深 圳 大 学 实 验 报 告

课程名称：算法设计与分析

实验项目名称：实验二：分治法——最近点对问题

学院：计算机与软件学院

专业：计算机科学与技术

指导教师：刘刚

报告人：林宪亮 学号：2022150130

实验时间：2024 年 3 月 31 日—2024 年 4 月 15 日

实验报告提交时间：2024 年 4 月 15 日

教务部制

一、实验目的：

（1）掌握分治法思想。
（2）学会最近点对问题求解方法。

二、实验内容：

1. 对于平面上给定的N个点，给出所有点对的最短距离，即，输入是平面上的N个点，输出是N点中具有最短距离的两点。
2. 要求随机生成N个点的平面坐标，应用蛮力法编程计算出所有点对的最短距离。
3. 要求随机生成N个点的平面坐标，应用分治法编程计算出所有点对的最短距离。
4. 分别对N=100000—1000000，统计算法运行时间，比较理论效率与实测效率的差异，同时对蛮力法和分治法的算法效率进行分析和比较。
5. 如果能将算法执行过程利用图形界面输出，可获加分。

三、实验内容及过程：

1. 蛮力法计算出所有点对的最短距离

（1）算法思想：蛮力法，即是对于给定的 N 个点进行两层循环，两两计算距离，最后得到最小的距离，算法简单，但是时间复杂度高，为 O（n^2）。

（2）伪代码：

Algorithm 1：Find Closest Points By Brute

Input：Array, startIndex, endIndex
Output: min_distance , index of the two points

```
1. if (end_index - start_index <= 0):
2. Return ∞
3. else:
4. for i in range(start_index, end_index):
5.     for j in range(start_index + 1, end_index + 1):
6.         dis = distance(arr[i], arr[j])
7.         dmin = dis if dis < dmin and i != j:
8. Return dmin
```

蛮力法的代码思路简单，如果数组只有一个点或者没有点，那么直接返回距离为正无穷，否则，则需要使用两层循环遍历这个数组中的每个点对，分别计算它们之间的距离，然后通过比较得出距离最小的点对，然后返回最小距离。值得注意的是，计算点对的距离的时候，可以进行优化，去掉开根号的步骤，这样可以减少运行的时间，同时也不会影响算法的结果。

（3）不同规模下的时间效率：

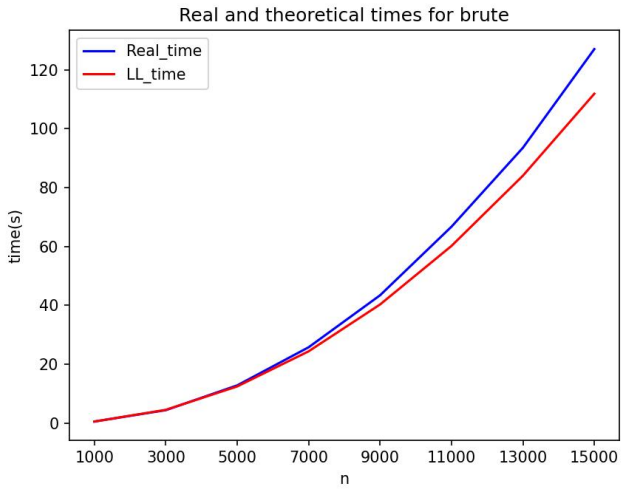

图 1 蛮力法的时间效率

如图 1，蓝色曲线为实际运行时间曲线，而红色曲线为理论时间曲线，理论曲线是使用规模为 1000 下的实际运行时间为基准计算而出，可以看出实际运行时间比理论运行时间要长一些。

表 1：蛮力法在不同规模下的运行时间

Size	$1*10^3$	$3*10^3$	$5*10^3$	$7*10^3$	$9*10^3$	$11*10^3$	$13*10^3$	$15*10^3$
Time（s）	0.497	4.367	12.785	25.737	43.391	66.695	93.507	127.023

如上表 1，记录了在不同规模下，蛮力法寻到最小距离所需要的时间（运行 5 次取平均值）。

2. 分治法计算出所有点对的最短距离

（1）算法思想：对给定的 N 个点平均分成两个部分，分别计算各部分的最短距离，然后合并检查有没有出现更短距离的两个点。对于分出的各个部分，采用一样的策略求最短距离，即继续平均的分成两个部分，计算各部分的最短距离，再合并检查有没有出现更短距离的两个点，以此递归寻找所有点对的最短的距离。

（2）步骤：

- 对于给定的 N 个点采用快速排序进行排序（按 x 值进行排序）。
- 确定递归的返回条件：

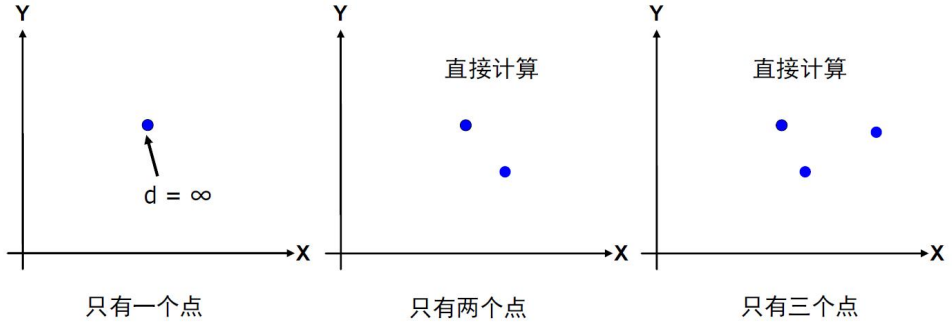

图 2 返回条件

当分出的组只有一个点的时候，直接返回距离为无穷大。
当分出的组有两个点或者三个点的时候，使用蛮力法计算它们的最小距离，然后返回。

• 当点数大于 3 时，直接使用中位数为分界线分成两个组 dl，dr。

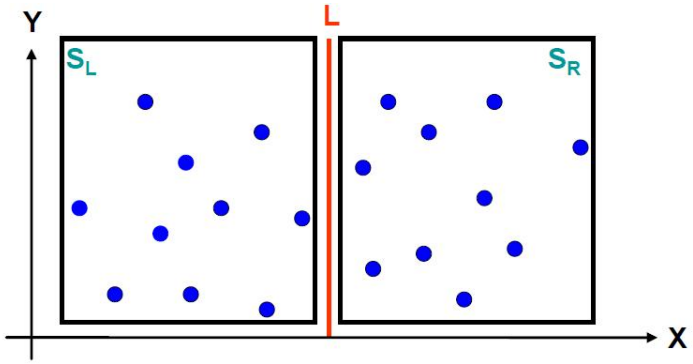

图 3 分成两个组

• 取 d=min(dl, dr)，在直线 L 两边分别扩展 d，得到边界区域 Y，Y'是区域 Y 中的点按照 y 坐标值排序后得到的点集（方便后续使用鸽舍定理降低时间复杂度）。

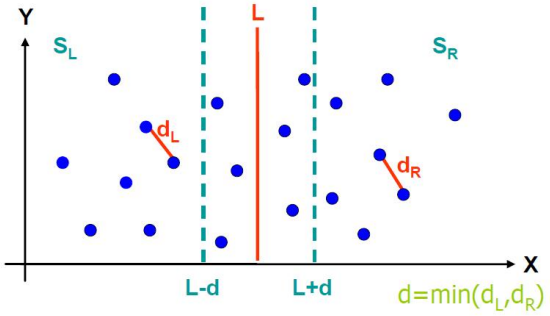

图 4 合并更新最短距离

• 对于 Y'中的每个点，计算出它和它接下来的六个点的距离，更新最短距离。
可以证明，对于 Y' 中的每个点，只有它接下来的六个点才有机会和它的距离小于 d。
因此，此步的时间复杂度为 O（n）。
证明过程：设此点为 x
只有在以 x 为上顶边中点画出的长为 2d，高为 d 的长方形中的点才有机会和 x 的距离小于 d。
把这个长方形平均切成八个正方形块，每一个块中最多只能有一个点，如果有两个点，那么这两个点的距离一定小于 d，与前提矛盾，而 x 会同时是其中两个块的点，因此只剩下了六个其余的点有可能。

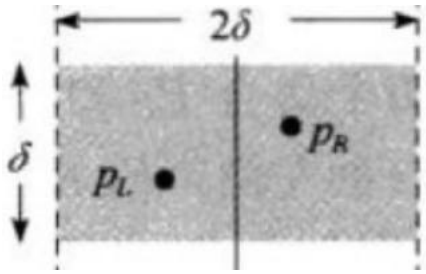

图 5 鸽舍定理

（3）伪代码：

• 使用分治法实现最近点对问题

Algorithm 2：Find Closest Points By Partition

Input：Array, startIndex, endIndex

Output: min_distance , index of the two points

```
1. if (end_index - start_index <= 0):
2. Return ∞
3. else:
4.     if（end_index - start_index <= 3）Return    Find Closest Points By Brute
5.     else:    dl=Find Closest Points By Partition(array,startIndex,mid)
6.              dr=Find Closest Points By Partition(array,mid+1,endIndex)
7.              d=min（dl，dr）
8.              Find closest after hebing
9. Return dmin
```

• 使用根据 x 进行排序的快速排序

Algorithm 2.1：Quick Sort By X

Input：Array

Output: Sorted Array

```
1. if len(arr) <= 1:
2. return arr
3. else:
4.   arrleft=[a if a.x < arr[left].x for a in arr]
5.   arrright=[a if a.x > arr[left].x for a in arr]
6. return fast_sort_x(arrleft) + [arr[left]] + fast_sort_x(arrright)
```

• 使用根据 y 进行排序的快速排序

Algorithm 2.2：Quick Sort By Y

Input：Array

Output: Sorted Array

```
1. if len(arr) <= 1:
2. return arr
3. else:
4.   arrleft=[a if a.y < arr[left].y for a in arr]
5.   arrright=[a if a.y > arr[left].y for a in arr]
6. return fast_sort_x(arrleft) + [arr[left]] + fast_sort_x(arrright)
```

• 计算合并后最短距离点对算法

Algorithm 2.3：Find Closest After Hebing

Input：Array ,dim

Output: new dmin

```
1. Array'=[a if |a.x - mid.x| < dmin for a in Array]
2. Quick Sort By Y （Array'）
3. for i in range(len(arrlj)-1 ):
4.   for j in range(i + 1,min(i+7,len(arrlj)) ):
5.       if (fabs(arrlj[i].y - arrlj[j].y) < dmin and i != j):
6.               dis3 = distance(arrlj[i], arrlj[j])
7.               dmin = dis3 if (dis3 < dmin)
8. return dmin
```

（4）不同规模下的时间效率：

• 小规模下：

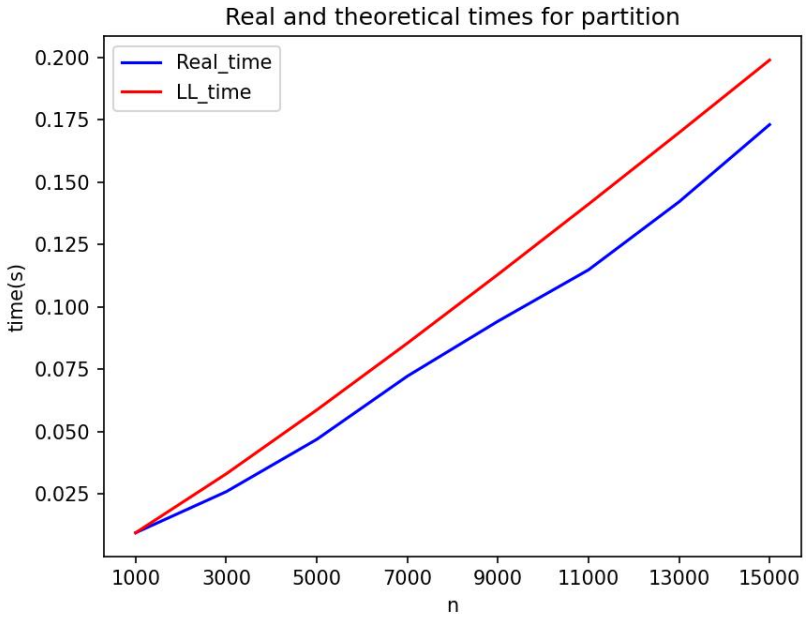

图 6 小规模下的时间效率

如图 6，为分治法在数据规模为 1000-15000 下的时间效率图，蓝色曲线为实际运行时间，而红色曲线为理论运行时间，可以看出实际运行时间是低于理论值的，并且它们之间的 gap 是比较大的，可能是因为 y 轴的标尺较小，所以画出来的图看着 gap 比较大，其实绝对的时间差异并不大，也可能是因为数据规模太小了，使不考虑低阶项导致的误差变大。

表 2. 小规模下的时间效率

Sizes	1000	3000	5000	7000	9000	11000	13000	15000
LL_Time(s)	0.009	0.033	0.058	0.085	0.112	0.141	0.169	0.198
R_Time(s)	0.009	0.025	0.046	0.072	0.094	0.114	0.142	0.173

如表 2，第二行为运行的理论时间，第三行为实际的运行时间。

表 3 小规模下的实际运行时间记录

Epoch / Size	1	2	3	4	5
1000	0.007	0.009	0.006	0.012	0.010
3000	0.024	0.029	0.020	0.030	0.024
5000	0.044	0.054	0.044	0.045	0.045
7000	0.068	0.074	0.069	0.070	0.078
9000	0.097	0.091	0.093	0.093	0.094
11000	0.116	0.116	0.121	0.109	0.110
13000	0.140	0.138	0.146	0.147	0.137
15000	0.171	0.183	0.188	0.164	0.157

上表为分治法在小规模数据下的实际运行时间记录。

• 大规模：

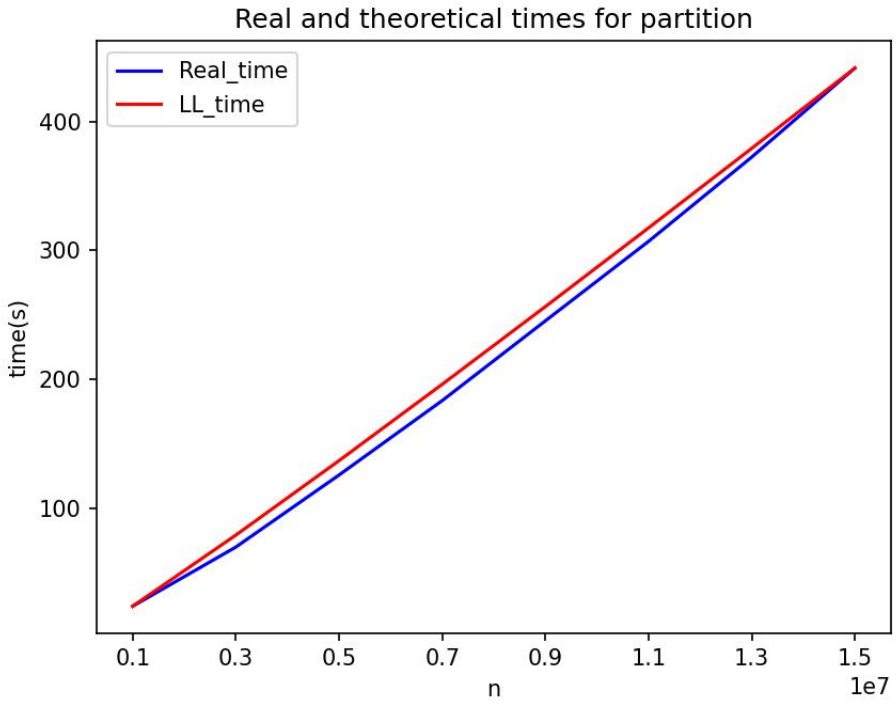

图 7 大规模下的时间效率

如图 6，为分治法在数据规模为 $1*10^6$-$1.5*10^7$ 下的时间效率图，蓝色曲线为实际运行时间，而红色曲线为理论运行时间，可以看出实际运行时间依旧是低于理论值的，而且在大规模数据下，它们之间的 gap 并没有很大。

表 4 大规模下的时间效率

Sizes	$1*10^6$	$3*10^6$	$5*10^6$	$7*10^6$	$9*10^6$	$11*10^6$	$13*10^6$	$15*10^6$
LL_Time(s)	24.59	79.66	137.31	196.43	256.58	317.53	379.13	441.28
R_Time(s)	24.57	70.39	126.17	183.79	245.54	307.10	372.34	441.28

如表 3，第二行为运行的理论时间，第三行为实际的运行时间(运行 5 次取平均)。

表 5， 大规模下的实际运行时间记录

Epoch / Size	1	2	3	4	5
$1*10^6$	20.201	25.153	25.685	25.987	25.835
$3*10^6$	70.593	68.849	71.136	71.286	70.127
$5*10^6$	126.206	125.235	126.696	126.892	125.834
$7*10^6$	184.240	181.349	182.757	187.364	183.282
$9*10^6$	244.333	242.182	245.049	250.493	245.681
$11*10^6$	305.830	298.486	313.659	307.697	309.827
$13*10^6$	366.197	366.482	386.846	370.534	371.681
$15*10^6$	434.624	441.487	452.939	441.313	436.046

上表为分治法在大规模数据下的实际运行时间记录。

3. 优化后的分治法

3.1 细节优化：
对距离求解过程进行优化，从原来的求距离变成求解距离的平方，减少了开根号这一步的时间花销。
3.2 合并求最短距离的点数的优化
根据周玉林的算法改进，我们合并时只需要检验四个点的距离即可，而不需要检验六个点了，所以这一步的时间复杂度从 3n 优化到了 2n。计算距离的复杂度在最坏的情况下也由原来的 3nlogn 减少到了 2nlogn。
3.3 合并过程的排序的优化

Algorithm 3：Find Closest Points By Partition（improve 1）

Input：Array, startIndex, endIndex
Output: min_distance , index of the two points
1. if (end_index - start_index <= 0):
2. Return ∞
3. else:
4. if（end_index - start_index <= 3）
5. Find Closest Points By Brute
6. **Merge（arrayleft，arrayright）**
7. Return dmin

```
8.    else:    dl=Find Closest Points By Partition(array,startIndex,mid)
9.             dr=Find Closest Points By Partition(array,mid+1,endIndex)
10.            Merge（arrayleft，arrayright）
11.            d=min（dl，dr）
12.            Find closest after hebing
13. Return dmin
```

Algorithm 3.3：Find Closest After Hebing（improve 1）

```
Input：Array ,dim
Output: new dmin
1.  Array'=[a if |a.x - mid.x| < dmin for a in Array]
2.  for i in range(len(arrlj)-1 ):
3.    for j in range(i + 1,min(i+7,len(arrlj))  ):
4.        if (fabs(arrlj[i].y - arrlj[j].y) < dmin and i != j):
5.            dis3 = distance(arrlj[i], arrlj[j])
6.            dmin = dis3 if (dis3 < dmin)
7.  return dmin
```

主要修改的地方为对 y 的排序，如果采用快速排序根据 y 的大小进行排序，那么时间复杂度就会达到 n（logn）^{2}，因此这是不可取的，改进的方法就是采用归并排序，因为归并排序“递”的过程和分治法拆分数组求最小距离的过程是一致的，所以不需要花额外的时间，只需要花“归”过程的额外时间，所需的时间复杂度为 O（n），这样最后的时间复杂度可以缩减到 O（nlogn）。

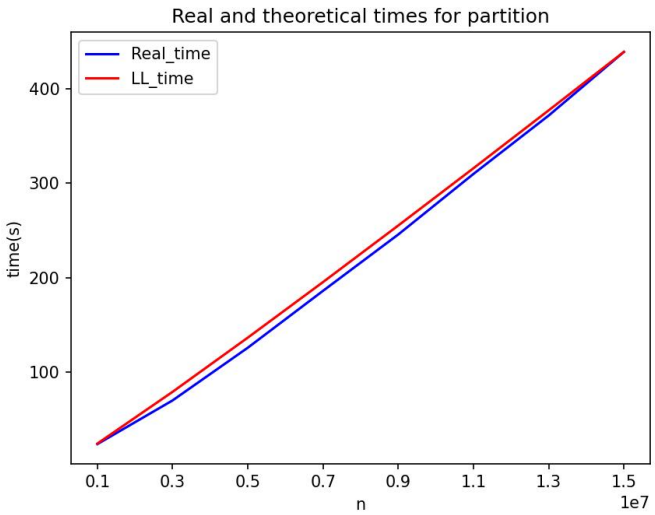

图 8 改进后的分治法

如图 6，为分治法在数据规模为 1*10^6-1.5*10^7 下的时间效率图，蓝色曲线为实际运行时间，而红色曲线为理论运行时间，可以看出实际运行时间依旧是低于理论值的。

表 6 运行时间记录

Sizes	$1*10^6$	$3*10^6$	$5*10^6$	$7*10^6$	$9*10^6$	$11*10^6$	$13*10^6$	$15*10^6$
LL_Time(s)	24.47	72.26	130.62	190.45	251.30	313.94	375.24	437.08
R_Time_ Improve(s)	23.91	70.17	125.96	181.17	243.80	305.91	370.74	437.08
R_Time(s)	24.57	70.39	126.17	183.79	245.54	307.10	372.34	441.28

对比可以看出改进后，在运行时间上是有所提升的。

3.4 合并过程使用蛮力法

Algorithm 4：Find Closest Points By Partition（improve 2）

Input：Array, startIndex, endIndex

Output: min_distance , index of the two points

```
1. if (end_index - start_index <= 0):
2. Return ∞
3. else:
4.     if（end_index - start_index <= 3）Return    Find Closest Points By Brute
5.     else:    dl=Find Closest Points By Partition(array,startIndex,mid)
6.              dr=Find Closest Points By Partition(array,mid+1,endIndex)
7.              d=min（dl，dr）
8.              Find closest after hebing
9. Return dmin
```

Algorithm 4.3：Find Closest After Hebing（improve 2）

Input：Array ,dim

Output: new dmin

```
1.  Arrayleft=[a if -(a.x - mid.x) < dmin for a in Array]
2.  Arrayright=[a if (a.x - mid.x) < dmin for a in Array]
3.  for i in range(len(arrleft) ):
4.    for j in range(len(arrright):
5.         if (fabs(arrlj[i].y - arrlj[j].y) < dmin ):
6.                dis3 = distance(arrlj[i], arrlj[j])
6.                dmin = dis3 if (dis3 < dmin)
7. return dmin
```

在合并过程使用蛮力法，即去掉归并前排序的步骤，而是直接选取出左右半区符合 |a.x-mid.x|<dmin 的点，然后使用蛮力法循环计算两两距离，然后比较求解更短的距离。根据对点平均分布下的结论，中间区域的有 $n^{1/2}$ 个数，所以最后的算法时间复杂度达到 O（nlogn）。

表 7 部分蛮力法运行时间记录

Sizes	$1*10^6$	$3*10^6$	$5*10^6$	$7*10^6$	$9*10^6$	$11*10^6$	$13*10^6$	$15*10^6$
R_Time_ Improve1(s)	23.91	70.17	125.96	181.17	243.80	305.91	370.74	437.08
R_Time(s)	24.57	70.39	126.17	183.79	245.54	307.10	372.34	441.28
R_Time_ Improve2(s)	21.91	67.22	120.21	173.89	237.32	298.12	360.67	425.85

从上表中不难得出，部分蛮力法拥有着更好的算法性能。

4. 图形化界面

图 9 图形化界面

如图，可以在“Number of points”输入框中输入想要生成的点的数量。可以选择产生随机点的阈值，在“Minimum value”输入框中输入最小值，在“Maximum value”输入框中输入最大值。然后点击“Generate Ramdom Points”即可产生随机点。

下图为产生一千个随机点的示例：

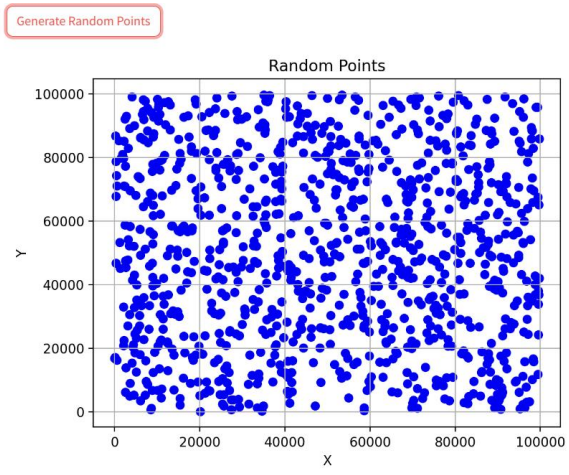

图 10 产生一千个随机点

• 点击“Find Closest Points By Brute”即可使用蛮力法寻找最短距离点对，寻找到的两个点会分别标记为红色和黄色。如下图所示。

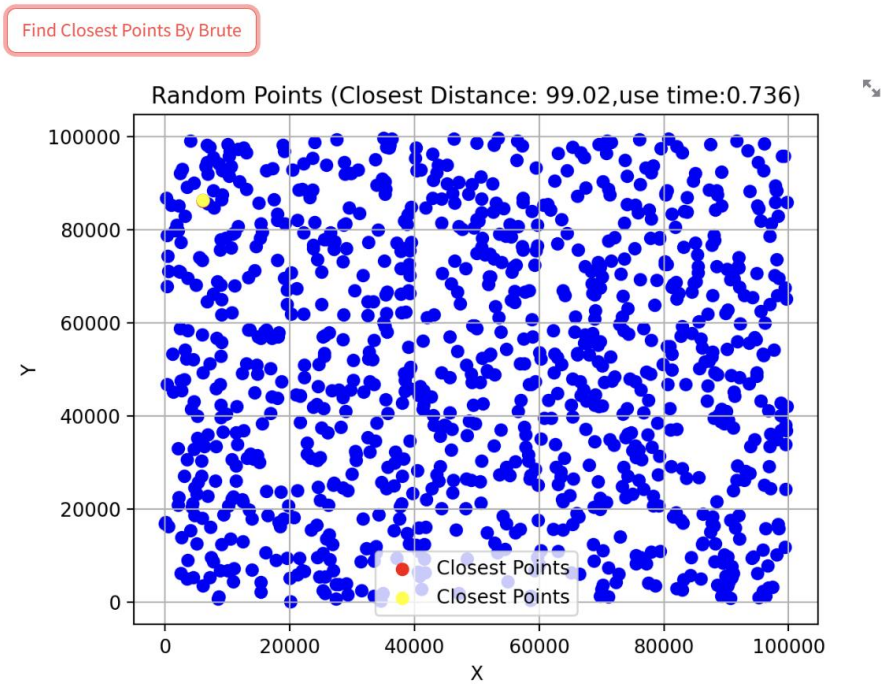

图 11 蛮力法寻找

• 点击“Find Closest Points By Partition”即可使用分治法寻找最短距离点对，同样的，寻找到的最短点对会标记为红色和黄色。

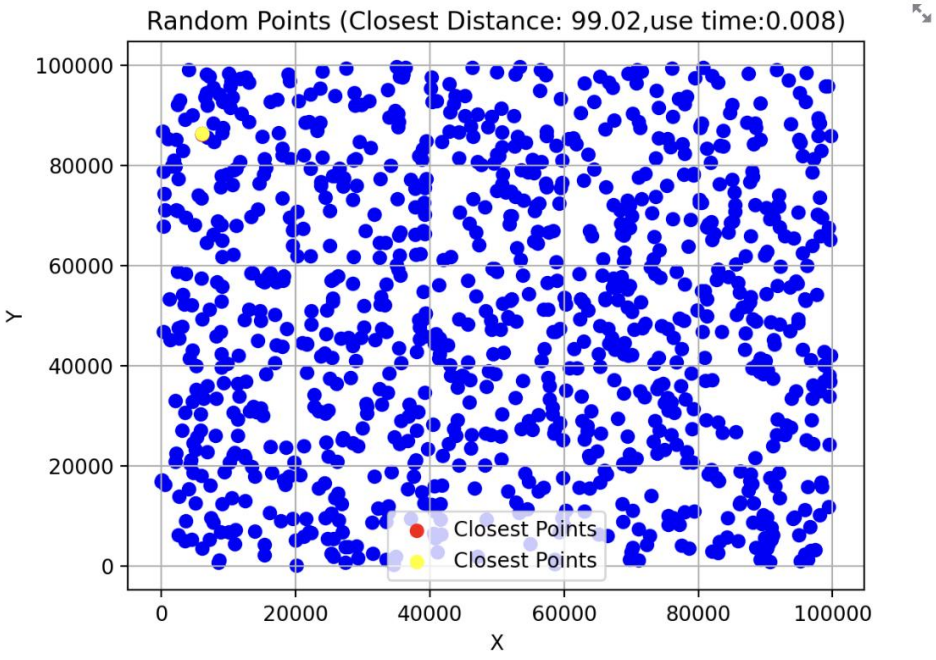

图 12 分治法寻找

可以看出使用分治法和蛮力法寻找出的最短距离点对是一样的，但是分治法显然有着更好的时间效率。

四、实验结果及分析：

1. 时间复杂度分析：

• 蛮力法：蛮力法需要用到两层循环，每一层循环的次数都与点的规模 n 有关，很容易得出时间复杂度为 O（n^2）。

• 分治法：使用分治法解决点对问题时，但 n<=3 使，时间复杂度为 O（1），当 n>3 时，时间复杂度为 T（n）= 2 * T（n/2）+ O（nlogn），其中 2 * T（n/2）为分成两个子序列进行求解需要的时间，O（nlogn）为合并求解时需要的时间，由于对于合并求解时的时间复杂度已经控制在 O（n），所以这一步的时间复杂度取决于快速排序的时间复杂度 O（nlogn），最后根据主定理求解，可得到算法的时间复杂度是 O（n（logn）^{2}）。

• 改进的分治法：对分治法中对 y 排序的步骤进行改进，使用归并排序，可以使时间复杂度降低到 O（nlogn）。对细节的优化也可以有效减少实际运行时间。使用部分蛮力法，也可以有效的减少运行时间。

2. 蛮力法和分治法的运行效率比较：

分治法比起蛮力法有着十分优越的运行效率。

表 6　蛮力法分治法运行效率比较

Sizes / 方法	1000	3000	5000	7000	9000	11000	13000
分治法	0.009	0.025	0.046	0.072	0.094	0.114	0.142
蛮力法	0.497	4.367	12.785	25.737	43.391	66.695	93.507

如表一，可以看出，即使在小规模数据下，分治法都比蛮力法快了 50 倍到 600 倍，随着数据规模的增大，它们之间的差距还会更大。

3. 理论和实际运行效率比较：

对于蛮力法，实际运行时间会比理论的运行时间更长，而且随着规模的增大，gap 有着增长的趋势。

对于分治法，实际的运行时间比理论时间更短，而且随着规模的增大，它们之间的 gap 会越来越细微。可能随着规模增大，偶然性减少，更趋于稳定。

五、实验结论：

1. 蛮力法的时间复杂度为 O（n^2），分治法时间复杂度为 O（nlogn）。
2. 在实测的实际运行时间上，分治法所需的时间远远的小于蛮力法。
3. 蛮力法的实际运行时间比理论运行时间更长，而分治法反之。
4. 使用鸽舍定理可以使合并求解最短距离步骤的时间复杂度降低为 O（n）。
5. 合并过程检验的点数最少可以是四个，这可以有效的降低时间复杂度。
6. 本题中，采用快速排序不如使用归并排序，主要是因为使用归并排序可以节省“递”步骤的时间。
7. 使用部分蛮力法拥有更好的时间效率，应该是因为它不需要对 y 进行重新排序。
8. 采用图形化界面可以更加直观的观测运行结果。

指导教师批阅意见：   成绩评定：   指导教师签字： 2024 年 4 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。