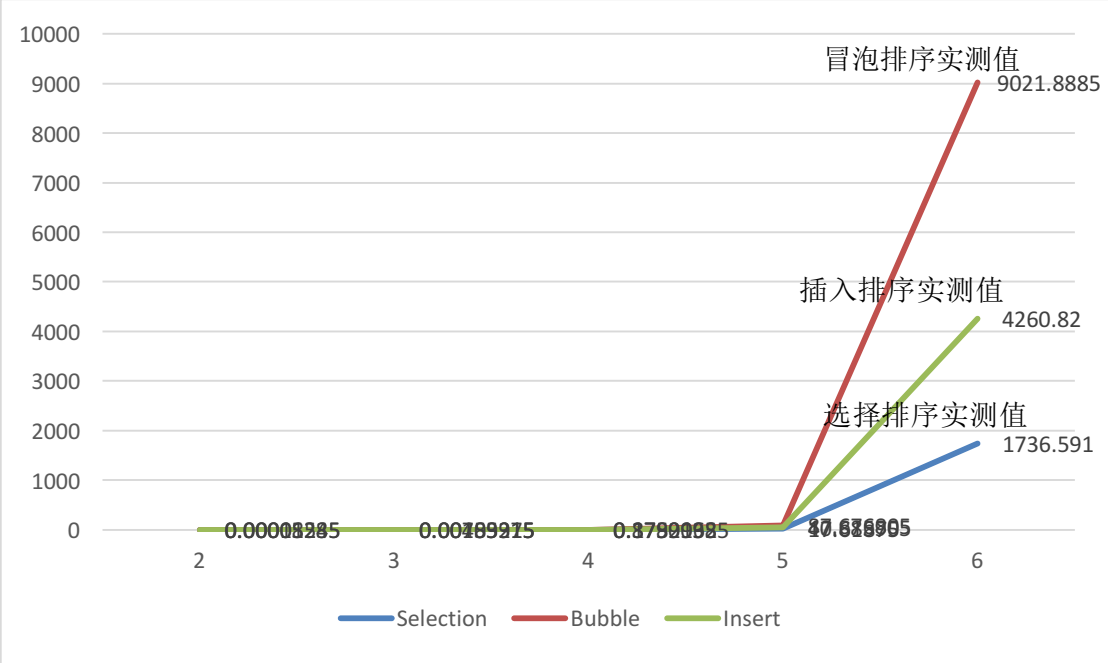

图 29　复杂度为 O（n2）的选择排序、冒泡排序、插入排序测试结果

分析：由上图可以发现冒泡排序算法耗时最多，插入排序次之，选择排序最小。**原因是冒泡排序和插入排序的元素移动方面效率非常低下，其中冒泡排序的比较次数为（n-1）*（n-1）约等于 n^2，而插入排序为 n！，约等于 $0.5n^2$，故冒泡排序效率比插入排序低。选择排序的移动效率最高，平均移动次数最少，故选择排序效率最高。**

然后进行理论对比分析。联系理论与实际进行对比，算法耗时与数据规模的关系可以用函数 $T = n^2$ 表示，横坐标增加 1，即 n 扩大 10，则 T 扩大 100。以 n = 10000 为基准，计算理论耗时，制作表格如下图所示。

	2	3	4	5	6
Bubble	8.79994E-05	0.008799935	0.8799935	87.99935	8799.935
Insert	3.7801E-05	0.003780098	0.3780098	37.80098	3780.098
Selection	1.75216E-05	0.001752163	0.17521625	17.521625	1752.1625

图 30　计算理论值

将上表的数据描点绘制在图 29 上，如图 31 所示。

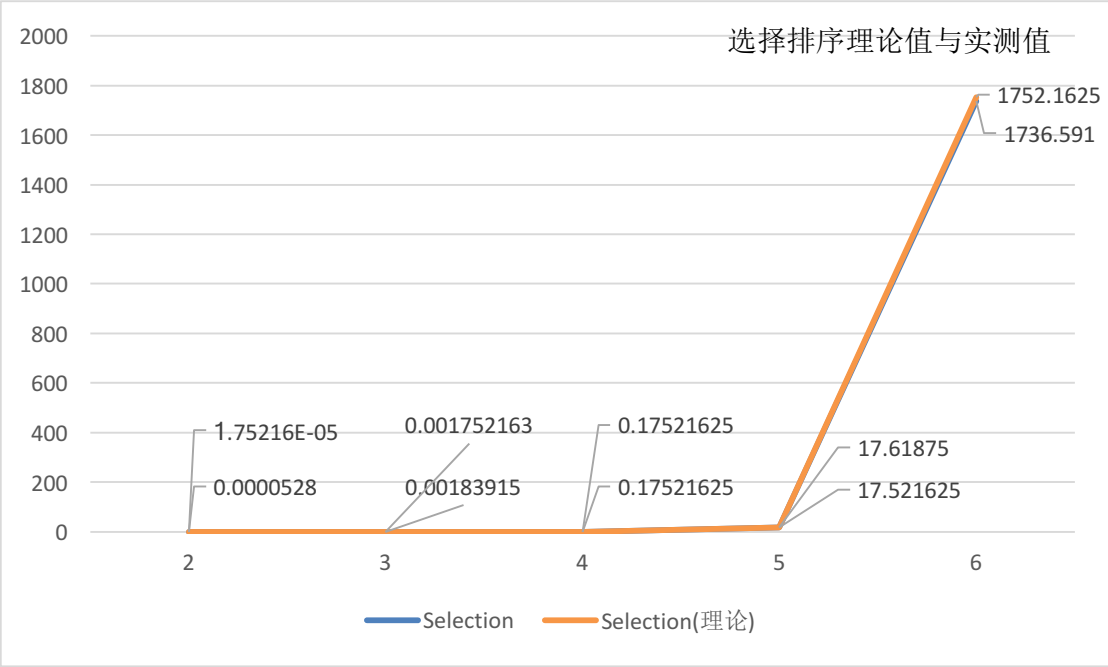

图 31　分析理论值与实测值差异

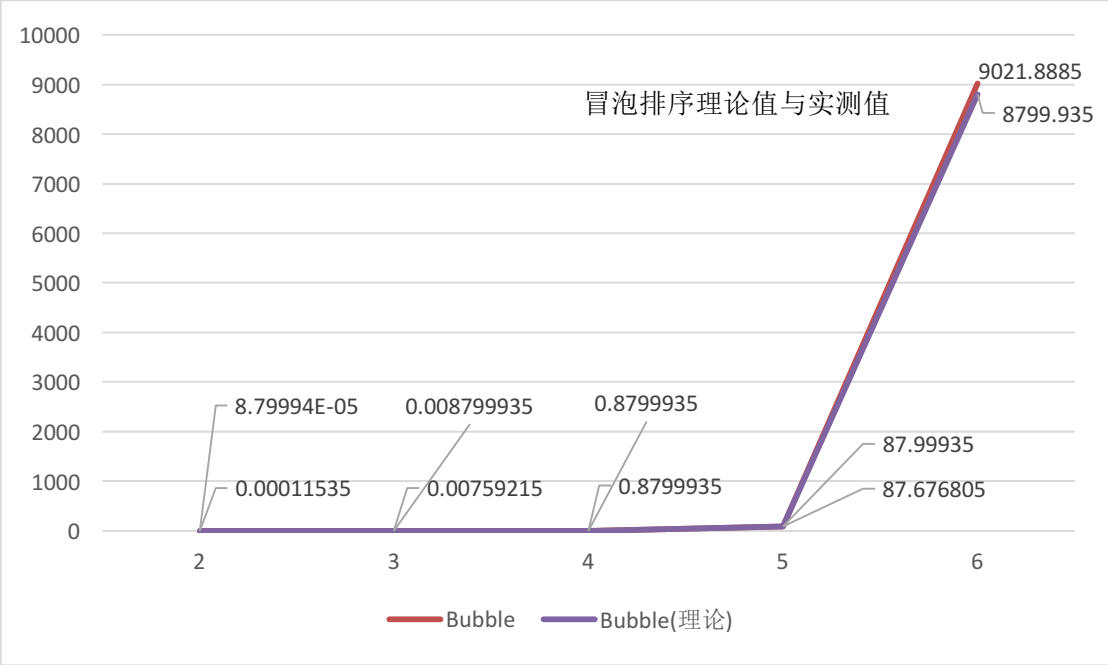

图 32　分析理论值与实测值差异

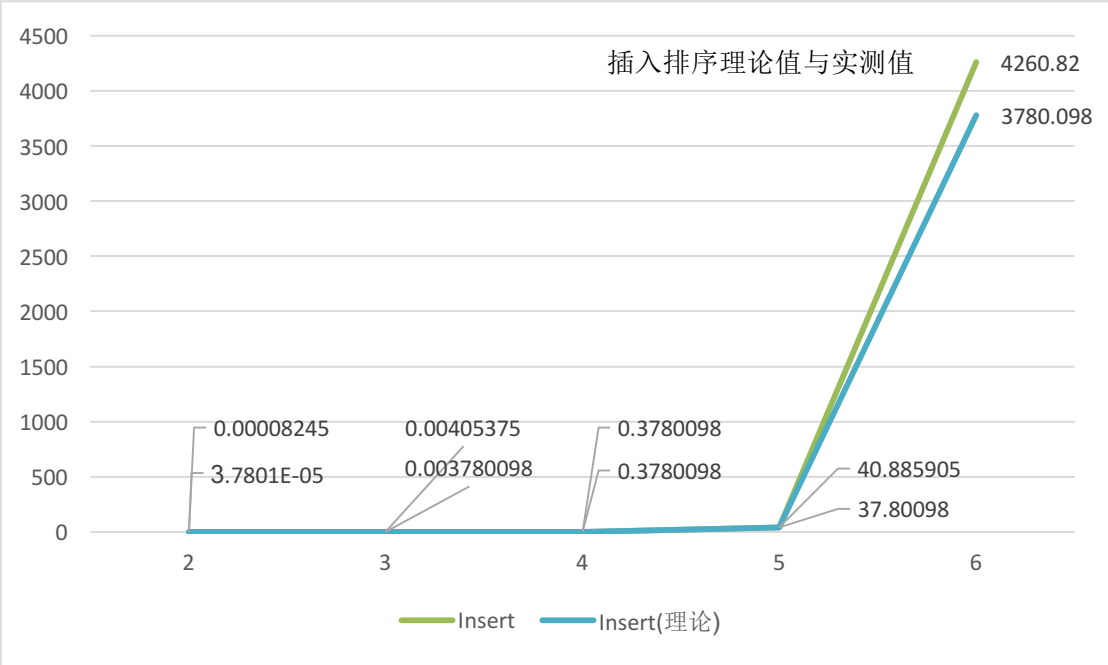

图 33　分析理论值与实测值差异

分析：由上图可以看出实测值与理论值基本相似。观察发现当数据量较大时只有插入排序实际值比理论值高的。经推测，可能是由于数据量太，此时算法的耗时除了理论耗时，还有插入排序时的交换赋值操作。插入排序的赋值操作是比较操作的次数加上 (n-1)次。同时也说明当数据量较大时，使用插入排序不是一个好的算法。

O（nlogn）：

将归并排序、快速排序的测试结果描点绘制在同一图表，如图 34 所示。

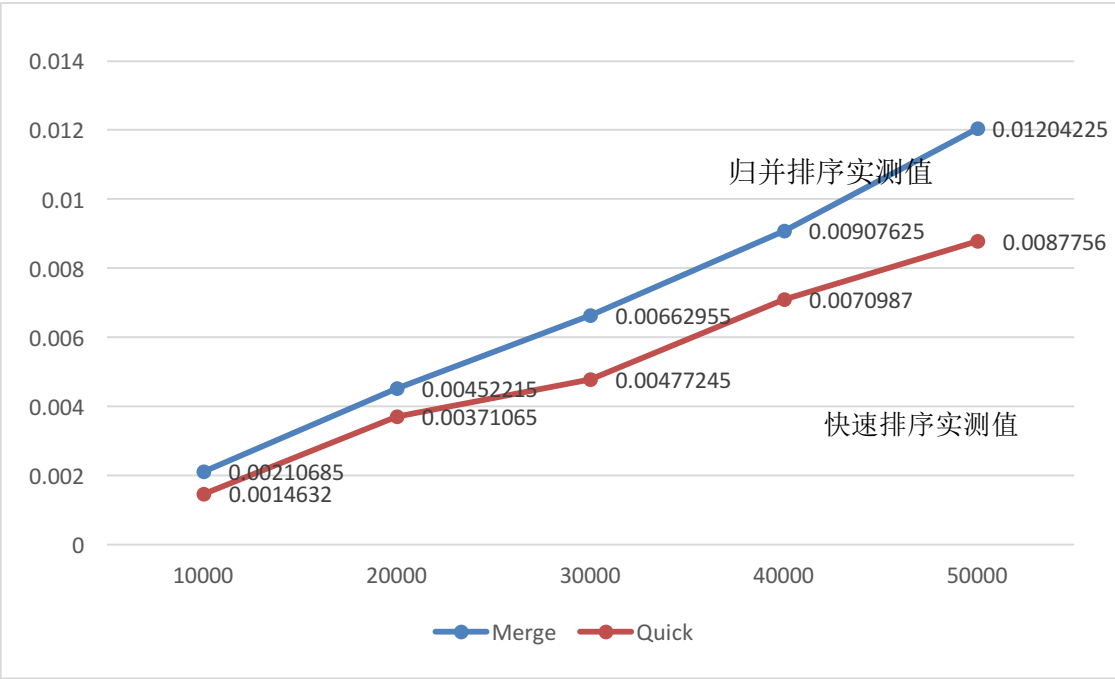

图 34　归并排序、快速排序实测数据

分析上图可得算法耗时大致呈线性趋势。快速排序效率比较高一点。归并排序和快速排序都采用了分而治之的思想。

然后进行理论对比分析。联系理论与实际进行对比，算法耗时与数据规模的关系可以用函数 T = nlogn 表示，横坐标增加 1，即 n 扩大 10，则 T 扩大 10 倍并加 10n。以 n = 10000 为基准，计算理论耗时，制作表格如下图所示。

数据量	10000	20000	30000	40000	50000
Merge 理论	0.00210685	0.004530813	0.007074467	0.00969585	0.012375031
Quick 理论	0.0014632	0.003146634	0.004913193	0.006733734	0.008594416

图 35　计算理论值

将上表的数据描点绘制在图表上，如图 36 所示。

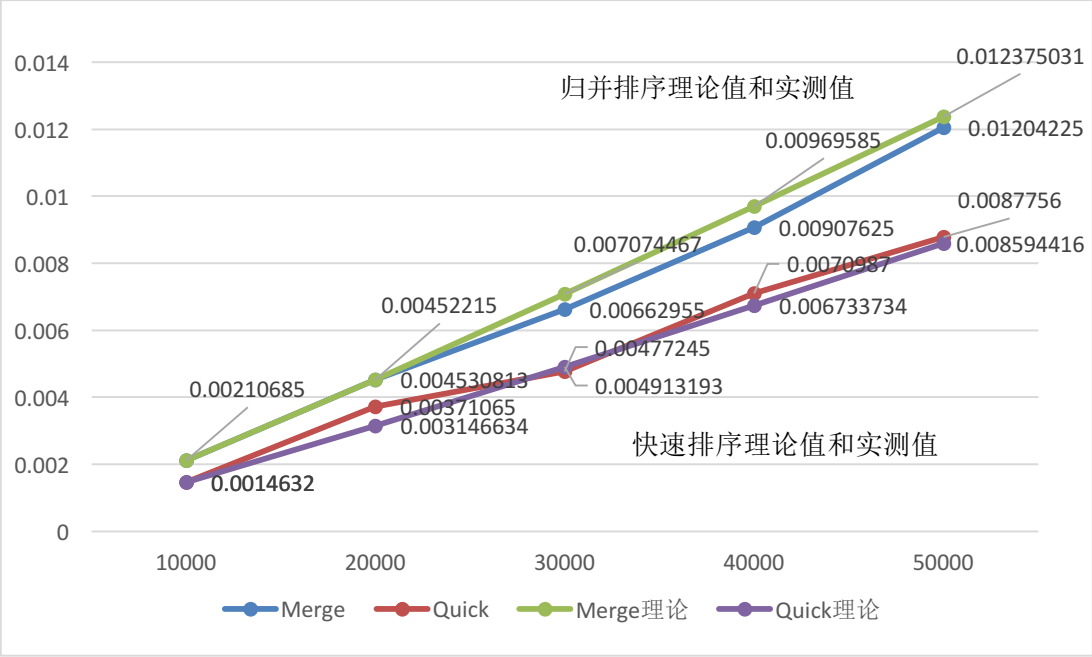

图 36　分析理论值与实测值差异（以数据量 10000 为基准）

由图可以得出三种算法的理论耗时与实际耗时十分贴合。与 n 的关系为 nlogn 函数，这与理论复杂度是相符合的。

将横坐标改写成 10 的 k 次方的形式，k = 2，3，4，5，6。如图 37 所示。

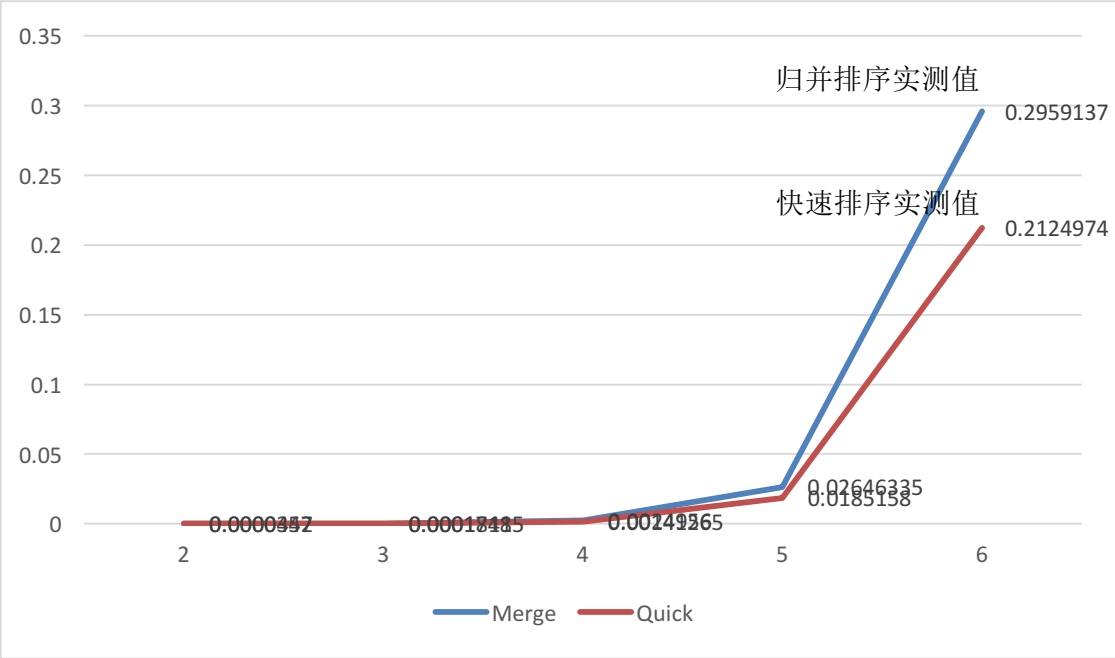

图 37　归并排序、快速排序实测数据

根据上图分析，合并排序需要赋值的操作较多，受输入数据的影响比快排大，所以当数据规模较大时，不受输入数据影响的快速排序更快。

接着进行理论对比分析。联系理论与实际进行对比，算法耗时与数据规模的关系可以用函数 T = nlogn 表示，横坐标增加 1，即 n 扩大 10，**则 T 扩大（10+10 / lgn）倍**。以 n = 10000 为基准，计算理论耗时，制作表格如下图所示。

数据量	100	1000	10000	100000	1000000
Merge 理论	0.000012478	0.00018717	0.0024956	0.031195	0.37434
Quick 理论	7.06325E-06	0.000105949	0.00141265	0.017658125	0.2118975

图 38　计算理论值（以数据量 10000 为基准）

将上表的数据描点绘制在图表上，如图 38 所示。

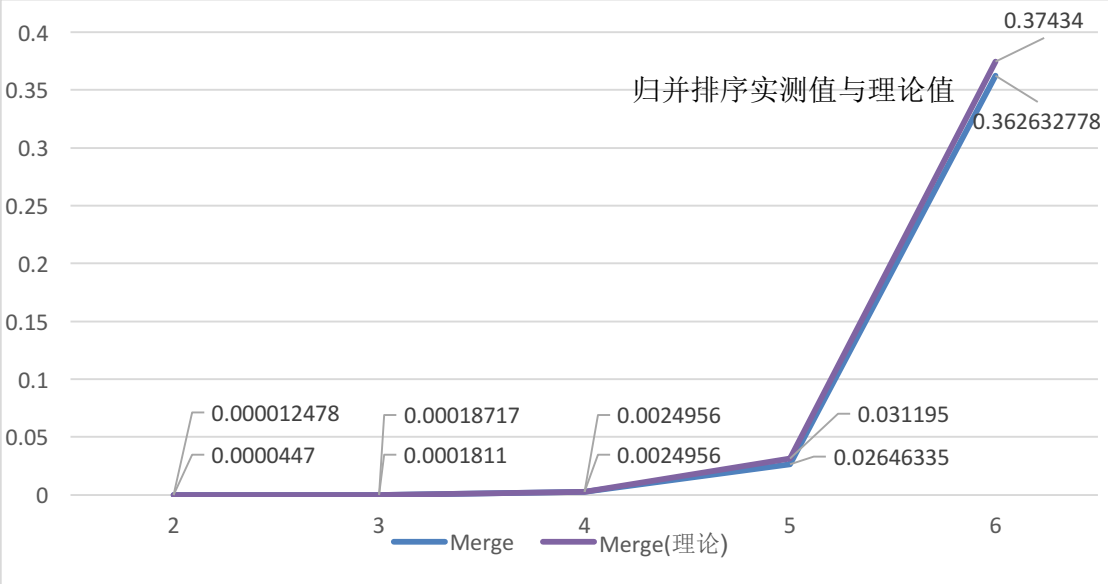

图 39　分析理论值与实测值差异（以数据量 10000 为基准）

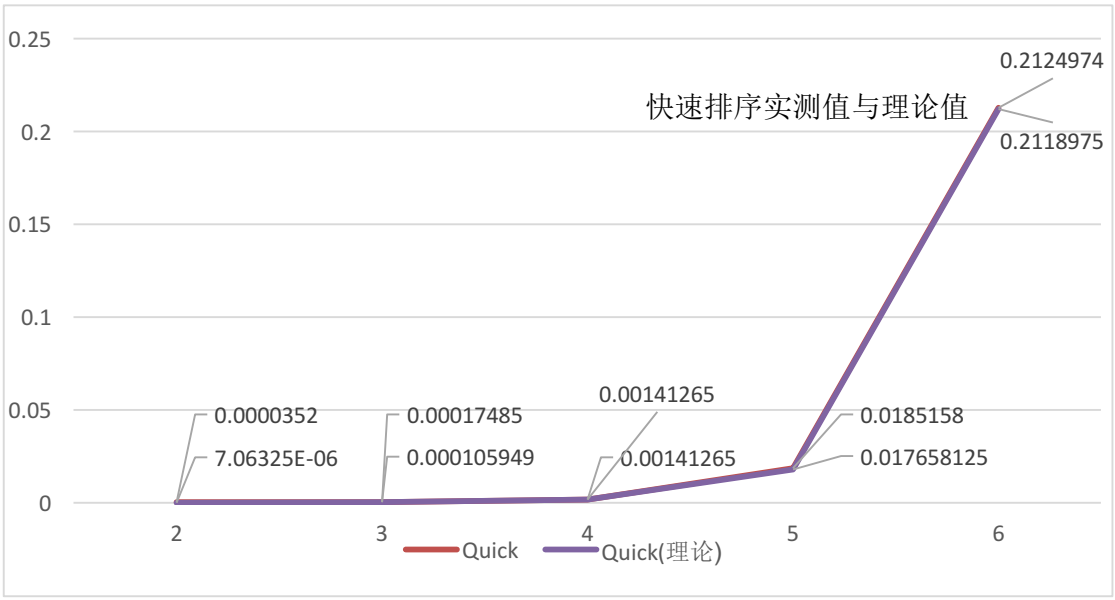

图 40　分析理论值与实测值差异（以数据量 10000 为基准）

可以看到实测效率与理论效率基本一致。

将上述五种算法的测试数据统计成一张图表，如图 40 所示。

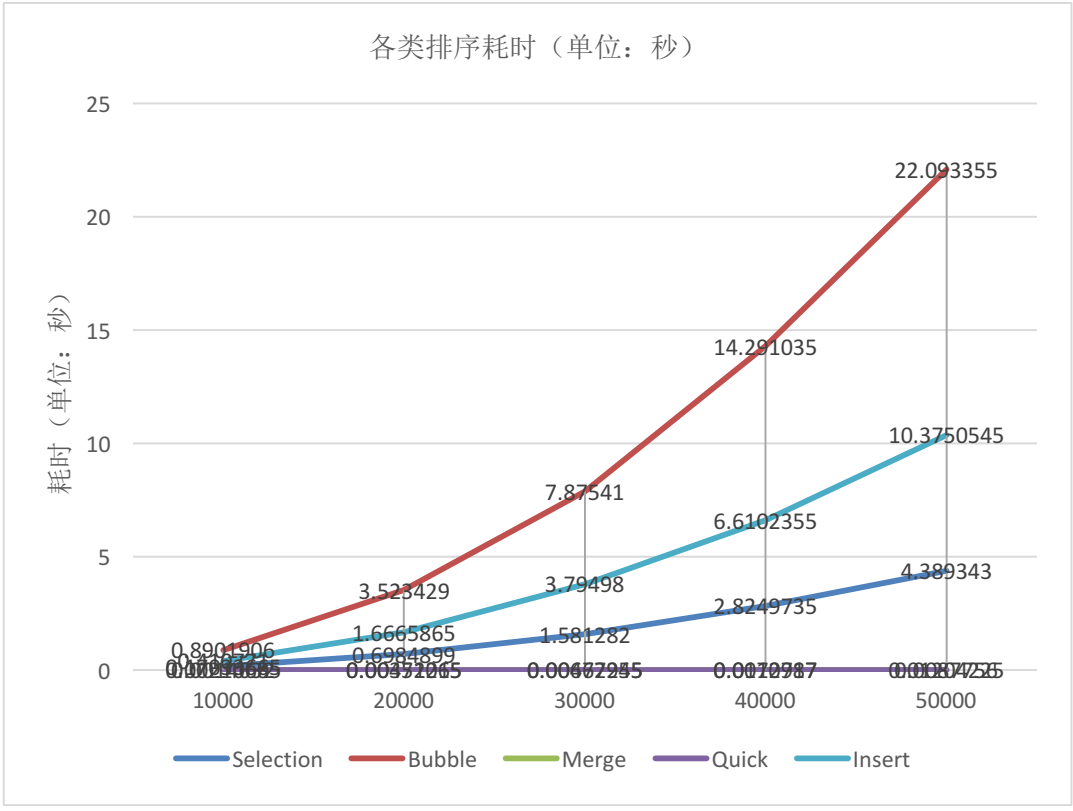

图 41　各类算法耗时

可以看出，复杂度为 O（nlogn）的快速排序和归并排序的效率大大高于复杂度为 O（n^2）的冒泡排序、插入排序和选择排序。

四．分析总结

（1） O(n^2)性能分析

平均性能为O(n^2)的直接插入排序，选择排序，冒泡排序：

在数据规模较小时，各算法效率差不多。当数据较大时，冒泡排序算法的时间代价最高。

解释：时间复杂度同样为o（n^2）的选择、冒泡和插入排序，在对于相同数据的处理上相差的比较大，其中冒泡排序平均耗时最多，其主要原因是：冒泡排序在比较次数上达到了o(n^2)，但这种排序同时也受交换次数的影响，而且最大时间复杂度也是o（n^2）。因此，虽然同样是o（n^2）的复杂度，冒泡排序的二次项系数会比另外两个大不少，最为耗时。

（2） O(nlogn)性能分析

平均性能为O(nlogn)的快速排序，归并排序：

其中，快排效率最高，但在待排序列基本有序的情况下，会变成冒泡排序，接近O(n^2)。

解释：同样是O（nlogn）但快速排序更快：快速排序出现最差的情况并不是由于输入数据，而是选取到的随机数本身，选到极端的情况非常小，所以对于绝大部分数据而言都是能达O（nlogn）的复杂度，而合并排序需要赋值的操作较多，受输入数据的影响比快排大，所以当数据规模较大时，不受输入数据影响的快速排序更快。

（3） 排序稳定性

插入排序，冒泡排序，归并排序都是稳定的

选择排序，快速排序是不稳定的。

（4） 各排序算法整体分析

冒泡排序、插入排序、快速排序对数据的有序性比较敏感，尤其是冒泡排序和插入排序。

选择排序不关心数据的初始次序，它的最坏情况的排序时间与其最佳情况差不多，其比较次数为 n(n-1)/2。因此对次序近乎正确的数据，选择排序可能比插入排序慢很多。

插入排序在最好的情况下有最少的比较次数 ，但是它在元素移动方面效率非常低下，因为它只与毗邻的元素进行比较，效率比较低。

五．实验心得

本次实验花费了我很多的时间。但加深了我对这几种排序算法的认识。

各种排序算法都有各自的优缺点，适用于不同的条件。因此在选择一种排序算法解决实际问题之前，应当先分析实际问题的类型，再结合各算法的特性，选择一种最合适的算法。

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。