北京郵電大學

Beijing University of Posts and Telecommunications

《算法分析—第二章实验》

实验报告

姓　　名　袁洁　胡敏臻

学　　号　2019211426　2019211424

班　　级　2019211307　2019211307

专　　业　计算机科学与技术

1 实验内容

1.1 采用合并排序与快速排序，取 N=30,000，生成长度为 n、最大值 x(n-1)≤N 的 L 组序列{SEQ(n,N)}，每组序列{SEQ(n,N)}中有 M 个长度均为 n、组成序列的数字相同，但数字间顺序不同的序列。要求：

- L=6 种不同长度，n=2,000，5,000，10,000，15,000，20,000，30,000
- M=5，即每组有 5 个长度相同的序列 SEQ(n, N)
- 对同组内长度相同的序列，ADD、DD 值不相同

1.2 采用线性时间选择算法，根据基站 k-dist 距离，挑选出

- k-dist 值最小的基站
- k-dist 第 5 小的基站
- k-dist 值第 50 小的基站
- k-dist 值最大的基站

要求

- 在排序主程序中设置全局变量，记录选择划分过程的递归层次
- 参照讲义 PPT，将教科书上的“一分为二”的子问题划分方法，改进为“一分为三”，比较这 2 种划分方式下，选择过程递归层次的差异。

1.3 采用平面最近点对算法，根据基站经纬度，挑选出

- 距离非零、且最近的 2 个基站
- 距离非零、且次最近的 2 个基站

要求：返回

- 最近/次最近的 2 个基站间距离
- 最近/次最近的 2 个基站点对（用基站 ENodeBID 表示）

2 排序算法

2.1 四种算法说明

本次实现了 4 种算法，递归合并排序算法，非递归合并排序算法，快速排序 1，快速排序 2。并且参照 PPT 讲义内容，当输入数组 a[p:r]已经按照非递减序排列好时，直接返回 a[p:r]，作为排序结果；当 a[p:r]已经按照非递增序排列好时，返回 a[p:r]中元素的逆序，作为排序结果。并且我们同时在排序主程序中设置全局变量，记录排序过程的**递归层次**。这四种算法都采取了以上这两个策略。

- 合并排序 1：自上而下分解数组，直至子序列长度为 1，再自下而上采用 merge 合并已经排好序的子序列
- 合并排序 2：省略了自下而上的分解过程，将数组中相邻元素两两配对，作为最底层的子问题，再由下而上使用 merge 进行排序
- 快速排序 1：快速排序 1 固定最左边的数为划分的基准元素
- 快速排序 2：因为考虑到划分后的子序列是否平衡、对称，快速排序 2 在 a[p:r]中随机一个数作为划分的基准元素

2.2 测试结果

2.2.1 对序列 SEQ(n, N)从小到大进行排序，要求

- 用表格记录每个序列的长度、ADD、DD、排序时间
- 对递归算法，观察统计递归层次。

测试结果如下表说明：

序列SEQ编号	长度n	组号	DD	ADD	递归合并层次/时间	非递归合并时间	快排1递归层次/时间	快排2递归层次/时间
1	2000	1	964619	482.31	10/0.001058	0.000532	21/0.000228	22/0.00038
2	2000	1	971007	485.503	10/0.000962	0.000347	21/0.000232	22/0.000387
3	2000	1	984371	492.185	10/0.001156	0.000339	21/0.000234	22/0.000381
4	2000	1	996800	498.4	10/ 0.00125	0.000439	21/0.000336	22/0.000499
5	2000	1	1.03042e+06	515.211	10/0.001049	0.000382	22/0.000428	22/0.000411
6	5000	2	6.13262e+06	1226.52	12/0.002762	0.001028	26/0.000677	27/0.001022
7	5000	2	6.19416e+06	1238.83	12/0.002863	0.001007	26/0.000675	25/0.001012
8	5000	2	6.29067e+06	1258.13	12/0.002918	0.001301	23/0.00099	25/0.001164
9	5000	2	6.32757e+06	1265.51	12/0.002885	0.00102	24/0.000732	25/0.001073
10	5000	2	6.38757e+06	1277.51	12/0.00294	0.001091	24/0.000911	28/0.001197
11	10000	3	2.47829e+07	2478.29	13/0.005897	0.001988	28/0.001362	28/0.002059
12	10000	3	2.48784e+07	2487.84	13/0.006096	0.002191	26/0.00143	26/0.002263
13	10000	3	2.49507e+07	2495.07	13/0.006214	0.002255	26/0.001327	28/0.002113
14	10000	3	2.51447e+07	2514.47	13/0.006476	0.002258	31/0.001588	26/0.002431
15	10000	3	2.51829e+07	2518.29	13/0.005814	0.002062	28/0.001417	35/0.00245
16	15000	4	5.58874e+07	3725.83	13/0.009317	0.003153	26/0.002168	29/0.003218
17	15000	4	5.59386e+07	3729.24	13/0.009833	0.003207	33/0.002187	32/0.003282
18	15000	4	5.59866e+07	3732.44	13/0.009848	0.003641	31/0.002323	29/0.003418
19	15000	4	5.63803e+07	3758.68	13/0.009426	0.003284	31/0.00213	31/0.003189
20	15000	4	5.67095e+07	3780.63	13/0.009455	0.003775	31/0.002647	29/0.003782
21	20000	5	9.96514e+07	4982.57	14/0.013375	0.004489	33/0.003006	32/0.004488
22	20000	5	9.9923e+07	4996.15	14/0.012513	0.004486	28/0.003476	30/0.004326
23	20000	5	1.00037e+08	5001.85	14/0.013891	0.004598	30/0.002964	31/0.004361
24	20000	5	1.00687e+08	5034.34	14/0.012592	0.00466	33/0.002942	32/0.004705
25	20000	5	1.00997e+08	5049.84	14/0.013229	0.004728	32/0.003025	36/0.004304
26	30000	6	2.22915e+08	7430.5	14/0.019105	0.00677	31/0.004652	32/0.006562
27	30000	6	2.23742e+08	7458.08	14/0.019693	0.006908	32/0.004876	32/0.00672
28	30000	6	2.24568e+08	7485.59	14/0.018343	0.007391	31/0.004508	30/0.00659
29	30000	6	2.24801e+08	7493.37	14/0.01939	0.007836	32/0.004594	33/0.006489
30	30000	6	2.26477e+08	7549.25	14/0.019561	0.006783	32/0.004468	30/0.006358

2.2.2 针对 L=6 种不同序列长度，从表中选 6 行，考察对同一输入 SEQ(n,N)，考察问题规模 n、DD 相同时，四种排序算法运行时间 T(n,I)差异。

我们选中第 5、10、15、20、25、30 行进行分析。根据分析表格我们可以发现，在同一行中，在规模 n、DD 相同时，递归合并排序的时间最长，非递归合并排序时间与快排 2 所用时间差不多，快排 1 的使用时间最短。然而在第五次实验中，快排 2 的时间比快排 1 的时间更短。

我们的分析是这样的。递归合并排序与非递归合并排序相比，递归所调用的时间更多，会存在冗余计算，而非递归合并排序因为不用递归调用，因此时间较短。快排 2 和快排 1 的区别是快排 2 中，所选取比较的数是随机选取的，而快排 1 指定在最左边的数。因为数本身是随机生成的，因此对于这两种选择结果相差并不大，而每次的随机选取需要消耗一部分时间，因此快排 2 的时间反而会更高。但是在第五次实验中，可能随机数选取有较好的表现，因此快排 2 的时间会更短。

2.2.3 观察在长度 n 相同的同一组内，同一算法的运行时间随 ADD、DD 增长的变化情况。

在 6 组实验中，从第 1-5，6-10，11-15，16-20，21-25，26-30 组实验中进行组内分析，我们发现对同一算法而言，并没有随着 ADD 或者 DD 的变化时间有递增或者递减的趋势，时间呈现的是一种无序性。

我们思考分析后考虑，这些算法可能对 ADD 或者 DD 的要求比较小，并且我们的序列是随机生成并打乱的，他们的 ADD 或者 DD 值都在一个正态分布的峰值区域，虽然其范围很大，但是我们随机到的数列 ADD 或者 ADD 相差比较小，因此对于时间变化的不怎么明显，并且每组只有 5 次实验，次数较少，随机性比较大，较难看出规律。

2.2.4 考察问题规模 n 对算法运行时间的影响.

针对 L=6 种不同序列长度 n=2000, 5000, 10000, 15000, 20000, 30000，

- 统计 6 组相同长度的输入序列的平均 avgDD、avgADD，
- 计算四种算法对每组中的 M 个序列进行排序的平均时间 avgT(n)
- 观察同一算法的 avgT(n)随 n 的变化情况

测试结果如下表说明：

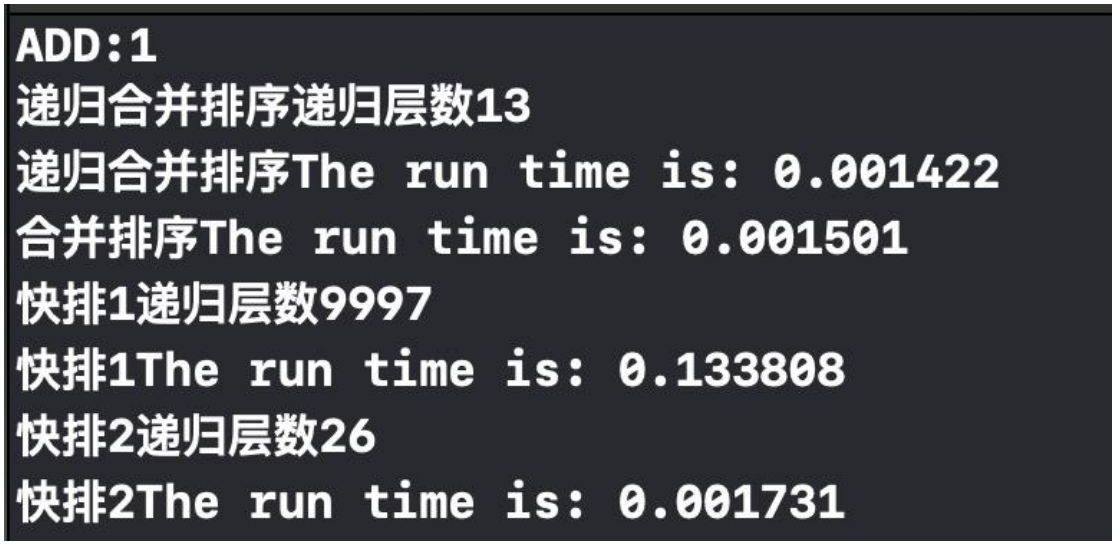
序列 SEQ 编号	长度 n	组号	DD	ADD	递归合并时间	非递归合并时间	快排 1 时间	快排 2 时间
1	2000	1	989443.6	494.7218	0.001095	0.0004078	0.0002916	0.0004116
2	5000	2	6.27E+06	1253.3	0.0028736	0.0010894	0.000797	0.0010936
3	10000	3	2.50E+07	2498.792	0.0060994	0.0021508	0.0014248	0.0022632
4	15000	4	5.62E+07	3745.364	0.0095758	0.003412	0.002291	0.0033778
5	20000	5	1.00E+08	5012.95	0.01312	0.0045922	0.0030826	0.0044368
6	30000	6	2.25E+08	7483.358	0.0192184	0.0071376	0.0046196	0.0065438

观察同一个算法，我们发现，**avgT(n)随 n 的增大不断增大。**

2.2.5 考察对排序难度相同/相近、问题规模 n 不同的序列，n 对算法运行时间的影响。

从 L=6 个组中，每组分别挑选 1 个序列，共挑选出 6 个长度 n 不同的序列 SEQ(n,N)，要求：这 6 个序列 SEQ(n,N)的 ADD 尽可能相同或接近；观察对同一算法，算法运行时间 T(n,I)随 n 的变化情况。

以 n=10000 为例，运行结果如下：



将测试结果汇聚成报告如下：

序列 SEQ 编号	长度 n	组号	DD	ADD	递归合并时间	非递归合并时间	快排 1 时间	快排 2 时间
1	2000	1	2000	1	0.000321	0.000222	0.0056	0.000379
2	5000	2	5000	1	0.000817	0.00058	0.033484	0.0009
3	10000	3	10000	1	0.00147	0.001528	0.136779	0.001515
4	15000	4	15000	1	0.002161	0.002061	0.294021	0.002827
5	20000	5	20000	1	0.002685	0.002938	0.527427	0.00365
6	30000	6	30000	1	0.004097	0.004903	1.16582	0.005941

因为在随机生成的数组中，每组的 ADD 相差都比较大，很难达到 ADD 相近的情况，因此在测试该组数据的时候，我们人为地进行了构造，使得每一组的 ADD 值都为 1 。在 ADD 相同的情况下，我们对同一算法进行分析。发现在其他条件相同的情况下，**算法运行的时间随着 n 的增大不断增大。**

3 线性时间选择算法

3.1 算法说明

此算法模仿了递归划分排序算法，对输入数据进行划分排序。并且因为我们知道要找第几小的，因此，我们可以得知我们要找数的划分区域，减少递归的次数，因此其时间复杂性可以为线性的。在上机实验中，对于一分为二和一分为三两种方式我们都采用了。截图答案情况下，一分为二和一分为三结果相同。

在此时，为了在线性时间内找到一个划分基准，使得按这个基准所划分出的 2 个子数组的长度都至多。我们将 n 个元素 5 个 5 个划分成了一组，取出每一组的中位数，并在所有中位数中再找到值较大的中位数，以这个元素作为划分的基准。

• k-dist 值最小的基站

```
1
递归层数: 6
k-dist值第1小 568030:103.075
Program ended with exit code: 0
```

- k-dist 第 5 小的基站

```
5
递归层数: 6
k-dist值第5小 567883:126.096
Program ended with exit code: 0
```

- k-dist 值第 50 小的基站

```
50
递归层数: 6
k-dist值第50小 568074:208.475
Program ended with exit code: 0
```

- k-dist 值最大的基站

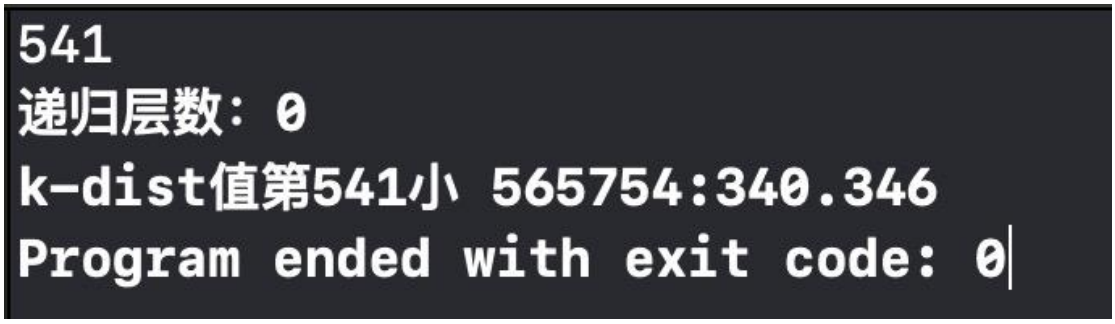
```
1033
递归层数: 6
k-dist值第1033小 568313:2735.8
Program ended with exit code: 0
```

- 不同划分情况

一分为二：

```
541
递归层数: 6
k-dist值第541小 565754:340.346
Program ended with exit code: 0
```

一分为三：



参照讲义 PPT，将教科书上的“一分为二”的子问题划分方法，改进为“一分为三”，比较这 2 种划分方式下，我们发现一分为三的减治法降低了递归深度，速度更快，在面对一些特殊位置的值的时候可以更快的命中。

4 平面最近点对算法

4.1 算法说明

选取垂直线 l:x=m 来作为分割直线。其中 m 为 S 中在 x 坐标上第⌈n/2⌉小、第(⌈n/2⌉+1)小的 2 个点的 x 坐标的平均值。由此将平面 S 分割为子平面 S1 和 S2。并且递归地在 S1 和 S2 上找出其最小距离 d1 和 d2，并设 d=min{d1,d2}，根据 d 构造点集 P1、P2

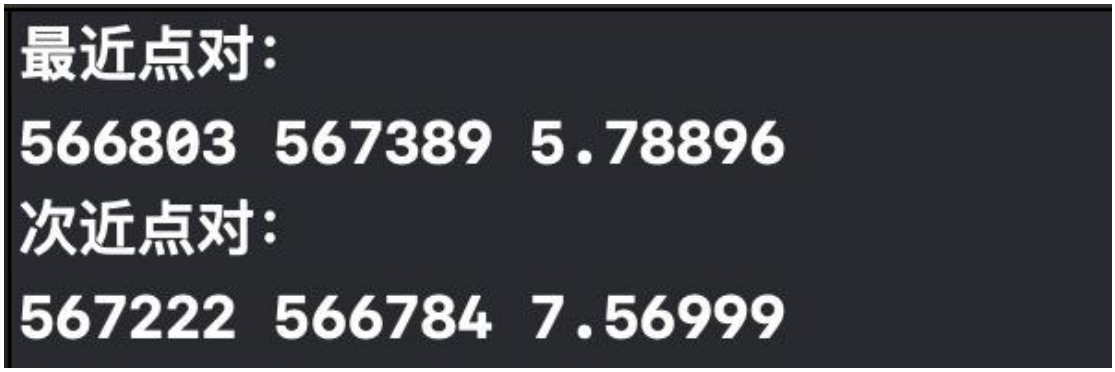
S 中的最接近点对(p, q)有 3 种情况

- 位于左边的 S1 中，距离 d=d1
- 位于右边的 S2 中，距离 d=d2
- 某个{p, q}，p∈ 左边点集合 P1 ，q∈ 右边点集合 P2

关于跨两个区域的情况，在中位线左右两侧找范围 d 以内的点进行查找，并且对于一个点来说，在另一边最多只有 6 个点满足条件。

4.2 算法结果

算法结果如下图展示，前两个数为用基站 ENodeBID 表示的基站点对，第三个数为两个基站间的距离。



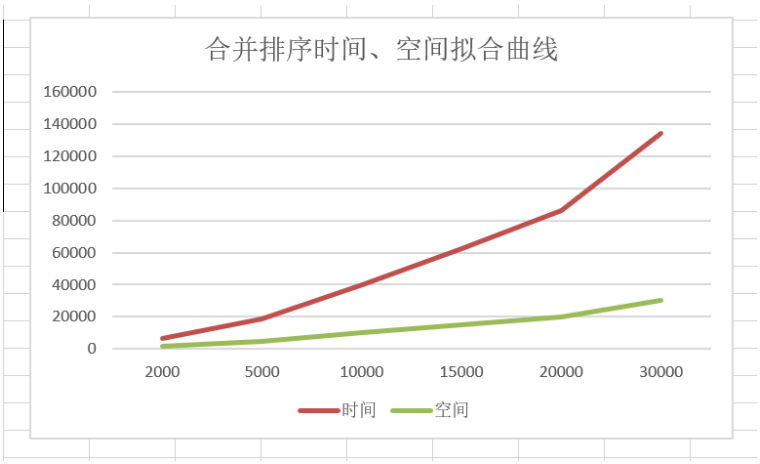
5 时间、空间复杂性分析

5.1 合并排序

复杂度分析

$$T(n)=\begin{cases} O(1) & n \le 1 \\ 2T(n/2)+O(n) & n>1 \end{cases}$$

T(n)=O(nlogn) 渐进意义下的最优算法

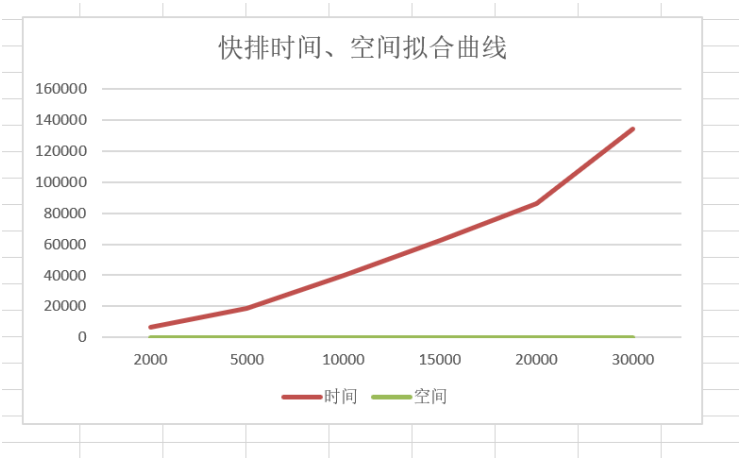


5.2 快速排序

复杂度分析

$$T(n)=\begin{cases} O(1) & n \le 1 \\ 2T(n/2)+O(n) & n>1 \end{cases}$$

T(n)=O(nlogn) 渐进意义下的最优算法

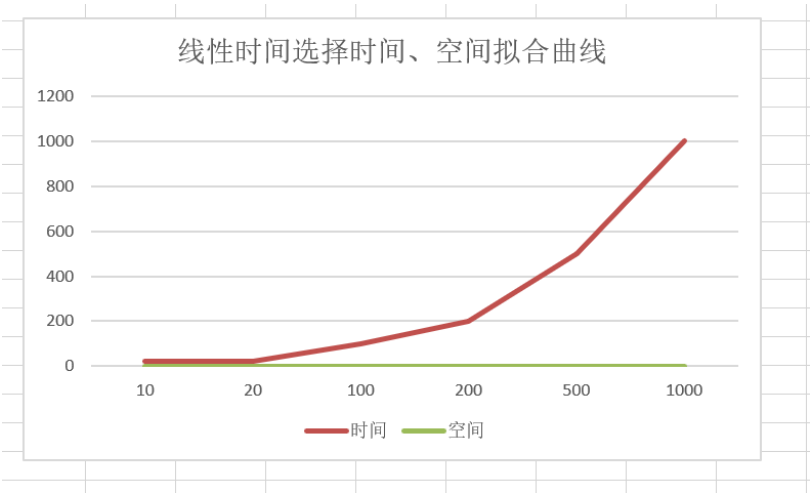


5.3 线性时间选择

复杂度分析

$$T(n)\le\begin{cases} C_1 & n<75 \\ C_2 n+T(n/5)+T(3n/4) & n\ge 75 \end{cases}$$

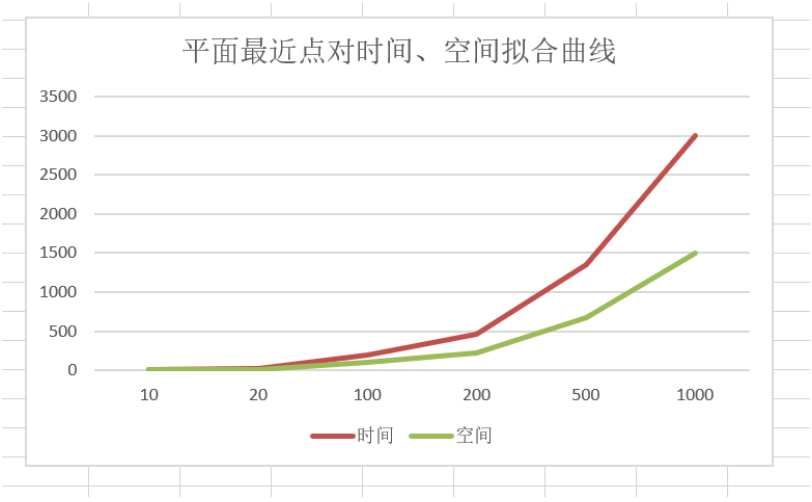
T(n)=O(n)



5.4 平面最近点对

$$T(n)=\begin{cases} O(1) & n<4 \\ 2T(n/2)+O(n) & n\geq 4 \end{cases}$$

T(n)=O(nlogn)



6 实验总结

在本次实验中，我们组总体花费的时间较长，对每个任务我们都付出了较多的努力。每个人的贡献度为 50%+50%。

关于优化性能方面：**第一**是在排序算法中，我们同时完成了 4 种排序算法。对于每种排序，我们都在操作之前检验了此时的数组是否是递增或者递减数组，当遇到这种情况的时候可以优化性能。并且我们都采用了全局变量来记录递归的层数。**第二**在快速排序中，考虑到划分后的子序列是否平衡、对称，我们采取在 a[p:r]中的随机一个元素作为划分元素。**第三**在线性时间选择算法中，我们采取了更加优化的策略，将 n 个元素 5 个 5 个划分成了一组，

取出每一组的中位数，并在所有中位数中再找到值较大的中位数，以这个元素作为划分的基准。这样子的策略进一步确保了划分区域的平衡性。并且我们同时编写了一分为二和一分为三两个程序，用于比较不同减治法的递归情况。**第四**，在平面最近点对算法中，在跨区域的时候，将待测点按照 y 的顺序进行排列，则只要检查每个点往上的有限区域即可，这样减少了重复的检查运算。

关于改进思路的一点想法。**第一**是在构造随机数列时，因为采用随机生成与随机打乱的策略，导致最后数组的 ADD 相差不大，可以在生成的时候人为的控制一些数的大小关系，让 ADD 的区分度更加大更加具有典型性。**第二**是在快排的时候，除了采取随机的策略之外，还可以考虑采用五个元素的分组方式，使划分更加平衡。**第三**，平面最近点对算法中，在排序跨区域数组 Z[]的时候，我们在实验中发现，采用快速排序的时间会变长，在改进的时候我们采用了冒泡排序的方式，时间就简短了。考虑到原因可能是因为此时 Z 的个数比较少，采用冒泡是更好的选择。**第四**，在做平面最近点对时，采用了预排序技术，即在使用分治法之前，预先将 S 中 n 个点依其 y 坐标值排好序，设排好序的点列为 P*。在执行分治法的第 4 步时，只要对 P*作一次线性扫描，即可抽取出我们所需要的排好序的 点列 P1*和 P2*。在第 5 步中再对 P1*作一次线性扫描，即可求 得 dl。因此，第 4 步和第 5 步的两遍扫描合在 一起只要用 O(n)时间。第三与第四点在我们的代码中已经实现。