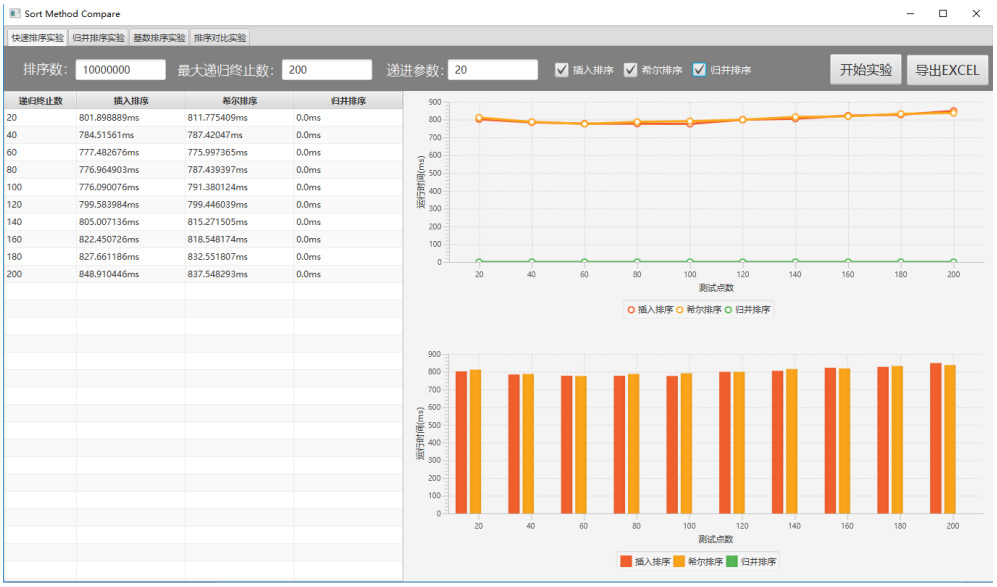
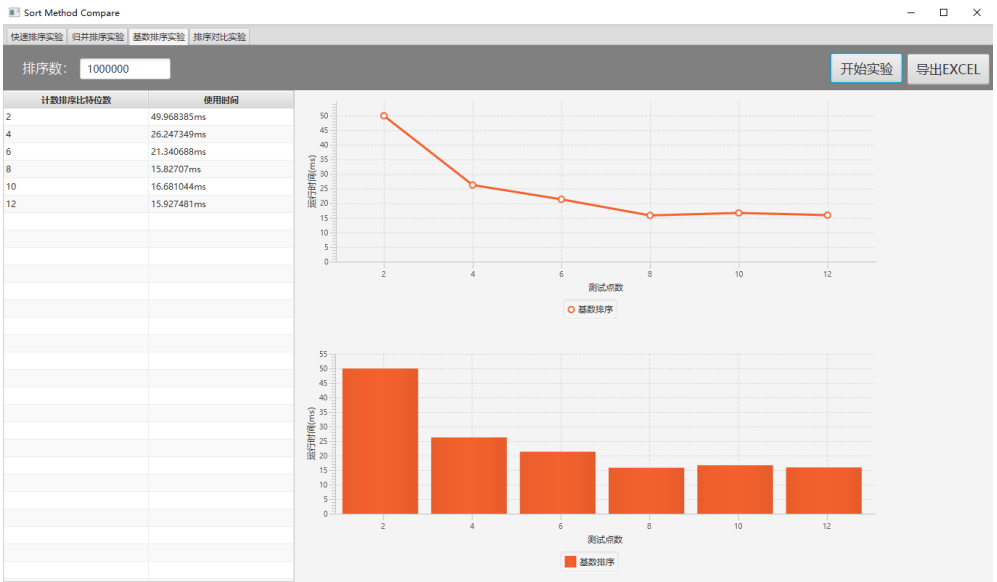
一、实验环境

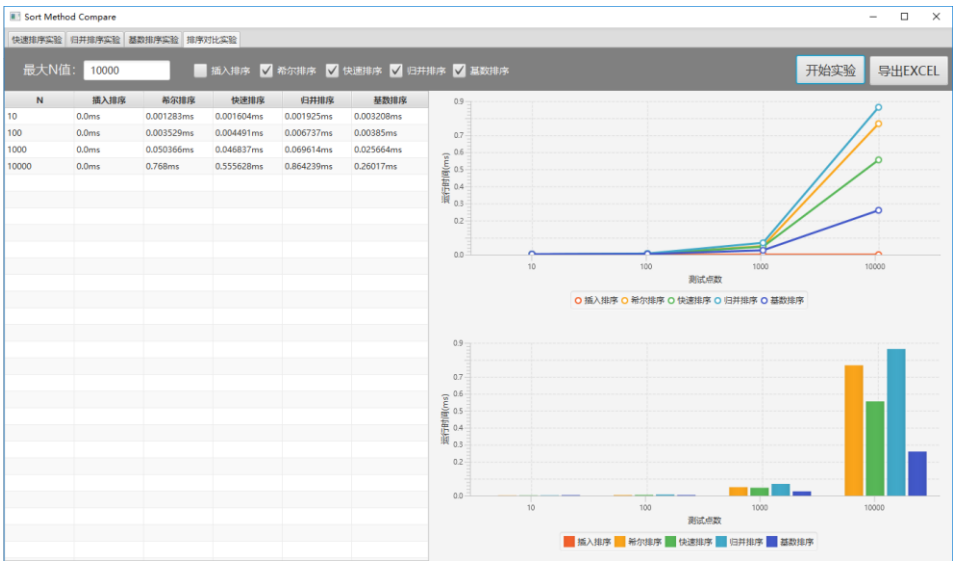
操作系统：Windows10 专业版 64 位
处理器：Intel(R) Core(TM) i7-8700 @3.2GHZ
内存：16G
编程语言： Java 1.8
编译参数 –Xmx12g

二、实验界面

本次实验为四个子实验编写了界面，分别是快速排序的终止条件，归并排序的终止条件，基数排序的排序位数和五种排序方式的比较。具体的程序界面如下图所示：







三、实验与结论

1. 生成数据

Java 不支持无符号整数，而实验要求的数据范围是$2^{32}-1$，Java 的 int 类型最大值是$2^{31}-1$，因此只能使用 long 来代替无符号整数。

Java 的 Random 使用一个 48 位的种子来构造随机数，它被一个线性同余公式修改，同一个种子生成的随机数序列是一致的。在不手动指定种子的前提下，无参的构造方法会自动产生一个种子，并使用 CAS 自旋保证每次获取的种子都不一样。

```
private static long seedUniquifier() {
    for (;;) {
        long current = seedUniquifier.get();
        long next = current * 181783497276652981L;
        if (seedUniquifier.compareAndSet(current, next))
            return next;
    }
}
```

种子确定以后使用固定的算法产生随机数：

```
protected int next(int bits) {
    long oldseed, nextseed;
    AtomicLong seed = this.seed;
    do {
        oldseed = seed.get();
        nextseed = (oldseed * multiplier + addend) & mask;
    } while (!seed.compareAndSet(oldseed, nextseed));
    return (int)(nextseed >>> (48 - bits));
}
```

本次实验使用 ThreadLocalRandom 来生成随机数，种子 SEED 变量是 ThreadLocalRandom 在 Thread 对象中的偏移，高并发获取 Random 时不需要使用 CAS 来保证每次获取的值不一致，因此提高了获取随机数的性能。

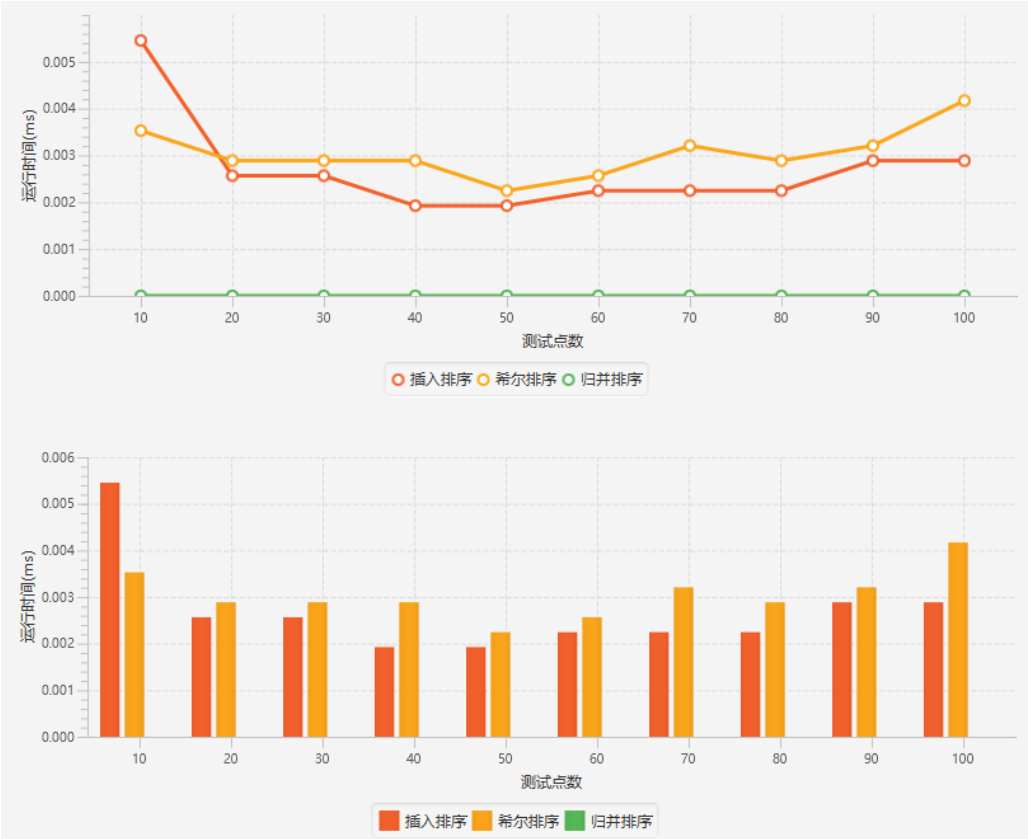
2. 快速排序的终止条件

快速排序的排序过程可以概括为划分，对左边排序，对右边排序，但划分到一定程度之后则不需要再进行划分，而是可以直接使用插入排序或希尔排序等对剩下未排序的部分进行排序。

在对 100 个数进行排序时，终止划分的数量与排序时间的对应关系如下表所示：

递归终止数	插入排序	希尔排序
10	0.005454ms	0.003529ms
20	0.002566ms	0.002887ms
30	0.002566ms	0.002888ms
40	0.001925ms	0.002887ms
50	0.001925ms	0.002245ms
60	0.002245ms	0.002567ms
70	0.002246ms	0.003208ms
80	0.002246ms	0.002887ms
90	0.002887ms	0.003208ms
100	0.002887ms	0.00417ms

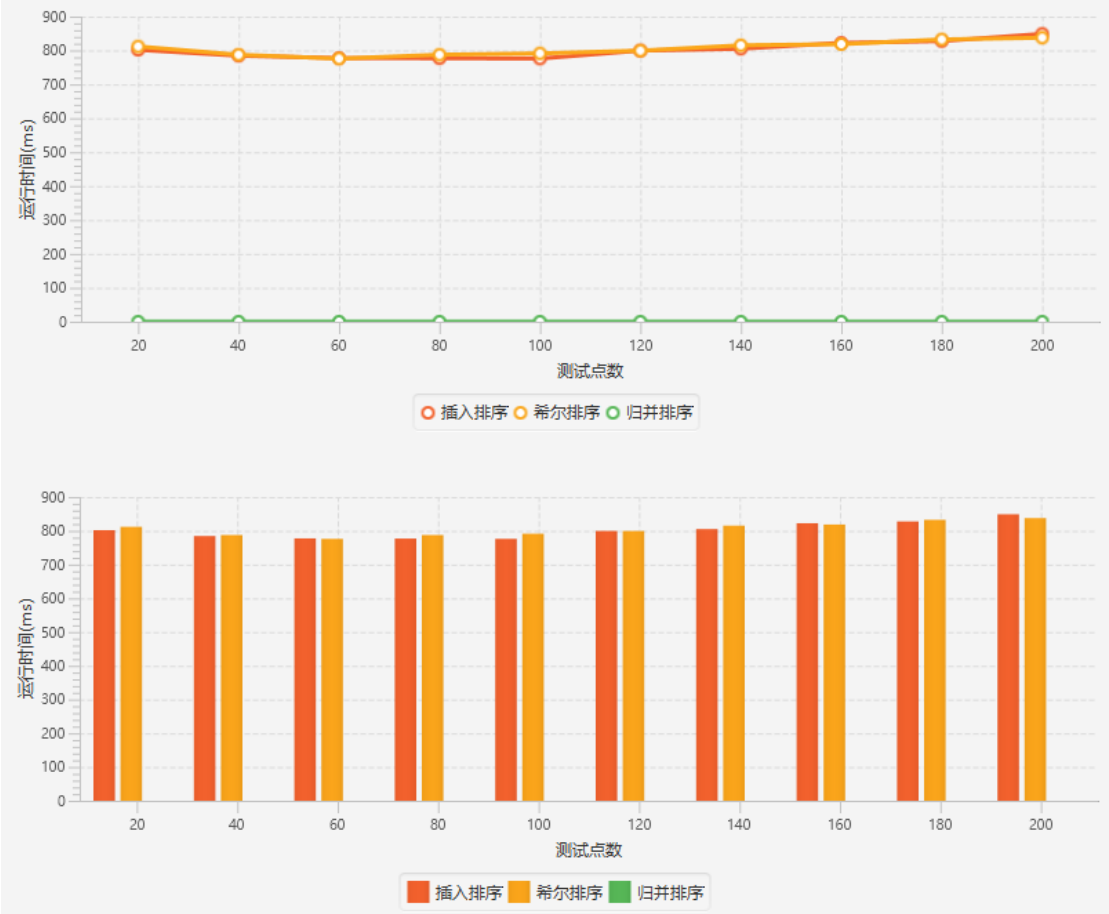
对应的图表如图所示：



由实验可以发现，在终止条件为 50 左右的时候，对算法运行效率的改进最为明显。下面对较多的数来进行排序，排序的时间与终止条件的对应关系表如下表所示：（排序的随机数为 10000000 个）

递归终止数	插入排序	希尔排序
20	801.898889ms	811.775409ms
40	784.51561ms	787.42047ms
60	777.482676ms	775.997365ms
80	776.964903ms	787.439397ms
100	776.090076ms	791.380124ms
120	799.583984ms	799.446039ms
140	805.007136ms	815.271505ms
160	822.450726ms	818.548174ms
180	827.661186ms	832.551807ms
200	848.910446ms	837.548293ms

对应的图表如下：



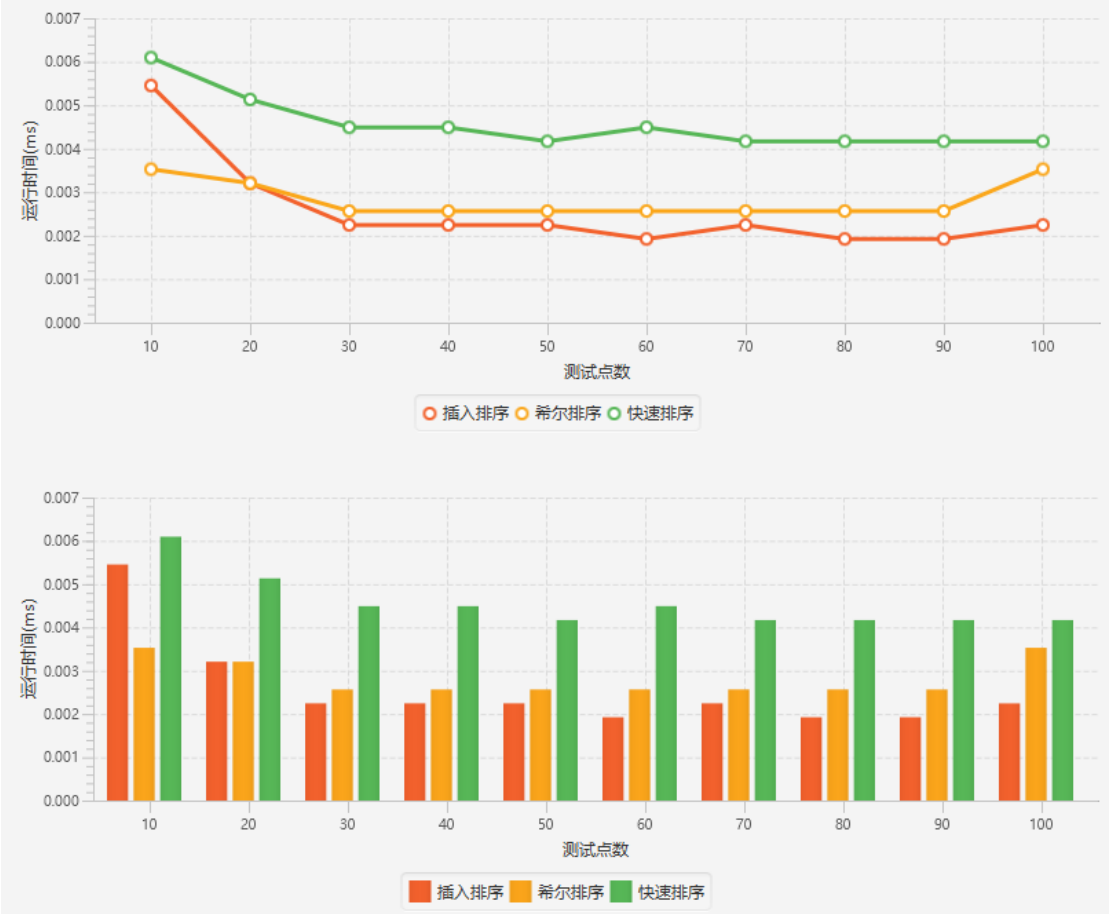
上面的实验仍然证明终止条件处于 50 左右时能提高快速排序的速度，因此，在需要排序的随机点数小于 50 时就直接使用插入排序，可以较好的提高快速排序的速度。

3. 归并排序的终止条件

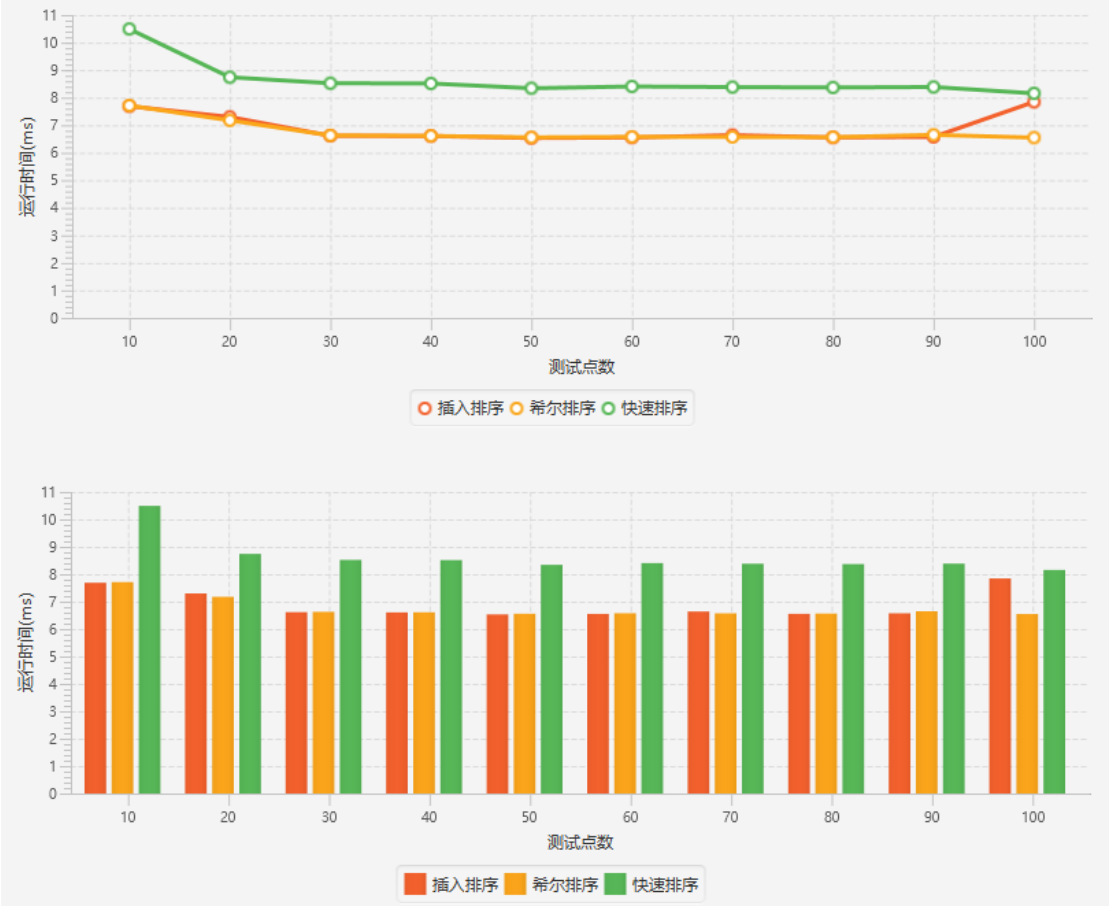
归并排序的过程也是一个分治的过程，因此也需要考虑排序的终止条件，在数量小于一

定值之后可以使用插入排序、希尔排序等进行排序，在对 100 个随机数进行排序时，终止的数量与运行时间的图表为：

递归终止数	插入排序	希尔排序	归并排序
10	0.005454ms	0.003529ms	0.006096ms
20	0.003208ms	0.003208ms	0.005133ms
30	0.002246ms	0.002566ms	0.004491ms
40	0.002246ms	0.002566ms	0.004491ms
50	0.002246ms	0.002566ms	0.00417ms
60	0.001924ms	0.002567ms	0.004492ms
70	0.002246ms	0.002567ms	0.00417ms
80	0.001924ms	0.002567ms	0.00417ms
90	0.001925ms	0.002566ms	0.00417ms
100	0.002245ms	0.003529ms	0.00417ms



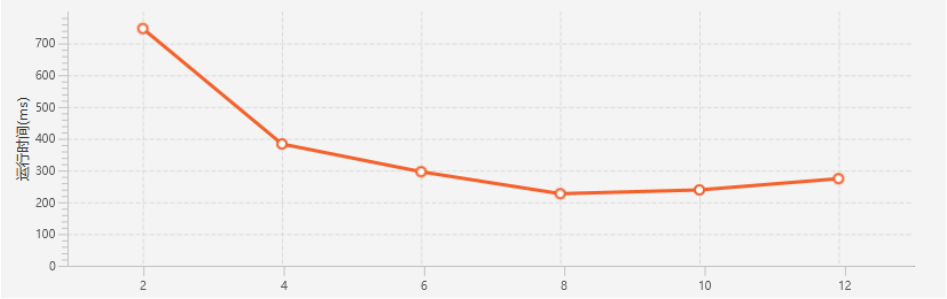
在对更多的随机数进行排序时，这样的规律也仍然保持，也就是在终止条件为 30 时，使用插入排序可以更多地减少归并排序使用的时间。



4. 基数排序的位数

基数排序是通过多次计数排序来实现的，由于分配$2^{32}-1$个数的空间并不现实，所以只能每次对 32 位无符号整数的某些位进行排序，而每次对多少位进行排序则没有明确的规定，因此本实验讨论基数排序时间与排序位数的关系。在对 10000000 个随机数进行排序时，排序时间与位数相关的图表如下所示：

计数排序比特位数	使用时间
2	746.81468ms
4	383.427258ms
6	296.067517ms
8	226.891455ms
10	238.824314ms
12	274.244664ms



改变随机点的数量并进行多次试验，试验得出的折线图仍然如上图所示，也就是每次对 8 个比特位进行排序，基数排序的算法效率最高。

从理论上来讲，比较位数越大，算法的运行速度越快，但是本实验确得出了不一样的结论，那就是每次只对 8 个比特位进行比较，算法运行速度越快。导致这一结果的原因可能是 JVM 分配空间也需要一定的时间，这些分配大空间的时间抵消了比较位数大的好处。

5. 五种算法的比较

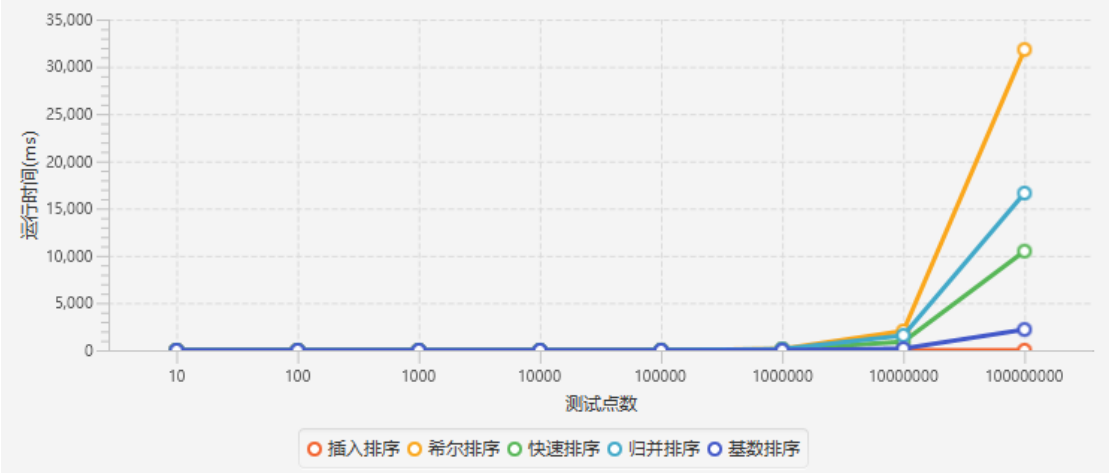
为了比较几种算法的真实效率，几种算法的参数设置如下：

- 快速排序：终止条件为 2 个数
- 归并排序：终止条件为 2 个数
- 基数排序：比较位数为 8 位

几种算法的运行时间如下表所示：

N	插入排序	希尔排序	快速排序	归并排序	基数排序
10	0.001604ms	6.41E-4ms	0.001925ms	0.002246ms	0.009945ms
10^2	0.002887ms	0.00417ms	0.006416ms	0.009303ms	0.01219ms
10^3	0.133454ms	0.065765ms	0.065443ms	0.094316ms	0.097203ms
10^4	10.228119ms	0.719558ms	0.568139ms	0.837292ms	0.123188ms
10^5	957.395937ms	10.545712ms	6.787842ms	10.128029ms	1.382655ms
10^6	104521.871775ms	145.655817ms	78.511711ms	127.33708ms	15.646138ms
10^7		2098.058155ms	911.01187ms	1553.163409ms	262.40452ms
10^8		32331.480169ms	10265.401125ms	16966.168258ms	2798.6456ms
$2*10^8$		75951.824254ms	21333.672207ms	33264.381297ms	5941.10274ms
10^9			115242.662384ms		

不考虑插入排序，在 10^8 数量级的情况下运行时间折线图如下图所示：



结论：在 10^2 数量级，插入排序更有优势，在 10^2~10^3 数量级，希尔排序更有优势，在 10^3 数量级快速排序更有优势，10^3 数量级以上基数排序更有优势，快速排序其次，归并排序在 10^6 数量级速度超过希尔排序位于第三名。