**深 圳 大 学 实 验 报 告**

**课程名称： 算法分析与设计**

**实验名称：** **分治法求最近点对问题**

**学院： 计算机与软件学院 专业： 软件工程**

**报告人：** **敖宇飞、朱伟晔** **学号： 2021150241、2021150205**

**班级： 软件工程02班**

**指导教师： 杜智华**

**实验时间： 2024/3/30**

**实验报告提交时间： 2024/4/11**

**教务处制**

## 实验二 分治法求最近点对问题

## 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、实验概述

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、实验内容以及步骤

**（一）、蛮力法求解**

1. 算法原理：
   1. 存在N个点，那么就存在N（N - 1）/ 2 对点间的距离。穷举所有情况，选出最小值。
2. 伪代码：

*for i = 1 to N - 1*

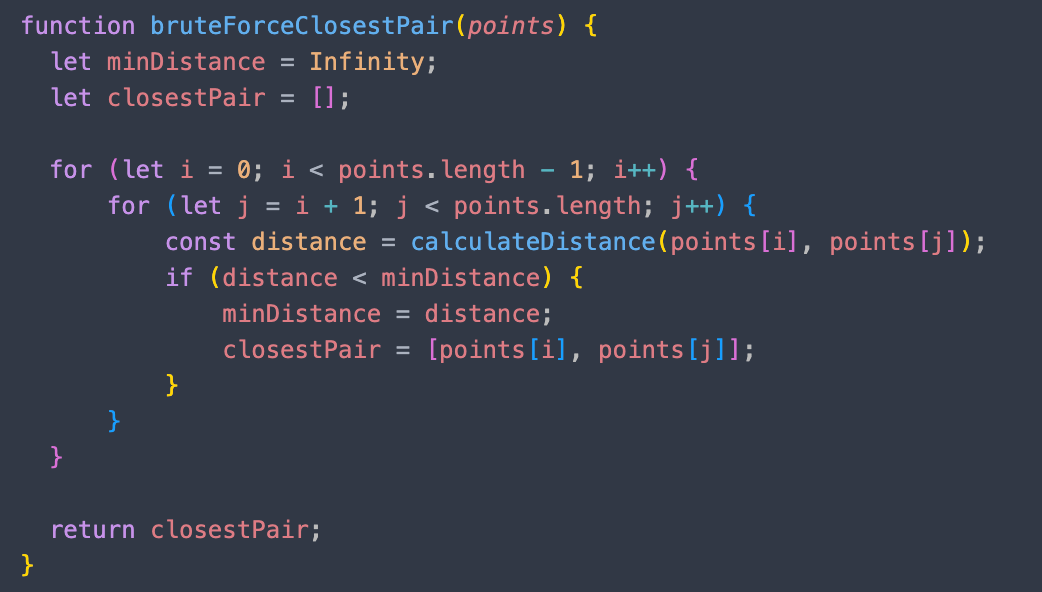
*for j = i + 1 to N*

*if ( dis(p[i], p[j]) < min )*

*min = dis(p[i], p[j])*

1. 复杂度分析：
   1. 需要遍历N（N - 1）/ 2种情况来找出最小值，最好最坏和平均情况的时间复杂度都为 O(n^2)
   2. 需要一个临时变量用来存储最小值，所以空间复杂度为 O(1)。
2. 上述思路代码实现：

这段代码是一个名为 bruteForceClosestPair 的函数，其目的是在给定的点集 points 中找到距离最近的点对。这个函数使用了一种简单粗暴的方法，即遍历所有可能的点对，并计算它们之间的距离，然后记录下最小距离和对应的点对。



1. 数据测试

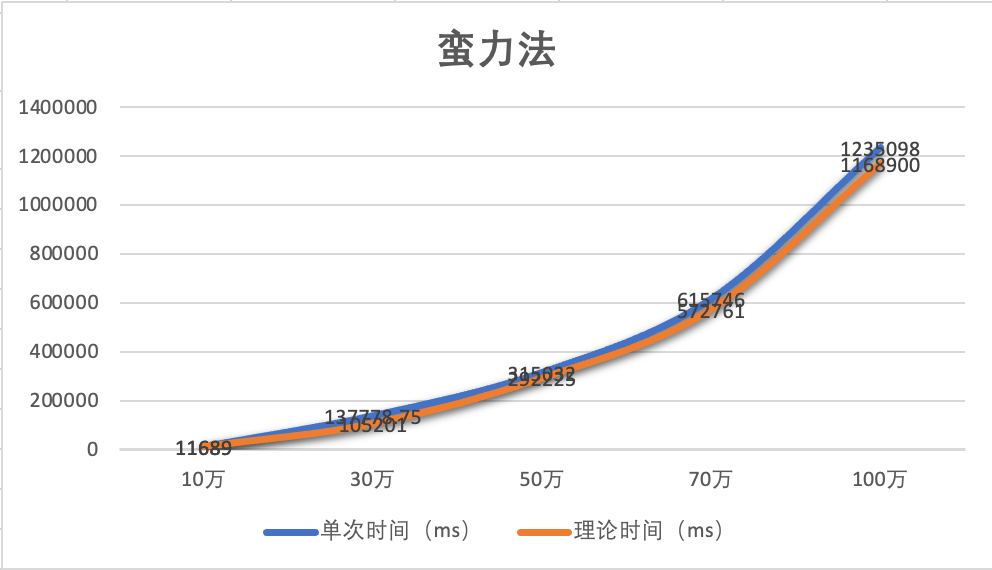
使用随机数生成，均匀分布的生成了10万、30万、50万、70万以及100万的数据集。为了减少数据的偶然性，每个数据量都进行了10次测试并取平均值。

为了检验实验是否准确，将实际值将理论值进行对比（基准点为10万）。理论值计算方法如下：



最终结果如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 10万 | 30万 | 50万 | 70万 | 100万 |
| 单次时间（ms） | 11689 | 137778.75 | 315032 | 615746 | 1235098 |
| 理论时间（ms） | 11689 | 105201 | 292225 | 572761 | 1168900 |



图像上符合O(n^2) 二次曲线，并且理论值与实际值误差较小。

**（二）、分治法求解**

1. 分治法基本思路

对于本题而言，可以转化为：

分治法基本思路：

分 -- 将问题分解为规模更小的子问题；

治 -- 将这些规模更小的子问题逐个击破；

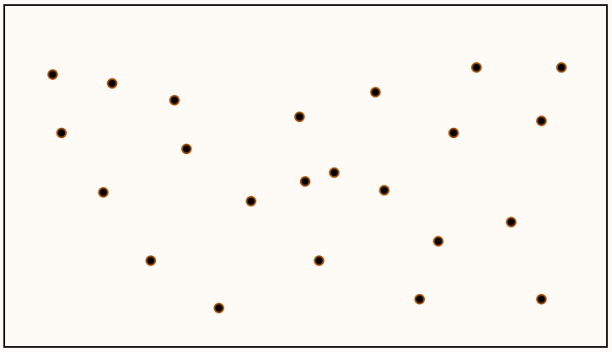
合 -- 将已解决的子问题合并，最终得出“母”问题的解；

本题思路：

分 -- 将整体分为左右两个区域；

治 -- 递归计算左右两区域的最短距离；

合 -- 合并左右区域，并求合并后的最短距离；

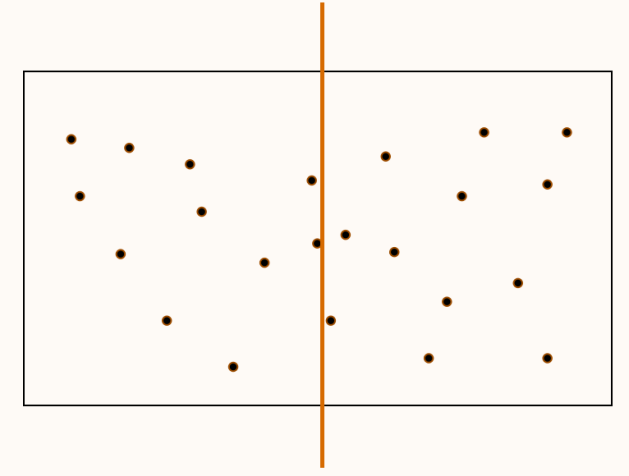


* 1. 分 -- 将整体分为左右两个区域；

将所有点根据x坐标进行排序，取中间点。所以算法时间复杂度下限：O（nlogn）

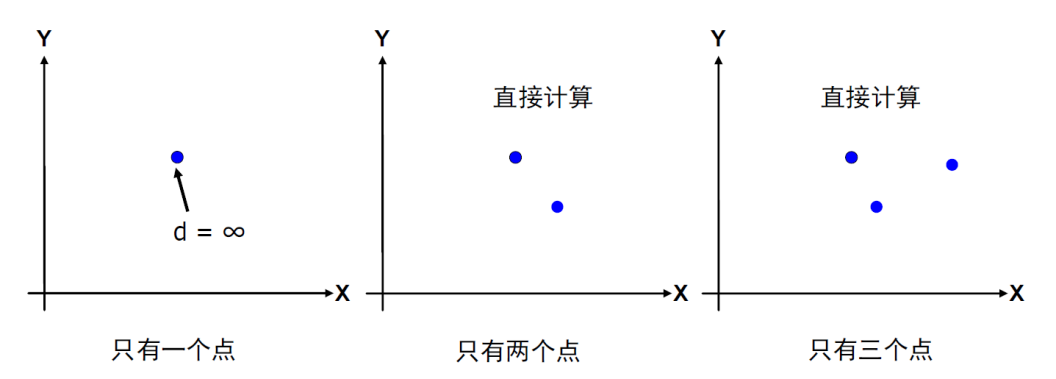
*mid = (l + r) / 2*

做到左右区域点集数目基本相同，降低数据随机性带来的影响



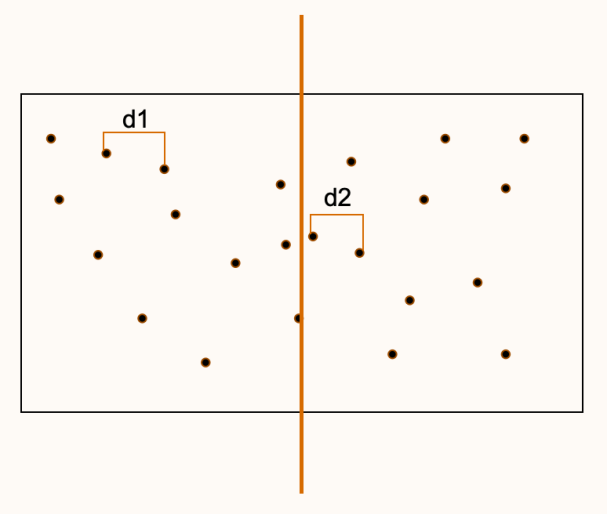
* 1. 治 -- 递归计算左右两区域的最短距离

子问题最小规模



递归调用函数，即可获取左右两区域的最短距离

* 1. 合 -- 合并左右区域，并求合并后的最短距离



**【问题转化为：“已知左右区域各自最短距离，求合并后的最短距离。”】**

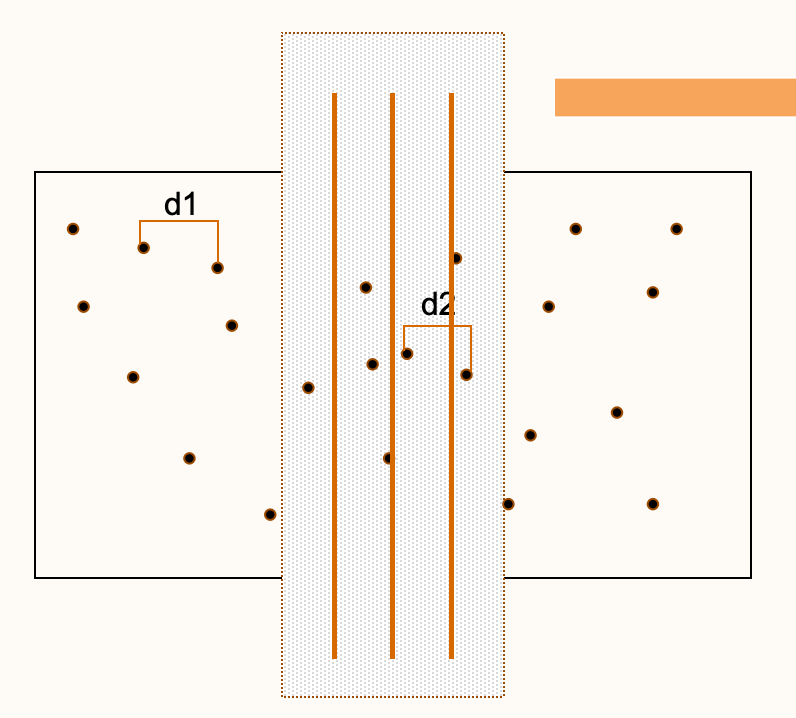
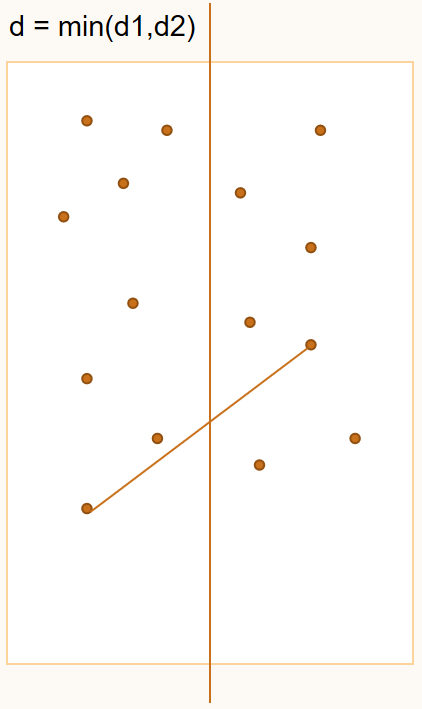
合并之后最短点对的选择一共有三种情况 ：左+右、左+左、右+右

对于左+左、右+右的情况，利用第二步中的递归调用即可获取。

所以主要问题在于，如果最短点对来自于左+右的合并操作。

解决思路：

两点必定来自于中轴线左右两侧附近，并且两点距离小于min(d1,d2)

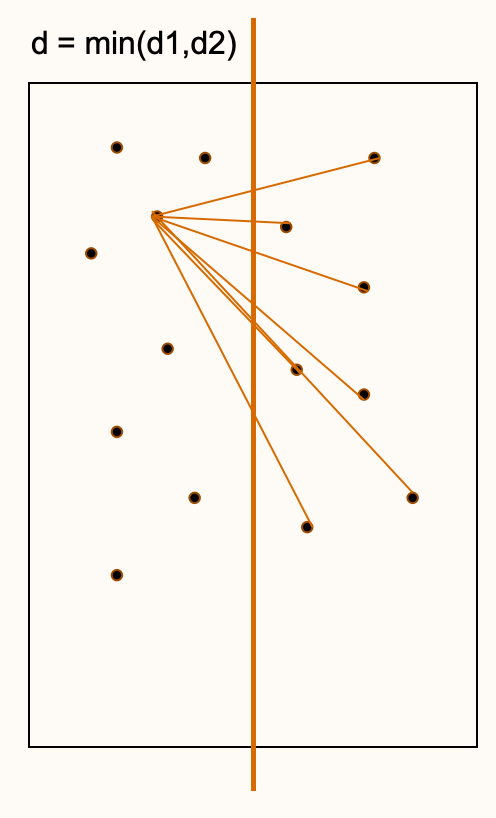
 

如左图所示，可以在中轴线附近取带状区域，其左右宽度为min(d1,d2)。在如此带状区域内，左右两点距离必定小于min(d1,d2)，极限状态为一侧宽度长度。

**【问题转化为：“从左右区域内各取一点，与d1，d2比较，取最短距离。”】**

解决方法有如下展示三种。

1. 分治——部分蛮力
2. 算法原理：



遍历左右带中的所有点，比较获得最短距离。

1. 伪代码：

ans = min(d1,d2)

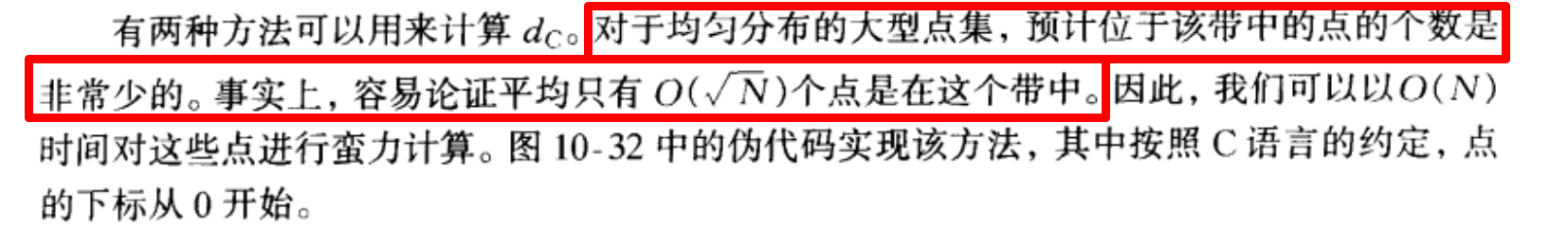
for i = 1 to left.size

for j = 1 to right.size

if ( dis(left[i], right[j]) < ans )

ans = dis(left[i], right[j])

1. 复杂度分析：

-----《数据结构与算法分析-C语言描述》P280

查询资料可知，对于均匀分布的点集而言，带中的元素个数为√n

**递推公式为：**

平均情况 T(n) = 2T(n/2) + (√n)^2

总体时间复杂度O(nlogn)

对于非均匀分布的点集，最差情况带中的元素个数为n

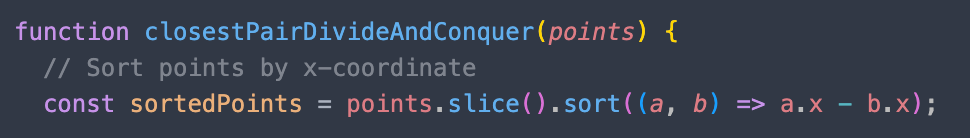
**递推公式为：**

最坏情况 T(n) = 2T(n/2) + (n)^2

总体时间复杂度O(n^2)

1. 上述思路代码实现：

先对点进行x轴上的排序：

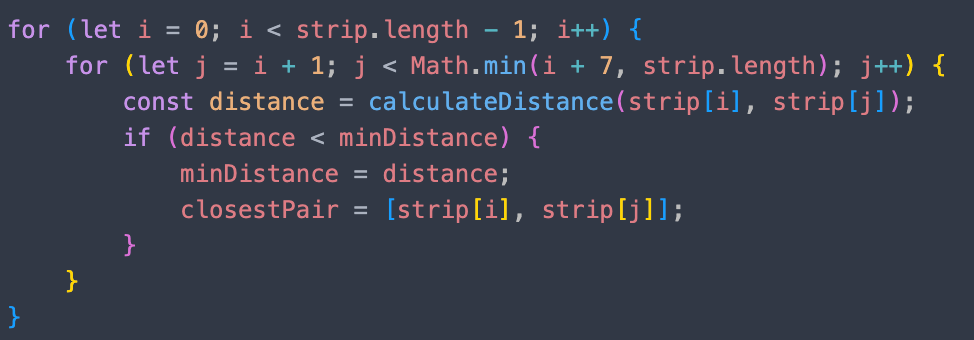


初始化变量：设置一个变量 minDistance 来存储最小距离，初始值为无穷大（Infinity），以及一个数组 closestPair 来存储距离最近的点对。

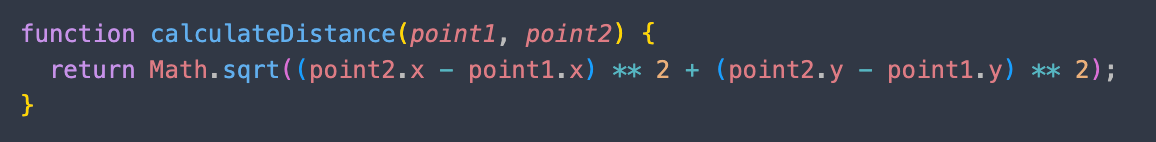




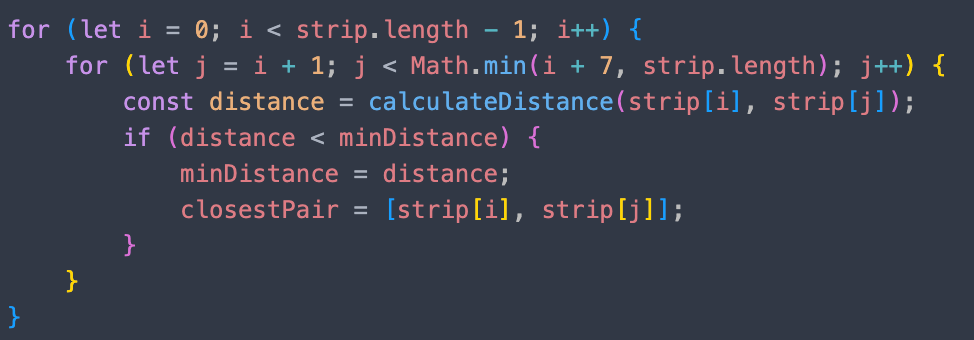
双重循环遍历：使用两个嵌套的循环来遍历点集中的所有点对。外层循环从第一个点开始，内层循环从外层循环当前点的下一个点开始，确保每一对点只被比较一次。



计算距离：对于每一对点，调用 calculateDistance 函数来计算它们之间的距离。



更新最小距离和点对：如果计算出的距离小于当前记录的最小距离，则更新 minDistance 和 closestPair。

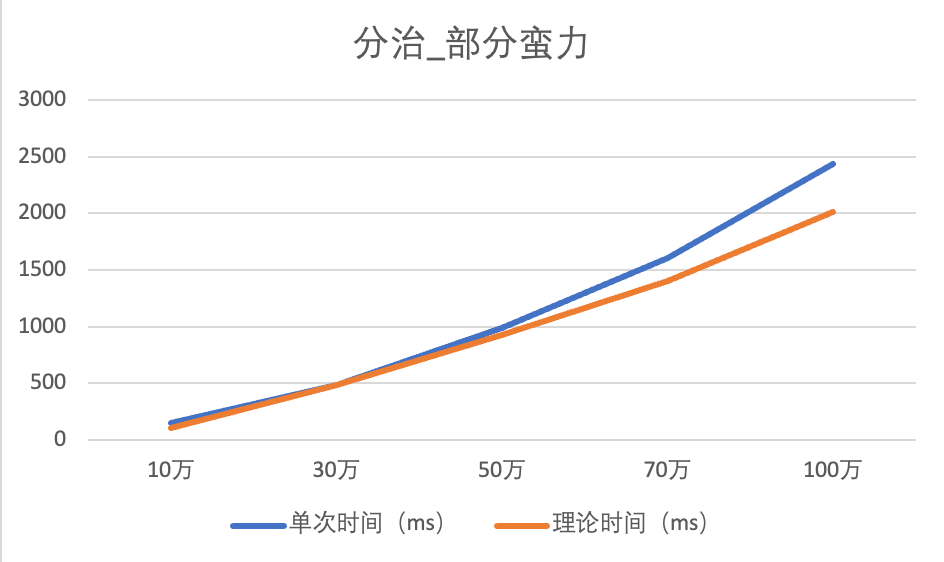


返回结果：在遍历完所有点对后，closestPair 将包含距离最近的点对，函数返回这个点对。

1. 数据分析：

最终结果如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 10万 | 30万 | 50万 | 70万 | 100万 |
| 单次时间（ms） | 152.599 | 482.966 | 993.825 | 1608.5 | 2310 |
| 理论时间（ms） | 108.988 | 482.966 | 925.837 | 1407.656 | 1993.231 |



图像上符合O(nlogn) 曲线，并且理论值与实际值误差较小。

显然对于最差情况仍然为O(n^2)，需要改进。

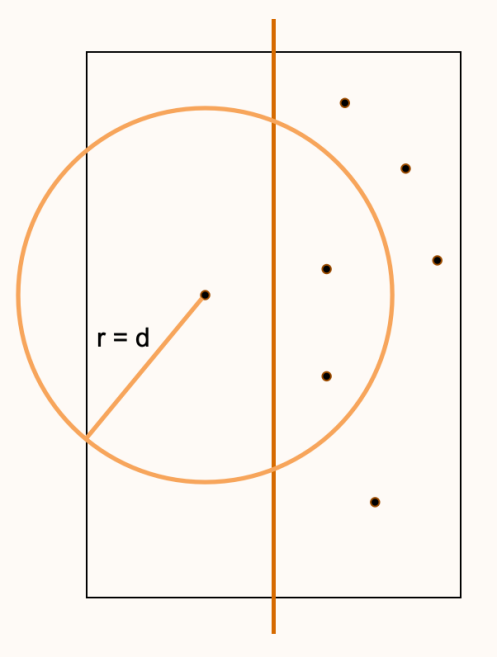
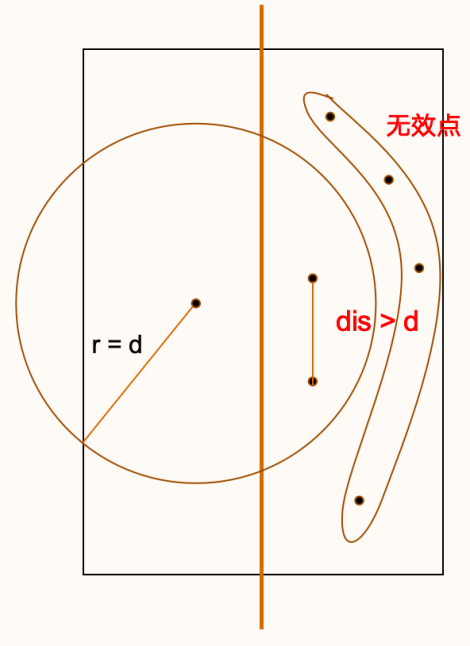
1. 分治——多趟查询
2. 算法原理：

遍历左带内的所有点，并与右带内所有符合条件的点进行长度比较。

右侧符合条件点集筛选过程及原理：

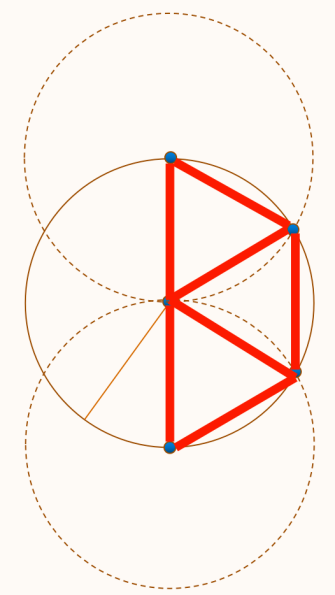
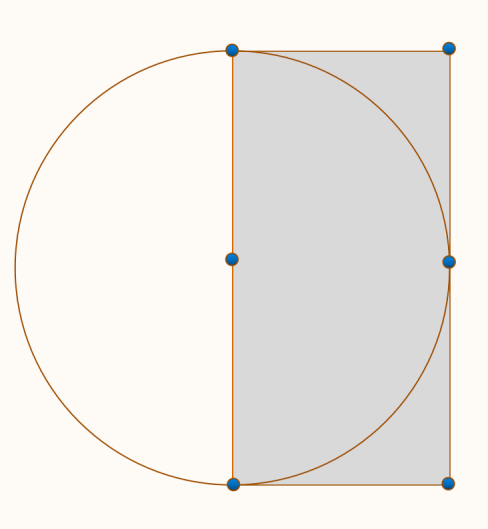
***结论：***右侧点位于以左侧点为中心，上下高度d，右侧宽度d的d\*2d的范围内。且右侧点的个数存在上限。

证明如下：

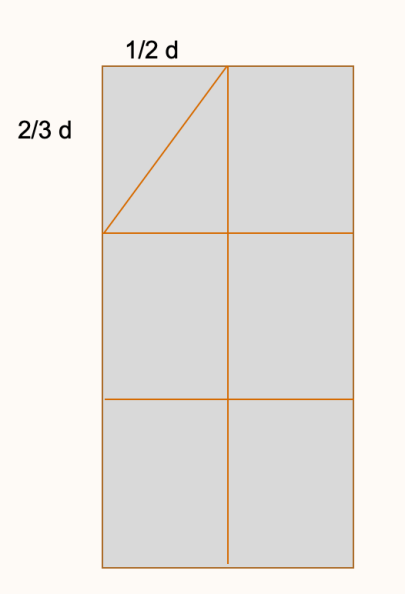
对于左侧的点，只有两点距离小于d才有合并时减小d的可能，所以右侧点必定在以左侧点为圆心、d为半径的圆内（上左图所示），其余点为无效点（上右图所示）。

又因为右侧区域内任意两点距离最小值为d2 >= d, 所以圆内两点距离大于等于半径。

因为右侧点一定位于圆心右侧，最大可覆盖区域为右半圆，又因为两点距离大于半径，所以最多可能存在5个点满足条件（上左图所示）。

在计算机中对点坐标的排序为x或y坐标排序，难以对圆形区域进行判断，所以将半圆扩展为矩形便于计算机运算。改矩形范围内最多可能存在6个点满足条件（上左图所示）。



将矩形分为6个等大的2/3d\*1/2d的小矩形，假设存在7个点，则必定有一个矩形内有两个点。

同一小矩形内两大距离最大值为对角线，即0.8333d < d 与题意不符，所以不能有7个点。而6个点的情况如上右图所示。

***综上所述，***右侧点位于以左侧点为中心，上下高度d，右侧宽度d的d\*2d的范围内。且右侧点的个数存在上限。

所以在查找过程中，只需要找到第一个属于左侧点对应矩形范围内的点，并之多向上查找6次，即可完成查询。

依次遍历左带中的点，并自下而上的遍历右带直到遇到第一个符合条件的点。

1. 伪代码：

for i = 1 to left.size

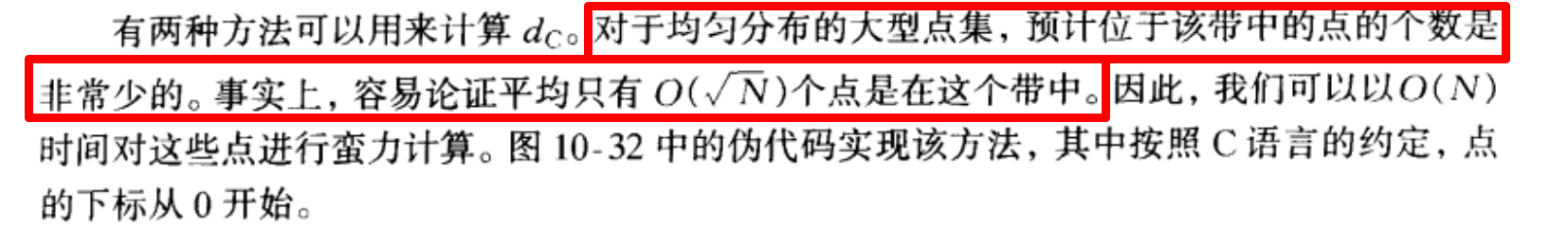
for j = 1 to right.size

if right[j] 在相应的矩形内

for k = j to j + 6 and k < right.size

ans = min(ans, dis(left[i], right[j]))

1. 复杂度分析：

-----《数据结构与算法分析-C语言描述》P280

**递推公式：**

平均情况 T(n) = 2T(n/2) + (√n)^2

总体时间复杂度O（nlogn)

最坏情况 T(n) = 2T(n/2) + (n)^2

总体时间复杂度O（n^2)

1. 上述思路代码实现：

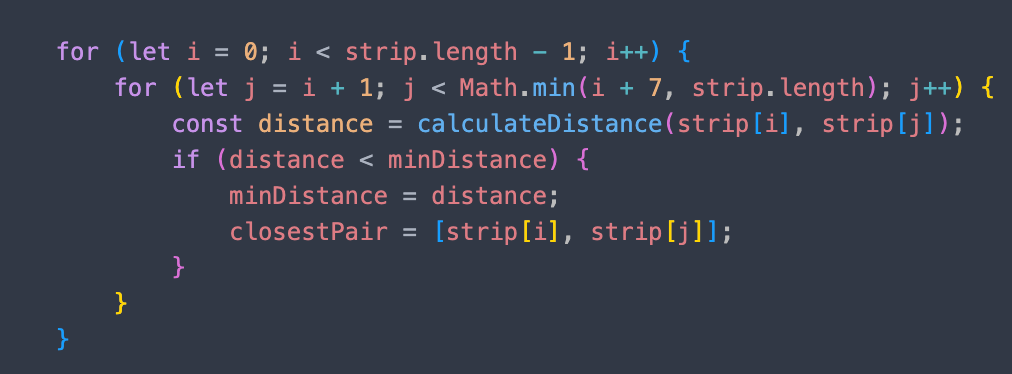
首先，对点集按照x坐标进行排序，得到 sortedPoints。然后，对点集按照y坐标进行排序，得到 sortedY。

递归函数 closestPairRec：这个内部函数是分治算法的核心，它递归地寻找最近点对。如果点集的大小 n 小于等于3，由于点集很小，直接使用暴力方法bruteForceClosestPair WithinStrip来找到最近点对。否则，找到中点 midPoint，并将点集分为左右两部分 leftX 和 rightX。同时，根据中点的x坐标，将 sortedY 分为 leftY 和 rightY。对左右两部分分别递归调用 closestPairRec 函数，找到左右两边的最近点对 closestLeft 和 closestRight。





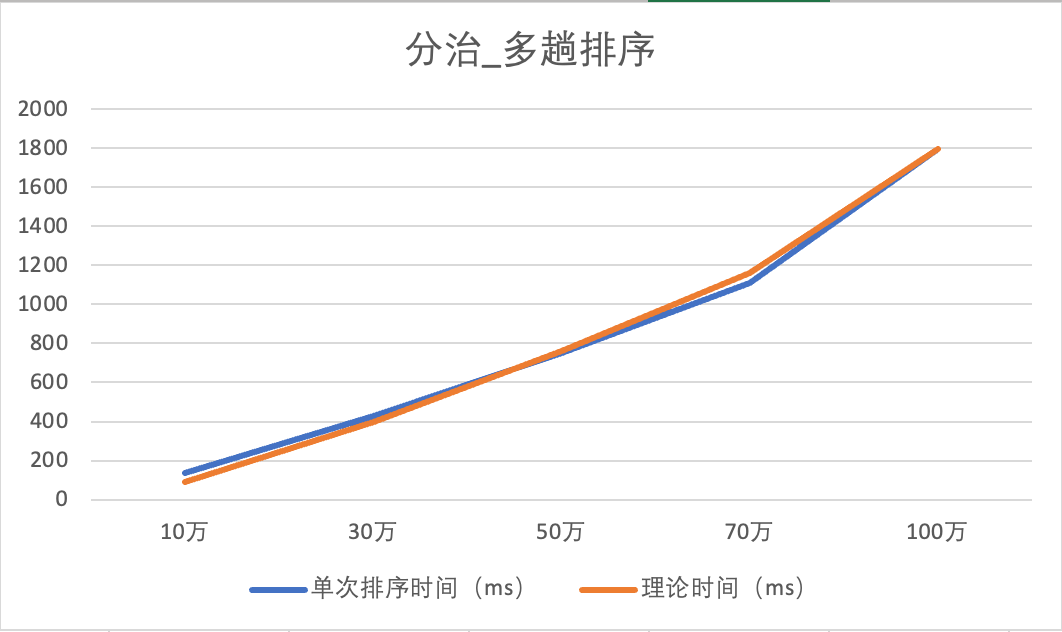
比较左右两边的最近点对，取距离更近的一个作为当前的最近点对 closestPair，并更新最小距离 minDistance。创建一个条带 strip，包含所有x坐标与中点x坐标距离小于 minDistance 的点。检查条带内的点对：在条带 strip 内使用暴力方法，遍历所有点对，计算它们之间的距离，并更新最小距离和最近点对。



1. 数据分析：

最终结果如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 10万 | 30万 | 50万 | 70万 | 100万 |
| 单次时间（ms） | 136.25 | 429.031 | 748.738 | 1111 | 1797 |
| 理论时间（ms） | 89.85 | 398.158 | 763.262 | 1160.474 | 1797 |



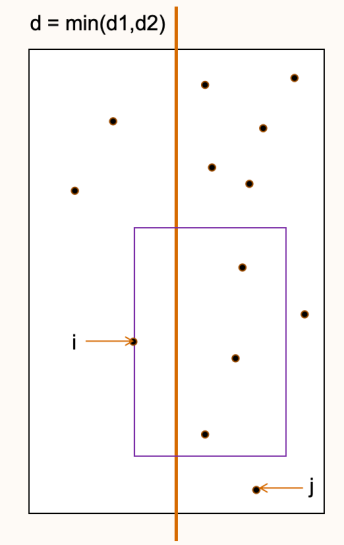
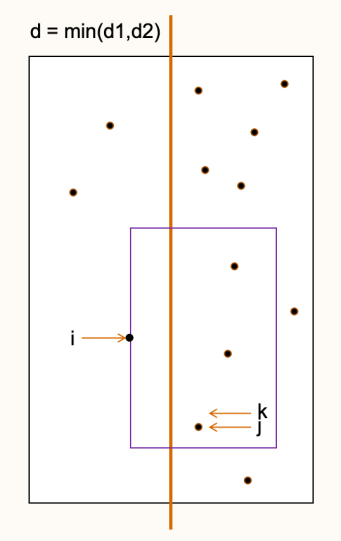
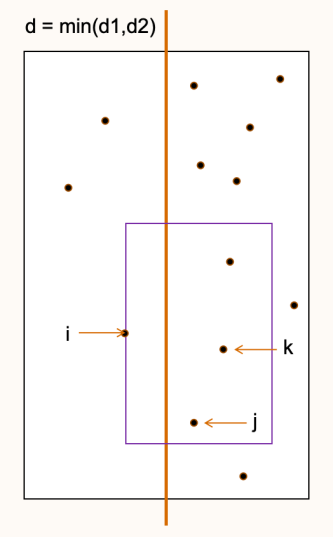
图像上符合O(nlogn) 曲线，并且理论值与实际值误差较小。

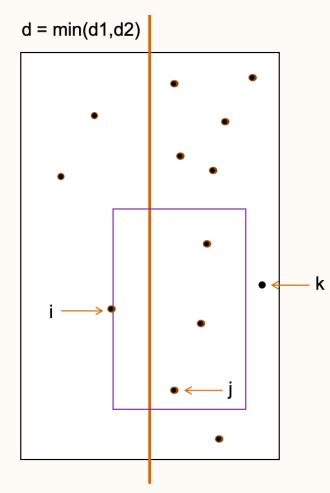
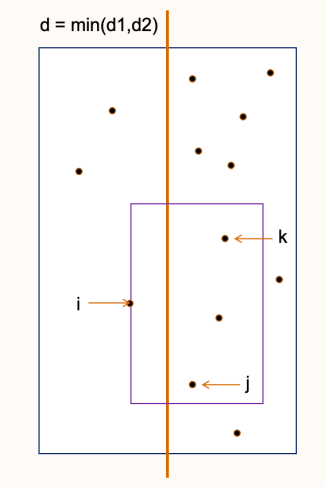
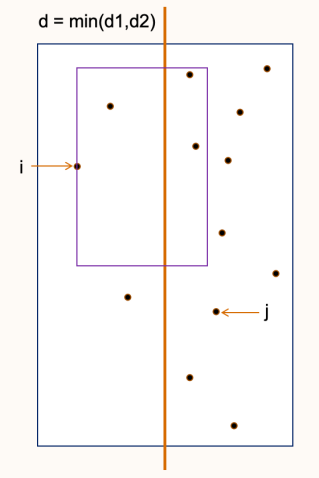
显然对于最差情况仍然为O(n^2)，需要改进。

1. 分治——一趟查询
2. 算法原理：

此算法在上一部分的基础上进行了部分改进，达到了线性的查找效率。

方法是对于左右侧的点都自下而上进行查找。因为矩形区域是固定的，所以随着左侧点的y坐标增大，矩形的最低点也是逐渐增大，右侧符合条件的第一个点的y也会逐渐增大，并不需要返回重新查找。

1. 伪代码：

j = 0

for i = 1 to left.size

while j < right.size and right[j].y < left[i].y - d

j += 1

for k = j to j + 6 and right[k].y < left[i].y + d

ans = min(ans, dis(left[i], right[j]))

1. 复杂度分析：

在此过程中，只需要对左右带中的点进行依次遍历，即可找到最短距离，最多比较6n次即可，达到了线性效率。

**递推公式：**

T(n) = 2T(n/2) + n

总体时间复杂度O（nlogn)

1. 上述思路代码实现：

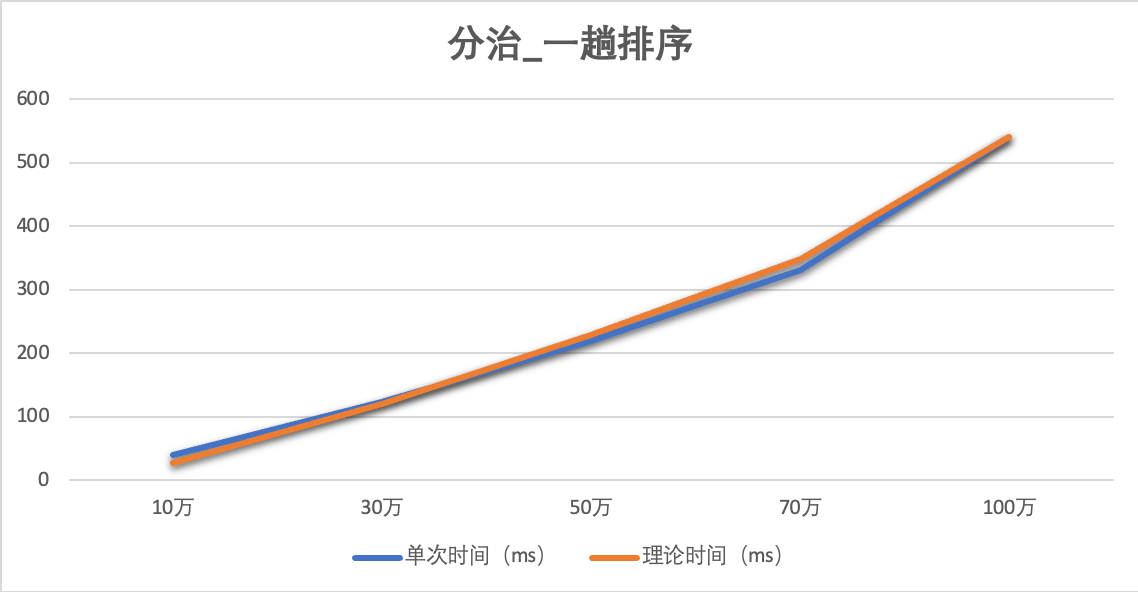
closestPairOnePass：这个函数尝试在一次遍历中找到最近点对。它首先对点集按照x坐标进行排序。然后，它使用两个指针（leftIndex 和 rightIndex）来遍历排序后的点集。在遍历过程中，它计算两个指针所指的点之间的距离，并更新最小距离和最近点对。如果右指针与左指针的距离大于1且当前点的y坐标与左指针所指点的y坐标之差大于最小距离，则移动左指针。



1. 数据分析：

最终结果如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 10万 | 30万 | 50万 | 70万 | 100万 |
| 单次时间（ms） | 39.967 | 123.128 | 219.903 | 331.7 | 540.15 |
| 理论时间（ms） | 27.008 | 119.680 | 229.425 | 348.820 | 540.15 |



图像上符合O(nlogn) 曲线，并且理论值与实际值误差较小。

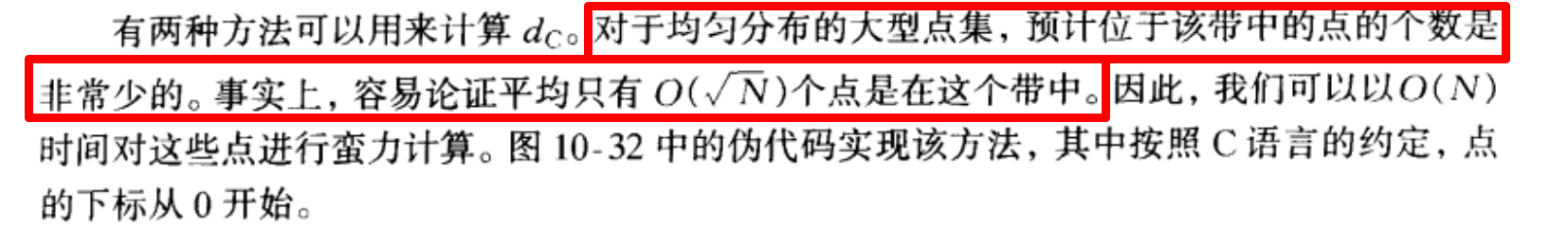
1. 问题思考：

上述合并的方法是利用自下而上的遍历左右带中的点集，所以需要对带中的点以y坐标进行排序。

前提条件：左右区域内的点是依据y坐标进行排序的。

实际条件：左右区域内的点是依据x坐标进行排序的。（获取中轴线时，已排序）

所以每次递归的过程之中都需要对y进行排序。

-----《数据结构与算法分析-C语言描述》P280

**递推公式：**

平均情况 T(n) = 2T(n/2) + √n∙log√n < 2T(n/2) + n

合并效率小于O(nlogn)，又因为一开始对x排序，O(nlogn)为时间效率下限。

总体时间复杂度O(nlogn)

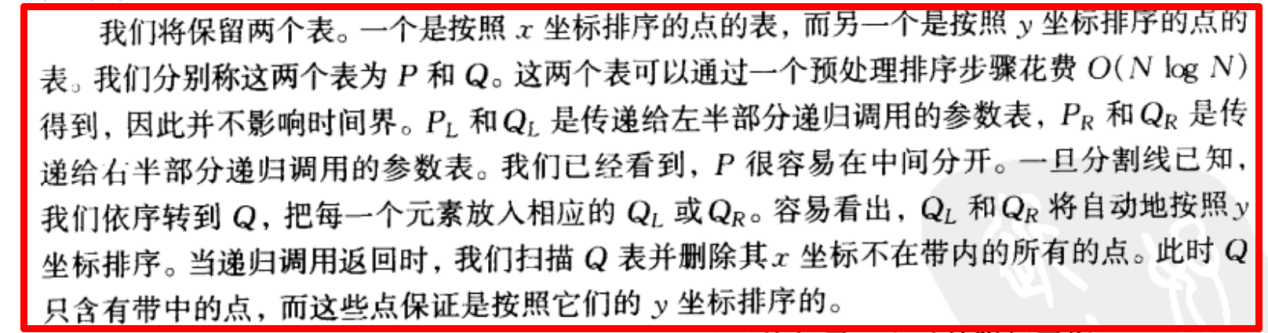
最坏情况 T(n) = 2T(n/2) + nlogn

总体时间复杂度O(nlognlogn)

虽然最坏情况下为O(nlognlogn)，但相较于O(n^2)有较大提升。

为了解决这一问题，引入以下方法。

解决方案：

 -----《数据结构与算法分析-C语言描述》P281

**递推公式：**

T(n) = 2T(n/2) + n

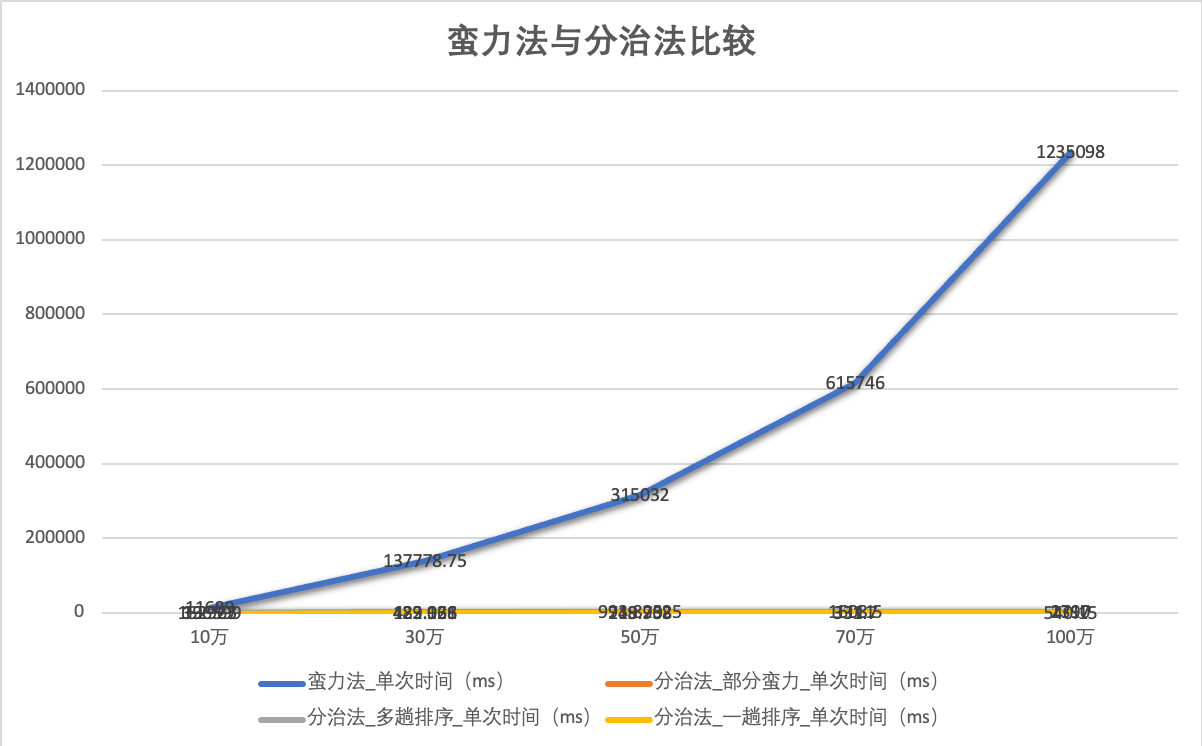
总体时间复杂度O(nlogn)

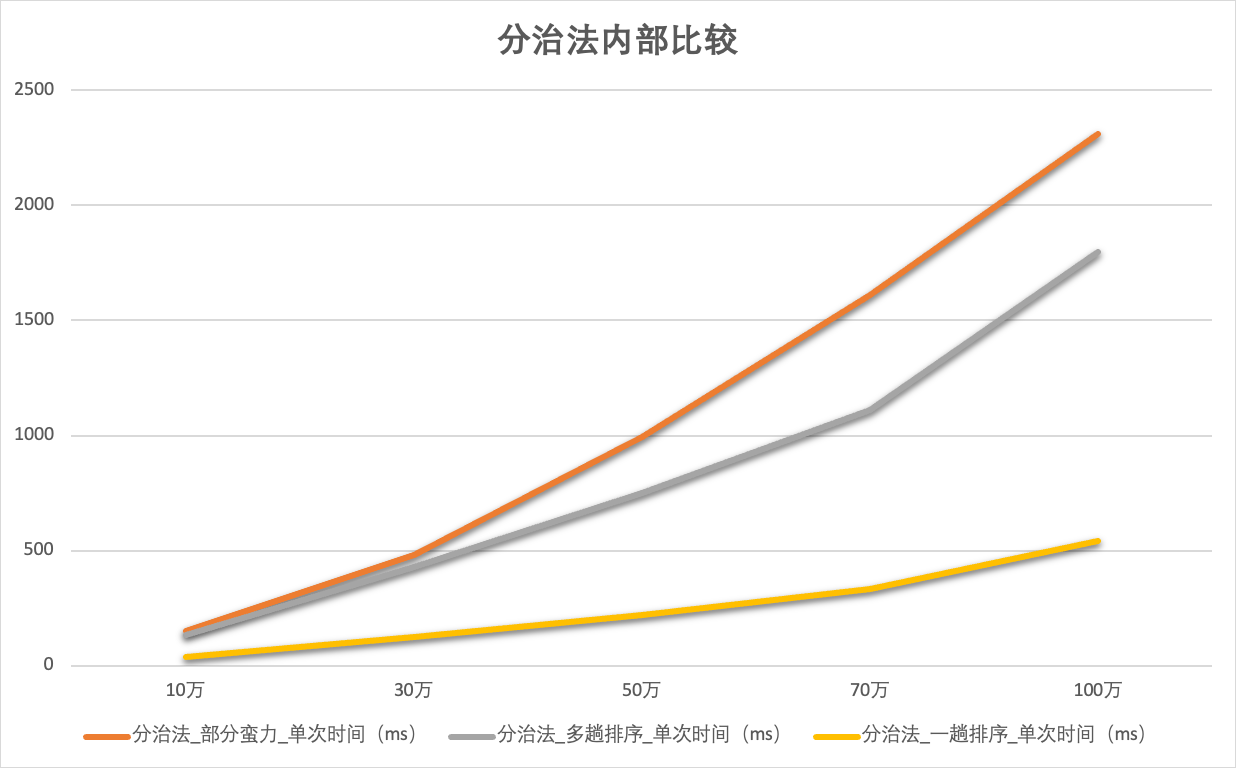
1. 综合分析

（1）数据分析：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 10万 | 30万 | 50万 | 70万 | 100万 |
| 蛮力法\_  单次时间（ms） | 11689 | 137778.75 | 315032 | 615746 | 1235098 |
| 分治法\_  部分蛮力\_  单次时间（ms） | 152.599 | 482.966 | 993.82525 | 1608.5 | 2310 |
| 分治法\_  多趟排序\_  单次时间（ms） | 136.25 | 429.031 | 748.738 | 1111 | 1797 |
| 分治法\_  一趟排序\_  单次时间（ms） | 39.967 | 123.128 | 219.903 | 331.7 | 540.15 |

（2）对比图像展示：





（3）结论：

显然，分治法的时间效率是要远远优于蛮力法。每一次的优化，都对于时间效率有着一定幅度的提升。

## 经验总结

通过本次实验，我深入理解了分治法的核心思想，即通过将大问题分解为若干个小规模子问题，逐个解决并合并结果，从而有效解决复杂问题。在求解最近点对问题的过程中，我掌握了预处理数据、递归求解子问题以及合并结果的关键步骤，进一步加深了对分治法的应用和理解。

在优化算法方面，我通过仔细分析合并过程，成功降低了算法的时间复杂度，从原本的O(n^2)提升到了O(nlogn)，显著提升了算法的性能。这一优化过程不仅提高了算法的效率，还锻炼了我的算法设计和分析能力。

为了验证分治法的正确性和效率优势，我设计了一系列实验，通过随机生成数据并使用蛮力法和分治法进行求解。实验结果显示，分治法不仅在时间效率上明显优于蛮力法，而且能够准确找到最近点对。这一结果验证了分治法的正确性和高效性，让我对分治法有了更深入的认识。

在实验过程中，我不断思考和优化代码细节，通过引入额外的数据结构进一步提升算法效率。这些实践经历让我深刻认识到，在实现算法时，细节优化和引入辅助数据结构对于提升算法性能至关重要。

综上所述，本次实验不仅让我掌握了分治法的应用技巧，还锻炼了我的算法设计和优化能力。通过不断实践和思考，我对分治法的理解更加深入，对算法实现和优化也有了更深刻的认识。

|  |
| --- |
| 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。