深 圳 大 学 实 验 报 告

课程名称：算法设计与分析

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：计算机科学与技术

报告人：张 跃 学号：**2015160180** 班级：**5**

同组人：无

指导教师：杨 烜

实验时间：**2017/10/01——2017/10/12**

实验报告提交时间：

教务处制

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. 实验步骤与结果

（一）实验总体思路：

1. 使用随机数生成函数生成所需规模的随机点数据。**使用 C++ 标准库容器 set（集合）储存生成的随机点数据。**保证数据中没有重复的点。
2. 对于同一组随机点数据，分别使用蛮力法和分治法找出最近点对。使用获取执行函数执行之前和之后的时钟时间，通过时间相减得到排序算法运行时间，用来衡量算法性能。**本次实验时间单位均为秒（s）。**
3. 使用主函数中的循环改变数据规模和随机种子值。对于测试数据规模 N = 10w，20w，30w，40w，50w，60w，70w，80w，90w，100w，每个数据规模测试 20 次，统计出平均值。将得到的数据制表，画出折线图。使用规模为中位数的实测值计算理论值，同样画出折线图，比较实测值与理论值曲线。
4. 为了控制无关因素，本次实验算法在**服务器环境**上运行，排除减少硬件的无关负载的影响，使实验结果更有说服力。
5. **使用 Python 改写原先 C++版本的分治算法，并实现算法过程的可视化，展示算法执行过程的动画。**

（二）实验过程解析：

1. 使用随机数生成函数生成所需规模的随机点数据。并**使用 C++ 标准库容器 set（集合）储存生成的随机点数据。**保证数据中没有重复的点。
   代码实现如下，如图 1 和 2 所示。

```
/*
 * 定义二维点类和全局变量
 */
class Point
{
  public:
    double x, y;
    Point() : x(0), y(0){};
    Point(double x_, double y_) : x(x_), y(y_){};
    bool operator<(const struct Point &right) const // 重载<运算符
    {
        if (this->x == right.x && this->y == right.y) // 根据坐标去重
            return false;
        else
        {
            if (this->x != right.x)
            {
                return this->x < right.x; // 按x轴升序排列
            }
            else
            {
                return this->y < right.y; // 若x轴相同按y轴升序排列
            }
        }
    }
};
```

图 1　定义 Point 类

```
set<Point> points_set;
n = counter * 10;
p = new Point[n];
p1 = new Point[n];

srand(counter); //设置随机种子值
for (i = 0; i < n; i++)
{
    points_set.insert(Point(rand(), rand()));
}
int i_ = 0;
for (auto it : points_set)
{
    p[i_].x = it.x;
    p[i_].y = it.y;
    p1[i_].x = it.x;
    p1[i_].y = it.y;
    i_++;
}
for (i = 0; i < n; i++)
{
    cout << p[i].x << " , " << p[i].y << endl;
}
```

图 2　使用 set 容器存储生成的随机点

2. 分别实现蛮力法和分治法找最近点对，蛮力法适用暴力枚举，对于待求解的点集中的每一个点，计算与其他点的距离，在所有数据中选取最小值。核心过程代码如下。

```
for (i = 0; i < n; i++)
    for (j = i + 1; j < n; j++)
    {
        dis = sqrt((p[i].x - p[j].x) * (p[i].x - p[j].x) + (p[i].y - p[j].y) * (p[i].y - p[j].y));
        if (dis < min)
        {
            x1 = p[i].x;
            y1 = p[i].y;
            x2 = p[j].x;
            y2 = p[j].y;
            min = dis;
        }
    }
```

图 3 嵌套循环找最近点对

总体代码实现如图 4 所示。

```
/*
 * 蛮力法求最近点对
 */
double brute_force(int n)
{
    double dis;
    int i, j;

    double min = RAND_MAX;
    int x1, x2, y1, y2;

    for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++)
        {
            dis = sqrt((p[i].x - p[j].x) * (p[i].x - p[j].x) + (p[i].y - p[j].y) * (p[i].y - p[j].y));
            if (dis < min)
            {
                x1 = p[i].x;
                y1 = p[i].y;
                x2 = p[j].x;
                y2 = p[j].y;
                min = dis;
            }
        }

    //记录
    pointx1 = x1;
    pointy1 = y1;
    pointx2 = x2;
    pointy2 = y2;

    return min;
}
```

图 4　蛮力法 C++实现

分治法使用递归的思想，（1）在随机点数|S|<=3 时，分情况计算返回距离，如图所示。

```
// n = 1, distance = ∞
if (low == high)
    return RAND_MAX;
// n = 2, 直接返回两点距离
if (low + 1 == high)
    return dis(p[low], p[high]);
// n = 3, 蛮力法查找
if (low + 2 == high)
    return min(min(dis(p[low], p[low + 1]), dis(p[low], p[high])), dis(p[low + 1], p[high]));
```

图 5　分情况计算

（2）分：在随机点数|S|>3 时，将平面点集 S 分割成为大小大致相等的两个子集 SL 和 SR，选取一个垂直线 L 作为分割直线，递归调用，分别求出 SL 和 SR 中的最短距离为 dl 和 dr。在 dl 和 dr 之间选出较小的距离 d。代码如图所示。

```
int mid = (low + high) / 2;
double d = min(divide_and_conquer(low, mid), divide_and_conquer(mid + 1, high));

// 计算得到的矩形区域里面最短距离的点对，并和之前递归调用得到的最短距离相比较，得出最短距离
int i, j, t = 0;
for (i = low; i <= high; i++)
    if (p[i].x >= p[mid].x - d && p[i].x <= p[mid].x + d)
        a[t++] = i;
```

（3）治：但这个距离不一定是整体平面中的最小距离：线 L 可能将一对点分割在两个子平面里。考虑边界为距离 L 为 d 的两侧的区域 S’。如果存在一对最近点被线 L 分割，那么这对点一定在区域 S’里。将区域 S’里的点按照 y 轴排序，得到数组 Y’。对于其中一个点 p1，如存在上述的最近点，只考虑 y 比它大的那部分（这样一对点不会重复枚举）。即若 p1 点坐标为(x ， y)，那么要考虑的点的横坐标 x-d< x′ < x+d，纵坐标 y ≤ y′ ≤ y+d。可证矩形内至多有 6 个可能点（8 个点，其中两对重复的点，根据鸽笼定理）。这样只需要检查数组 Y’中每个点之后的 6 个点（因为不能有重复点）。至此算法完成递归调用。代码实现如

下图所示。

```
for (i = 0; i < t; ++i)
{
    int one = 0;
    for (j = i + 1; j < t; ++j)
    {
        if (one < 7)
        {
            if (p[a[j]].y - p[a[i]].y <= d)
                d = min(d, dis(p[a[i]], p[a[j]]));
            one++;
        }
        else
        {
            break;
        }
    }
}
```

图 6　根据鸽笼定理找出中间区域的可能点

总体使用 C++实现如下所示。

```
/*
 * 分治法递归求最小距离
 */
double divide_and_conquer(int low, int high)
{
    // n = 1, distance = ∞
    if (low == high)
        return RAND_MAX;
    // n = 2, 直接返回两点距离
    if (low + 1 == high)
        return dis(p[low], p[high]);
    // n = 3, 蛮力法查找
    if (low + 2 == high)
        return min(min(dis(p[low], p[low + 1]), dis(p[low], p[high])), dis(p[low + 1], p[high]));


    int mid = (low + high) / 2;
    double d = min(divide_and_conquer(low, mid), divide_and_conquer(mid + 1, high));

    // 计算得到的矩形区域里面最短距离的点对，并和之前递归调用得到的最短距离相比较，得出最短距离
    int i, j, t = 0;
    for (i = low; i <= high; i++)
        if (p[i].x >= p[mid].x - d && p[i].x <= p[mid].x + d)
            a[t++] = i;

    // 快速排序，线性效率
    sort(a, a + t, compy);

    for (i = 0; i < t; ++i)
    {
        int one = 0;
        for (j = i + 1; j < t; ++j)
        {
            if (one < 7)
            {
                if (p[a[j]].y - p[a[i]].y <= d)
                    d = min(d, dis(p[a[i]], p[a[j]]));
                one++;
            }
            else
            {
                break;
            }
        }
    }

    return d;
}
```

图 7　分治法求最近点对

3. 使用主函数中的循环改变数据规模和随机种子值。对于测试数据规模 N = 10w，20w，30w，40w，50w，60w，70w，80w，90w，100w，每个数据规模测试 20 次，统计出平均值。C++实现如下图所示。

```
for (int counter = 1; counter <= 10; counter++)
{
    set<Point> points_set;
    n = counter * 10;
    p = new Point[n];
    p1 = new Point[n];

    srand(counter); //设置随机种子值
    for (i = 0; i < n; i++)
    {…
    }
    int i_ = 0;
    for (auto it : points_set)
    {…
    }
    for (i = 0; i < n; i++)
    {…
    }

    cout << "Generating random points done." << endl;
    cout << "Data Scale:     " << n << endl;
    cout << "Size of set:    " << points_set.size() << endl;

    // 蛮力法
    cout << "Brute Force..." << endl;

    time3 = clock();
    cout << "Distance: " << brute_force(n) << endl;
    time4 = clock() - time3;
    cout << "Time cost: " << time4 << "ms" << endl;
    cout << "The Points: "
    << "(" << pointx1 << "," << pointy1 << ")  "
    << "(" << pointx2 << "," << pointy2 << ")" << endl;
    // 分治法
    cout << "Divide and Conquer..." << endl;

    time1 = clock();

    // 线性效率的快速排序
    sort(p, p + n, compx);
    cout << "Distance: " << divide_and_conquer(0, n - 1) << endl;
    time2 = clock() - time1;
    cout << "Time cost: " << time2 << "ms" << endl;

    cout << "The Points: "
         << "(" << pointx1 << "," << pointy1 << ")  "
         << "(" << pointx2 << "," << pointy2 << ")" << endl;
}
return 0;
```

图 8　main 函数实现

4. 分别统计蛮力法和分治法的数据，计算理论值。

在服务器上运行程序，记录运行时间，统计得到下表：

N	100000	200000	300000	400000	500000
Time cost	106.824	231.452	370.177	495.586	651.391
N	600000	700000	800000	900000	1000000
Time cost	796.774	928.553	1065.03	1228.58	1387.72

图 9　分治法实测值表格

以 5w 为基准，计算理论值，得到下表：

N	100000	200000	300000	400000	500000
Time cost	114.2997769	242.3626183	375.6202424	512.2513657	651.391
N	600000	700000	800000	900000	1000000
Time cost	792.5296785	935.3307626	1079.55499	1225.023463	1371.597323

图 10　分治法理论值表格

根据以上表格绘制折线图，如下图所示。

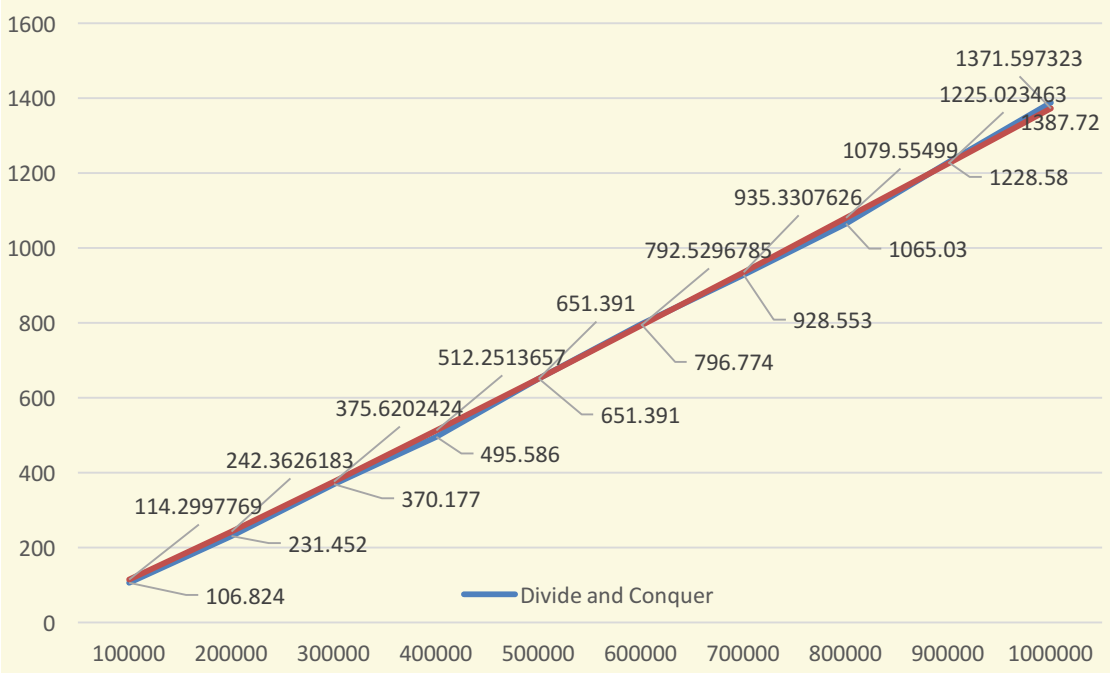

图 11　分治法折线图

可见分治法呈线性效率。

统计蛮力法数据并计算理论如下表。

N	100000	200000	300000	400000	500000
Time cost	71.9043	288236	653193	1163290	1819830
N	600000	700000	800000	900000	1000000
Time cost	2620970	3567940	4661720	5900210	7285740

图 12　蛮力法实测值表格

以 5w 为基准，计算理论值，得到下表：

N	100000	200000	300000	400000	500000
Time cost	114.2997769	242.3626183	375.6202424	512.2513657	651.391
N	600000	700000	800000	900000	1000000
Time cost	792.5296785	935.3307626	1079.55499	1225.023463	1371.597323

图 13　蛮力法理论值表格

将实测数据和理论数据绘制成图表：

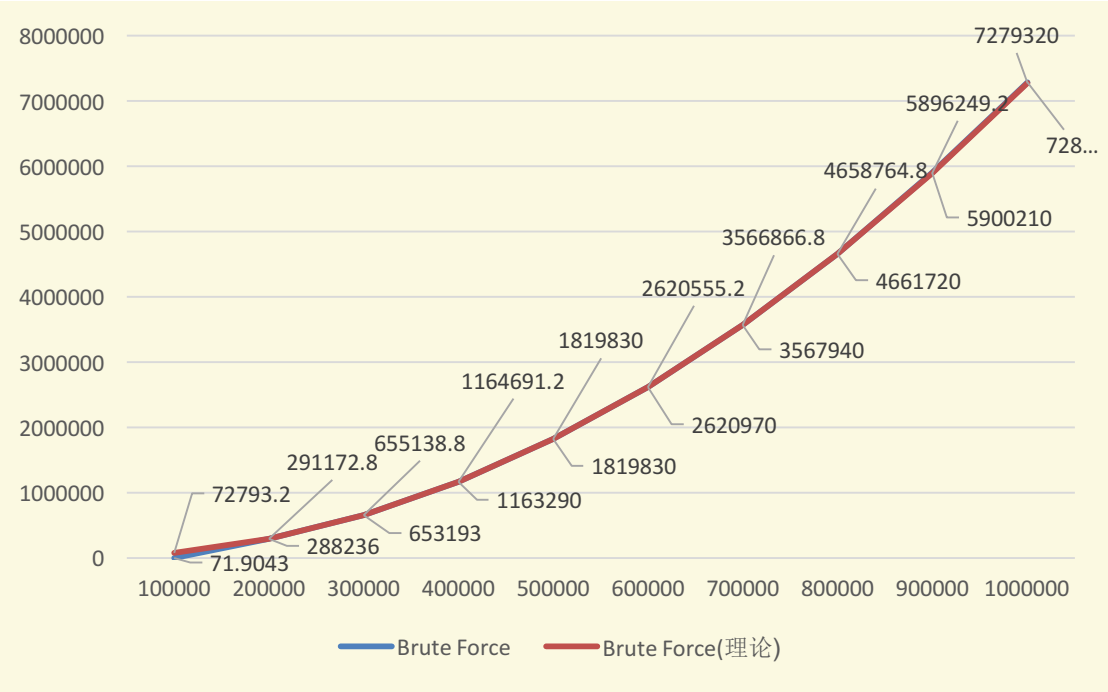

图 14　蛮力法折线图

将分治法和蛮力法的实测数据绘制在同一张折线图中，如下图所示。

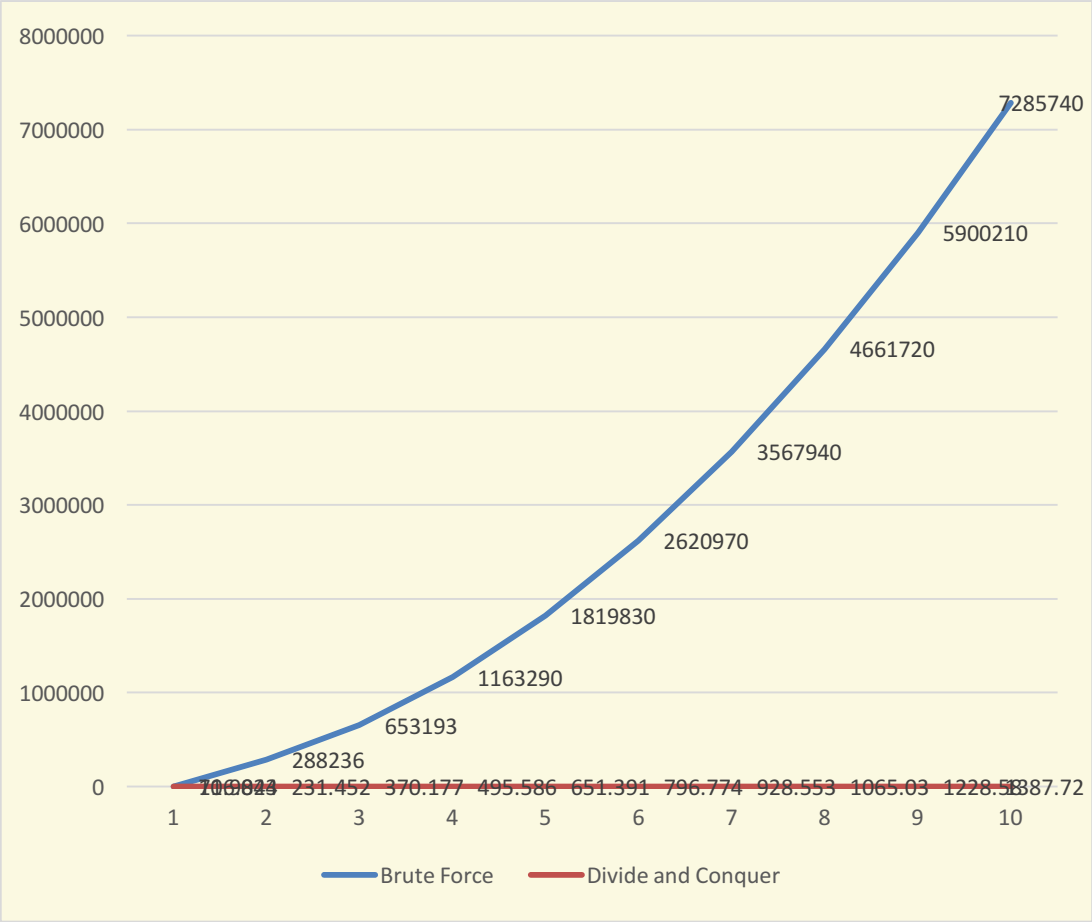

图 15　两种方法的效率比较

可见分治法效率远远高于蛮力法。

5. 使用 Python 改写以上分治法的 C++实现，并实现算法可视化动画。具体实现和注释见源代码文件。运行截图如下

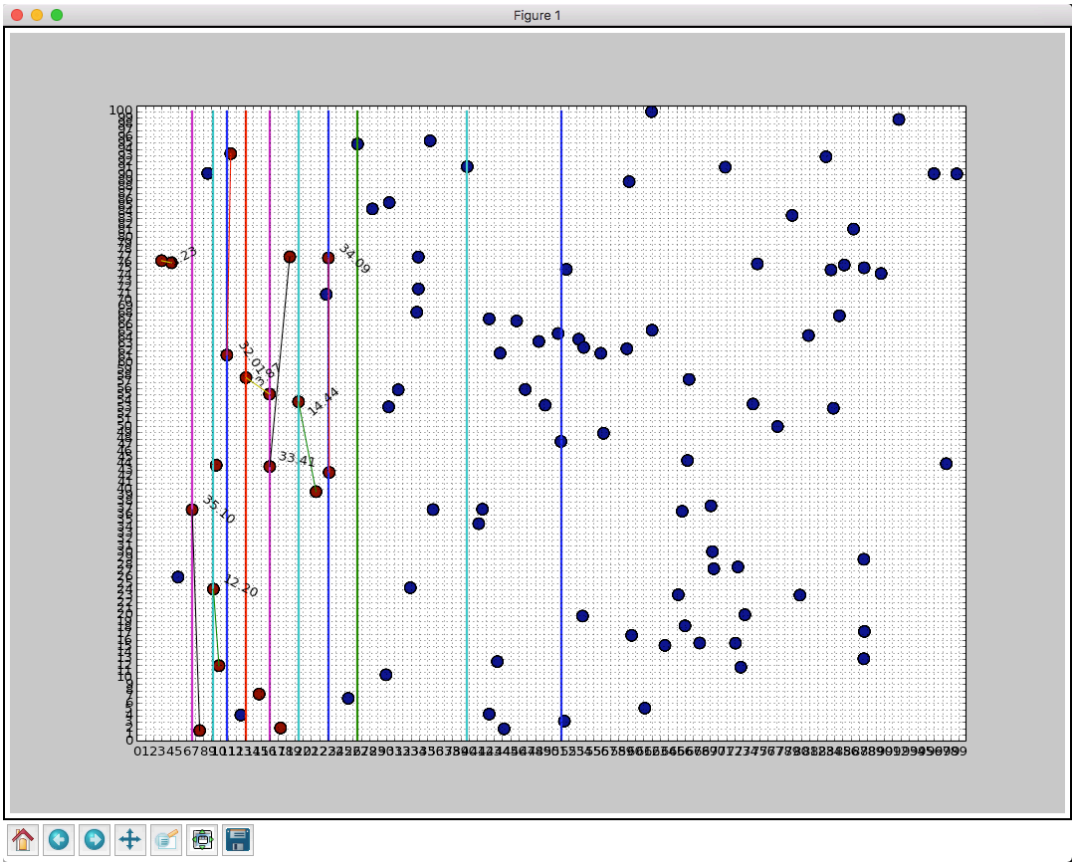

图 16　分治法算法可视化

运行过程输出中垂线两侧的点坐标，运行结束输出最短距离。如图所示。

```
Middle: (15.3240590488857 44, 9.91578617459714)
Y_RIGHT: [(19.248615206239947, 1.4243756575560658), (19.575951707518733, 5.993610903715268), (15.324059048885744, 9.91578617459714), (19.970070023972166, 17.538588936
000096), (15.483731158691578, 17.91180675981772)]
Y_LEFT : [(12.41015153821314, 5.234709655687011), (9.278505650578923, 5.523734503217474), (12.5308854407679, 10.054992032554278), (12.68903944101141, 10.6975640212901
43), (13.986327992869484, 16.590055630488457)]
Middle: (12.5308854407679, 10.054992032554278)
Y_RIGHT: [(12.5308854407679, 10.054992032554278), (12.68903944101141, 10.697564021290143), (13.986327992869484, 16.590055630488457)]
Y_LEFT : [(12.41015153821314, 5.234709655687011), (9.278505650578923, 5.523734503217474)]
min_dis: 0.661748780506
Middle: (19.248615206239947, 1.4243756575560658)
Y_RIGHT: [(19.248615206239947, 1.4243756575560658), (19.575951707518733, 5.993610903715268), (19.970070023972166, 17.538588936000096)]
Y_LEFT : [(15.324059048885744, 9.91578617459714), (15.483731158691578, 17.91180675981772)]
min_dis: 4.58094530854
min_dis: 0.661748780506
min_dis: 0.147892248931
Minimum distance between two points is 0.147892248931
```

图 17　算法可视化输出结果

四． 实验心得

本次实验花费了我很多的时间。但通过对两种算法效率的分析和测试比较，以及对分治算法的可视化动画实现，加深了我对分治算法的认识。

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。