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：排序算法	**学号**：10225501403	**上机实践日期**：2023.3.27
上机实践编号：No.4	**组号**：1-403	

一、目的

1．熟悉算法设计的基本思想

2．掌握计数排序、基数排序的基本思想，并且能够分析算法性能

二、内容与设计思想

1. 随机生成 1～M 范围内的 N 个整数，输入参数包括 N，M，T；可随机生成一个大小为 N、数值范围在[1, M]之间，类型为 T 的数据集合；T 包括三种类型（顺序递增、顺序递减、随机取值）；

2. 编程实现计数排序；

3. 对不同类型（顺序递增、顺序递减、随机取值）的数据集合而言，在相同 M(100000，1000000)的条件下，N 分别等于 0.1M，0.2M，0.5M，1M 时的运行时间；

4. 对不同类型（顺序递增、顺序递减、随机取值）的数据集合而言，在相同 N(100000，1000000)的条件下，M 分别等于 2N，5N，10N，20N 的运行时间；

5. 编程实现基数排序

6. 对不同类型（顺序递增、顺序递减、随机取值）的数据集合而言，在不同数据规模情况下（数据规模 N=10000, 100000, 1000000），M 分别等于 N，5N，10N，20N，算法的运行时间各是多少？

思考题：将快速排序与计数排序进行对比（自由发挥）。

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

1. 写出计数排序、基数排序算法；

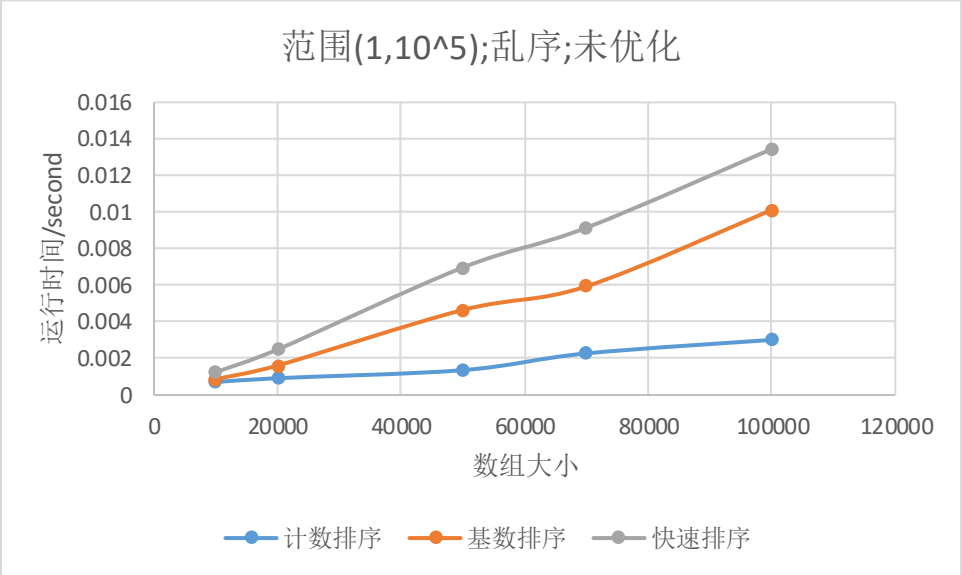
2. 分别画出各个实验结果的折线图

五、总结

实验平台：windows VS code

首先，我实现了最基础的计数排序与基数排序。即计数排序接收的参数有数组指针、数组大小、数据最大值（数据最小值固定为 1），基数排序以 10 为基数，接收的参数有数组指针、数组大小。同时，为了比较非比较排序和比较排序算法在时间上的差异，我在实验中加入了快速排序（来自实验三优化后的版本）。然后分别进行了 3、4 标号的实验。

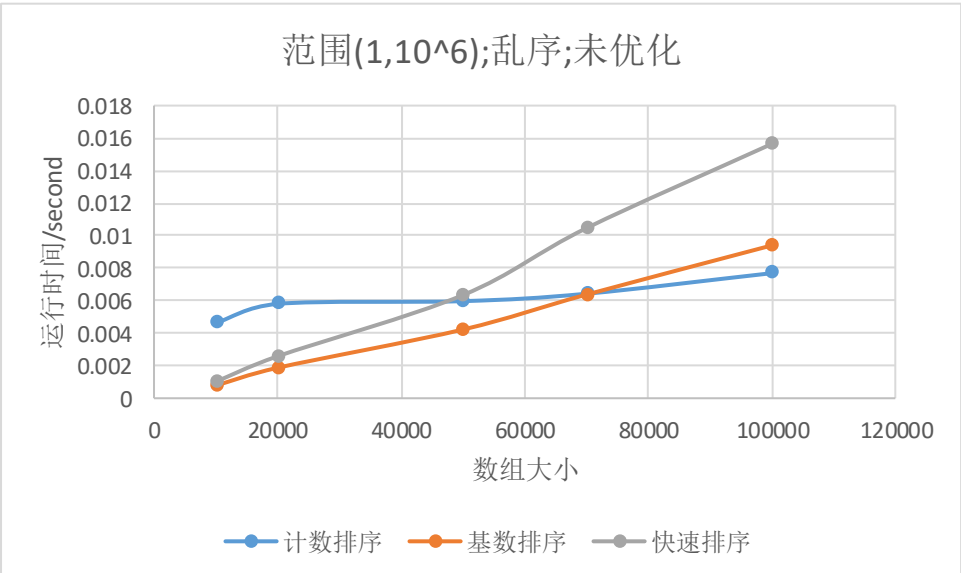
首先是对未优化的算法进行了测试，测试的数组规模分别固定为10^5,10^6，数组大小分别为 10^4,2*10^5,5*10^5,7*10^5,10^6。得到以下数据，绘制为散点图。



	计数排序	基数排序	快速排序
10000	0.000711	0.000867	0.001249
20000	0.000921	0.001612	0.002489
50000	0.001360	0.004649	0.006970
70000	0.002276	0.005940	0.009145
100000	0.003013	0.010120	0.013451

由图中结果可见，由于计数排序与基数排序得益于无序比较，其二者时间复杂度均为 O(n)，使得二者相对于快速排序有着相当大的运算优势，差距也随着数组大小的增大越来越大。

接下来再进行 10^6 数据范围的一组，猜测来说，快速排序的速度应该会有所提升，因为重复的数据会有减少。



	计数排序	基数排序	快速排序
10000	0.004626	0.000780	0.001016
20000	0.005803	0.001873	0.002581
50000	0.005971	0.004224	0.006343
70000	0.006428	0.006373	0.010461
100000	0.007692	0.009398	0.015640

由结果可见，快速排序由于单步的运算很快，但是由于其时间复杂度为O(nlgn)，最终在50000规模时慢于基数排序与计数排序。

但是实验进行到这里也反应出一个问题，就是不公平。

计数排序接收的参数有一个数据的最大值，而且默认知道数据的最小值为1，这对于另外两种算法实际上是不公平的，同时基数排序有很大的优化空间，因此我开始对于计数排序与基数排序进行修改。

首先对于计数排序的修改，其实很简单，就是在让排序函数内部添加寻找最大值最小值的操作，这样可以尽可能节省空间。

而对于基数排序的优化，一共有 点。

第一点就是对于每个数据取key的操作。

这一步操作在整个运算过程中大量运用，且包含一步除法和一步取余。我们假定基数排序的基数为base，那么对于nums[i]/base%10这一步的base可以提前运算好储存在数组中，即int depart[10],depart[0]=1,depart[j]=depart[j-1]*10。但是这样只能简化了乘法，并不能简化除法，数组其实可以修改为depart[9]=1/10^9,depart[j]=depart[j+1]*10。这样对于取key就可以修改为nums[i]*depart[j]%10。

第二步就是基数的改变。

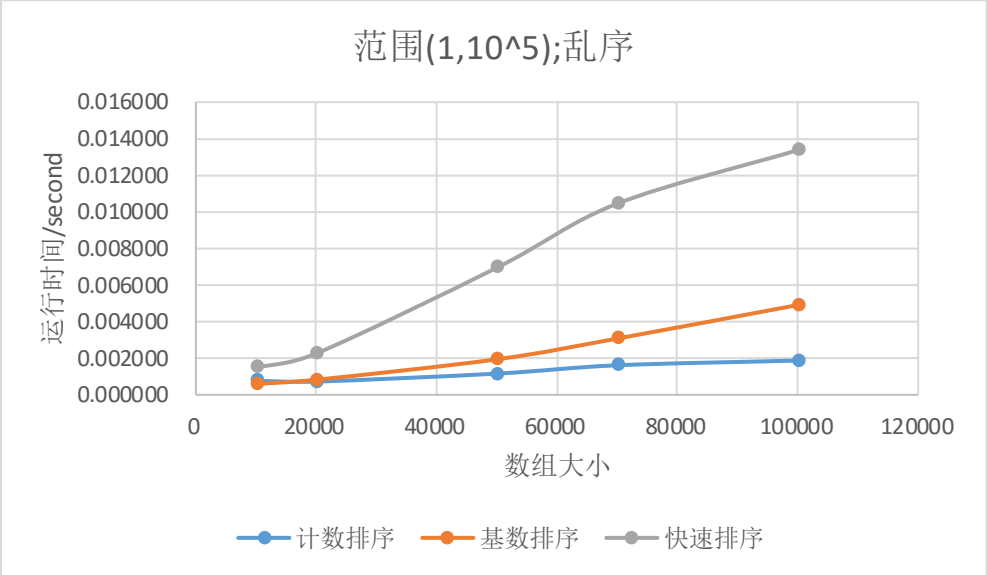
对于一组数据，我们需要循环的次数其实为max/base，而每一步操作其实就是对0~base-1的数据进行桶排序，因此base对于运算有着非常大的影响。再者，我们知道对于计算机而言，数据的运算其实都是基于二进制的，如果我们最开始就以2的倍数作为基数，然后通过二进制的操作来进行除法取余操作，则可以节约非常多的时间。

优化思路有了，下满阐述细节。

首先我们选取base为2^10，即1024，因为对于一个int类型而言数据的范围为-2^31~2^31-1,而2^10三次遍历就可以达到了一个非常大的范围，完全可以包括实验的数据范围，节省了遍历次数。然后我们将depart数组改为{0,10,20,30}，然后再来修改取key。对于取key，除法操作可以换位二进制的右移，右移的位数分别取depart数组里面的元素。对于取余操作，由于我们的基数为2的倍数，所以可以修改为与base-1按位与，最后将取key写为函数，即最终修改为num>>depart[i]&(base-1)。其余的基本步骤不变，接下来重新再进行实验。

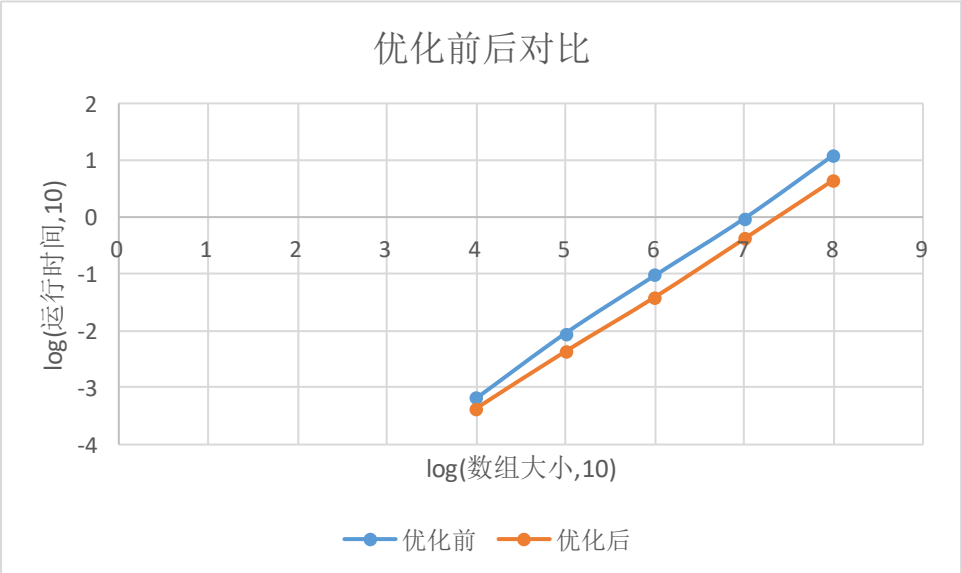
注意：为了保证实验的公平性，即任何一种算法均不能提前知道数据具体范围，而对于基数排序，我采用的方案为首先遍历一遍找到最大值，然后再根据最大值取位移次数t分别为1,2,3,4。

首先是测试的数组规模分别固定为10^5,10^6，数组大小分别为10^4,2*10^4,5*10^4,7*10^4,10^5；10^5,2*10^5,5*10^5,7*10^5,10^6。数据为乱序。得到以下数据，绘制为散点图。



	计数排序	基数排序	快速排序
10000	0.000767	0.000594	0.001491
20000	0.000723	0.000824	0.002260
50000	0.001158	0.001940	0.006948
70000	0.001613	0.003074	0.010450
100000	0.001873	0.004908	0.013374

由此可见，在经过优化与修改后，基数排序的速度得到了很大的提升，但是为了更明显的比较，我将修改前后函数进行了运算比较，数组大小为10^4,10^5,10^6,10^7,10^8，数据范围为 1 到对应的数组大小，数据类型为乱序，得到以下结果。为了便于比较，我对运行时间与数组大小均取了以 10 为底的对数，其中运行时间单位为秒。

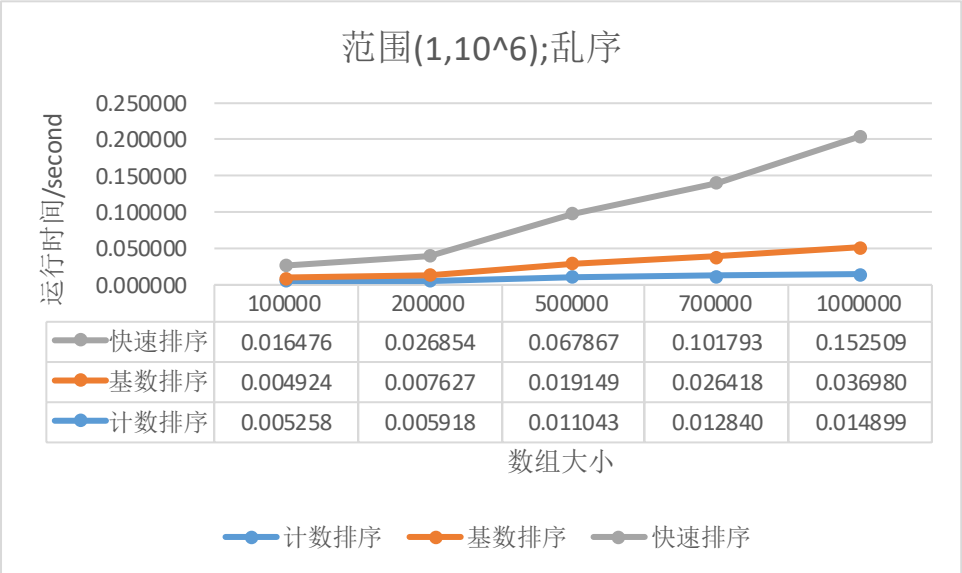


	优化前	优化后
10000	0.000651	0.000434
100000	0.009087	0.004365
1000000	0.094335	0.039223
10000000	0.944215	0.415625
100000000	12.372214	4.497508

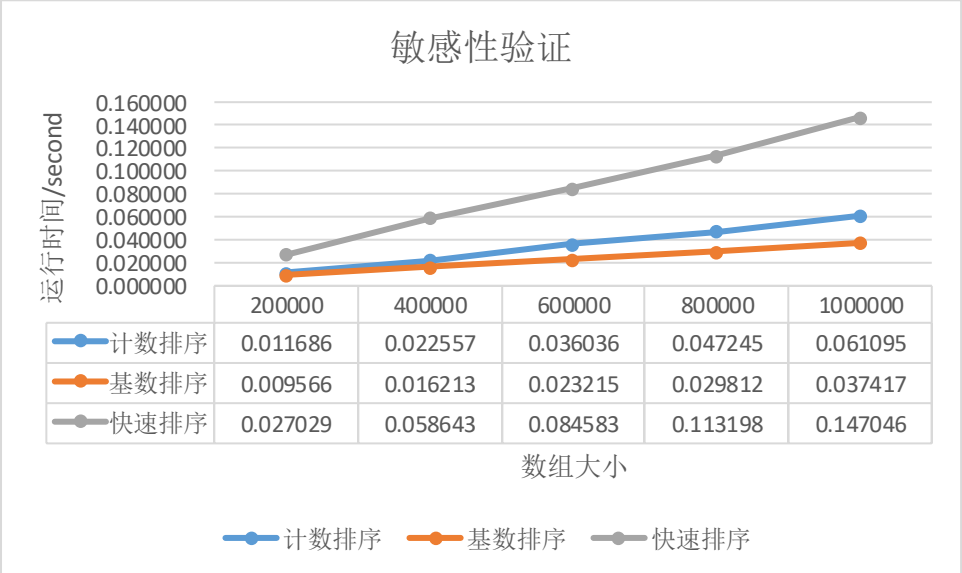
这是未取对数的数据。

我们可以很明显的观察到，优化对于基数排序的提升。

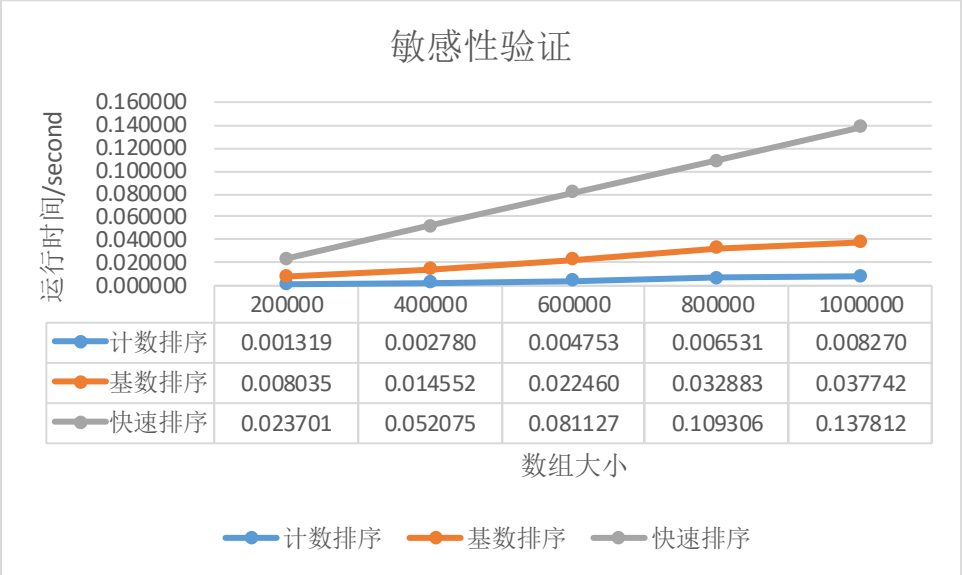
接下来再进行三种算法在固定数据范围为 1~10^6 时，数据为乱序，数组大小分别为 10^4,2*10^4,5*10^4,7*10^4,10^5 时的运算时间检测，得到以下数据以及图表。



但是到这里我产生了一个疑问，既然基数排序只有在第一组数据比计数排序快，那么是不是可以猜测基数排序对于计数排序对数据的重复性更敏感？我进行了如下实验，即数组大小依次为 200000,400000,600000,800000,1000000，同时数据范围为 1 到对应的数组大小的十倍，乱序数组，得到一下结果。

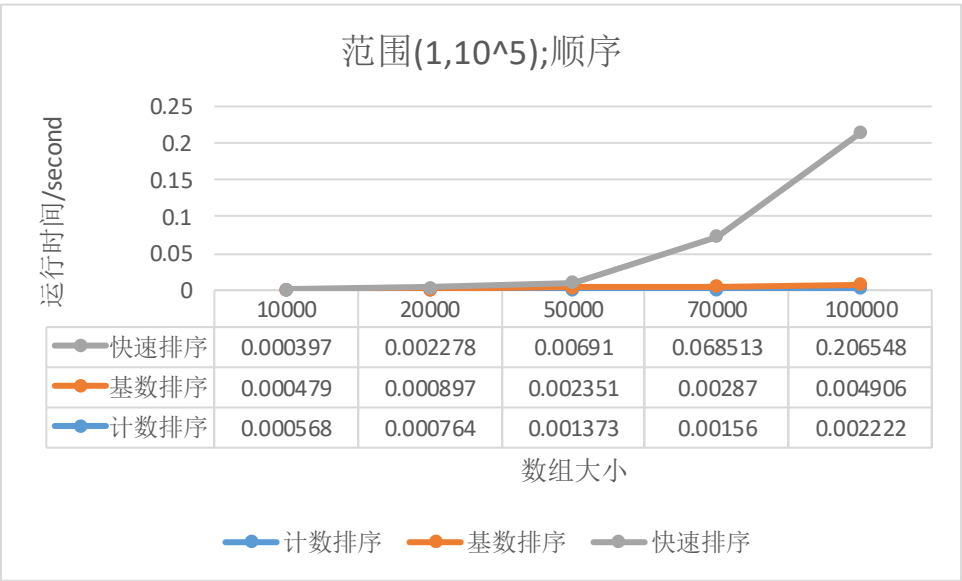


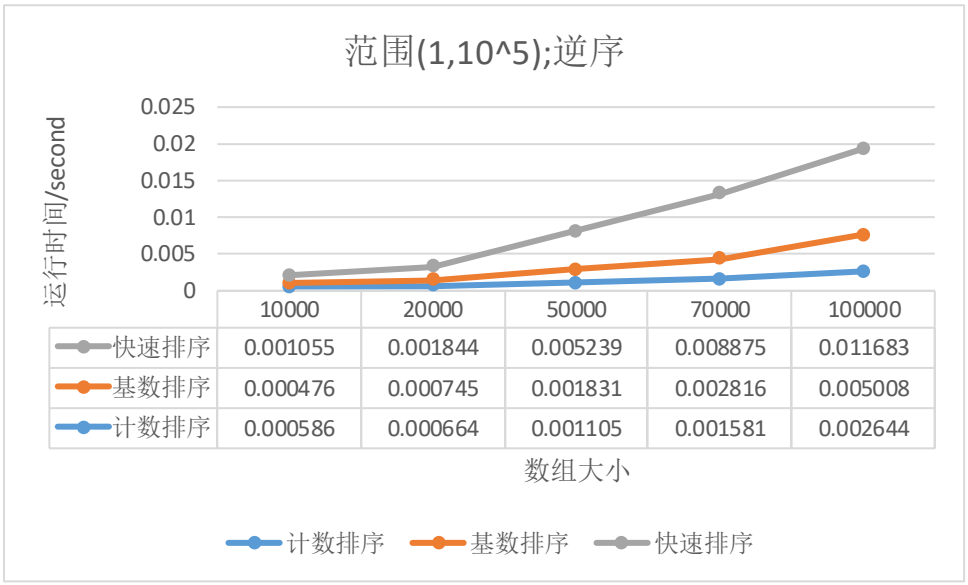
同时，我将对应组的数据规模都减小 100 倍，即数据范围为数组大小的 1/10，得到以下结果。



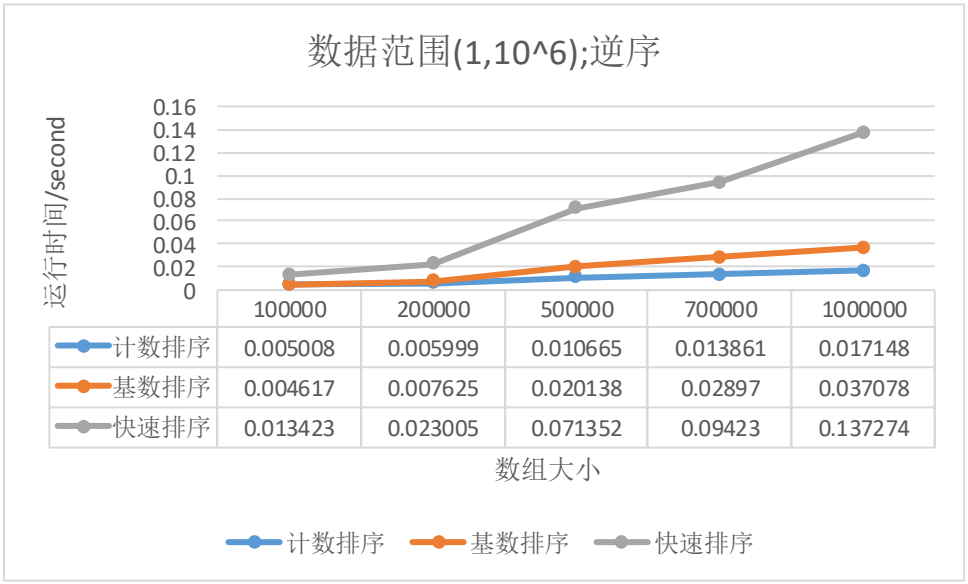
结论显而易见，基数排序对于数据的重复有着比较强的敏感性。

接下来用顺序和逆序数据来进行 3 问题中的实验，得到以下结果

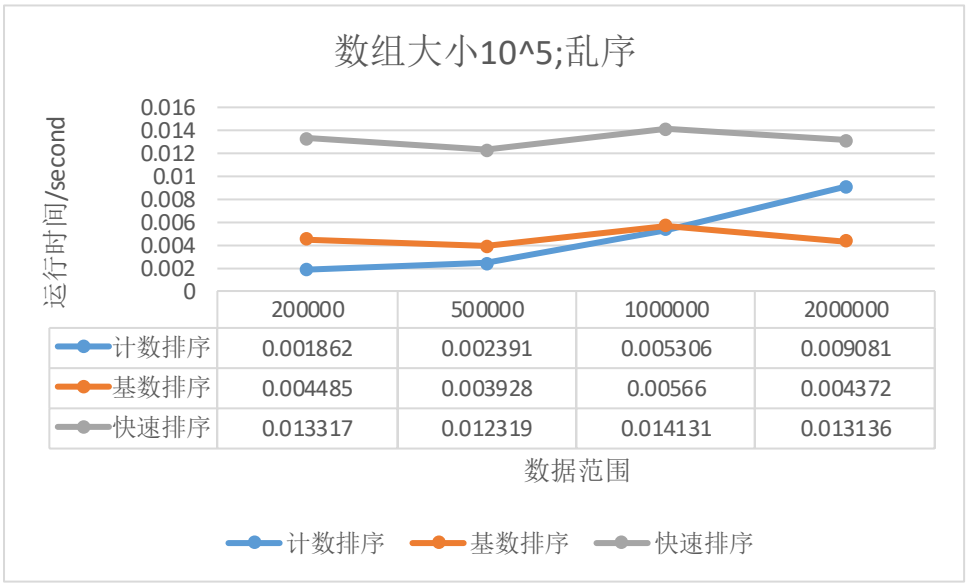




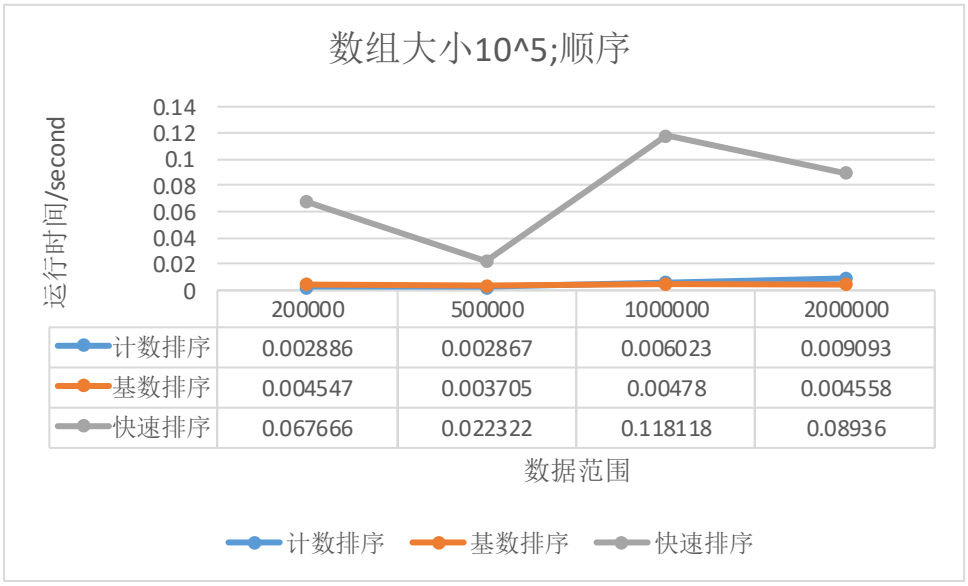
对于顺序的快排很慢，个人猜测是 Windows 随机数生成的问题，同样的代码在水杉上运行速度远快于 Windows。

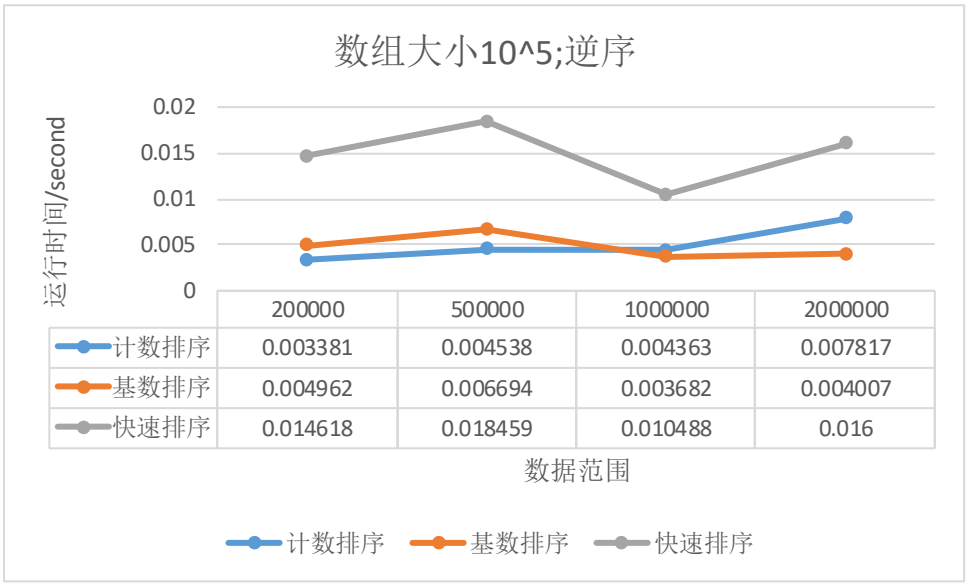


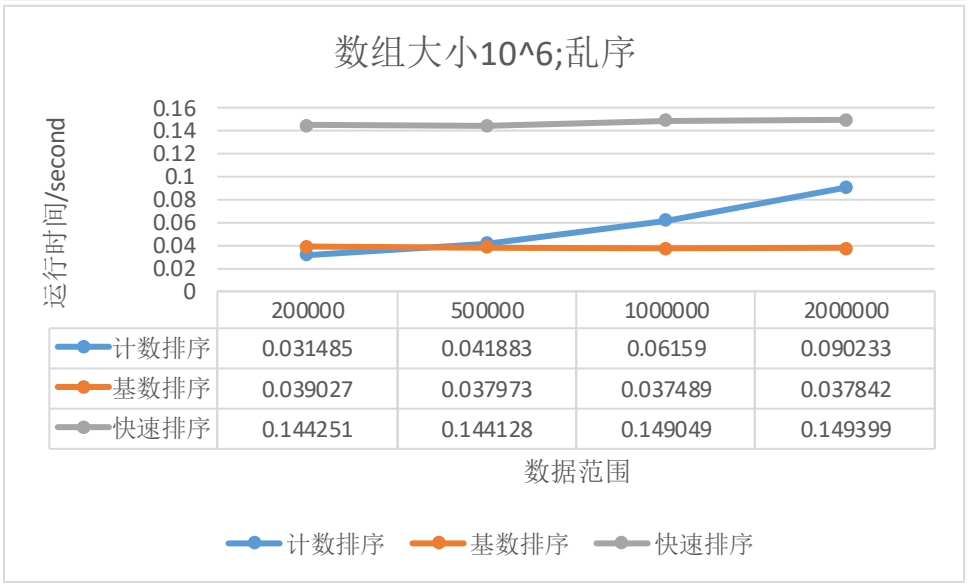
至此，3 问题已经全部完成，下面进行 4,5 问题。
首先是 4 问题，这个问题将数组大小改为定值

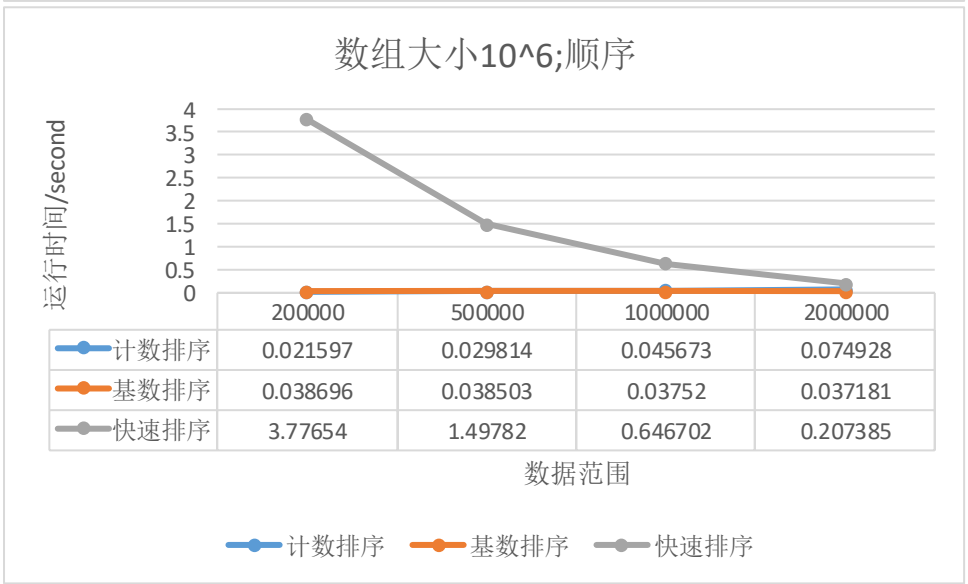


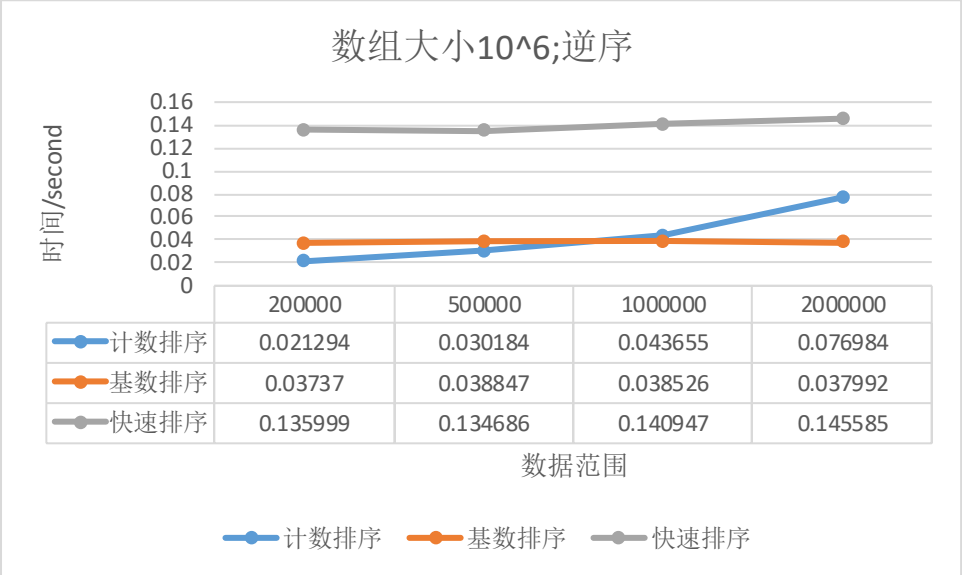
可以看到，快速排序变化不大，计数排序由于其原理稳定上升，基数排序由于我代码实现的原因，在时间增加上不明显，需要达到一定阈值（2^10,2^20,2^30）才会明显。




数组大小10^5;逆序
运行时间/second
0.02
0.015
0.01
0.005
0
200000 500000 1000000 2000000
计数排序 0.003381 0.004538 0.004363 0.007817
基数排序 0.004962 0.006694 0.003682 0.004007
快速排序 0.014618 0.018459 0.010488 0.016
数据范围
计数排序 基数排序 快速排序


数组大小10^6;乱序
运行时间/second
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
200000 500000 1000000 2000000
计数排序 0.031485 0.041883 0.06159 0.090233
基数排序 0.039027 0.037973 0.037489 0.037842
快速排序 0.144251 0.144128 0.149049 0.149399
数据范围
计数排序 基数排序 快速排序


数组大小10^6;顺序
运行时间/second
4
3.5
3
2.5
2
1.5
1
0.5
0
200000 500000 1000000 2000000
计数排序 0.021597 0.029814 0.045673 0.074928
基数排序 0.038696 0.038503 0.03752 0.037181
快速排序 3.77654 1.49782 0.646702 0.207385
数据范围
计数排序 基数排序 快速排序



由上图的实验可以发现一件很明显的事情，即计数排序对于数据范围的跨度极为敏感，而基数排序经过优化后对于数据的敏感性极低。这主要由于计数排序的原理，跨度越大，需要申请的空间地址越大，越耗费时间，而基数排序无论数据范围跨度大小，桶排序的数组都是固定大小，且不用主动申请地址，节约时间。

总结：

计数排序与基数排序都是十分优秀的排序方式，得益于其线性的时间复杂度，其时间开销相较于快速排序这种传统的通过比较来排序的算法有着鲜明的优势。但是为什么 c 语言内置了快排而不是基数排序或者计数排序呢？我认为可能有两点原因。第一点是内存的开销。计数排序与基数排序都是牺牲了空间去换取时间，在数据量非常大的时候会很不稳定。第二点我认为是 c 语言内置的快速排序的回调函数使其适用范围很广，通过自定义的回调函数可以对很多东西进行比较。

这次对于基数排序的优化我认为也比较成功，大大缩短了基数排序所需要的时间。但是这次实验的过程中也存在不足，即写代码时有一些小细节遗漏，导致超出了数组范围，在纠错上浪费了时间。