算法设计与分析

寻找平面上距离最短的点实验报告

一、算法设计

1、O(n^2)算法

遍历每个点与其他所有点的距离，选择最近的距离

```
for i := 0 to N-1
    for j:=i to N-1
        if distance(points[i], points[j]) < minDistance
            minDistance := distance(points[i], points[j])
    end
end
```

2、O(nlgn)算法

使用分治法的思想，将平面分为两个部分，分解为两个子问题，在进行合并，算法的思路是：

（1） 将所有点按照 x 值排序，并划分为（start,(start+end)/2）和（(start+end)/2,end）两个部分

（2） 使用分治法计算这两个部分的最短距离，记为 minHalf

（3） 考虑最短的两个点分别在这两个区域的情况，以（1）中划分点的中线 half 为基础，分别取 half-minHalf 和 half+minHalf 两条边界，将中间部分的点存到临时空间。

（4） 将临时空间的点按 y 轴排序，从小到大开始检查与临时空间其余每个点的距离，如果检查点数超过 7 或两点的 y 轴距离大于最小距离则终止。

（5） 比较(4)中的最小距离 minMiddleDistance 与 minHalf，选择 min(minMiddleDistance, minHalf)为最终的最小距离。

```
sort(points, compareX)
function minDistance(start, end):
    if end-start==0
        return INF
    else if end-start==1
        return distance(points[start], points[end])
    minLeft := minDistance(start, (start+end)/2)
    minRigth := minDistance(start, (start+end)/2)
    minHalf := min(minLeft, minRight)
    for i:=start to end
        if abs(points[i].x-points[half].x) < minHalf
```

```
        then middlePoints.add(points[i])
    end
    sort(middlePoints, compareY)
    for i := 0 to N-1
        for j:=i+1 to i+7
            if middlePoints[j].y-middlePoints[i].y > minHalf
            then break
            if distance(points[i], points[j]) < minDistance
            then minDistance := distance(points[i], points[j])
        end
    end
    return min(minDistance, minHalf)
```

二、实验设计

1、实验环境

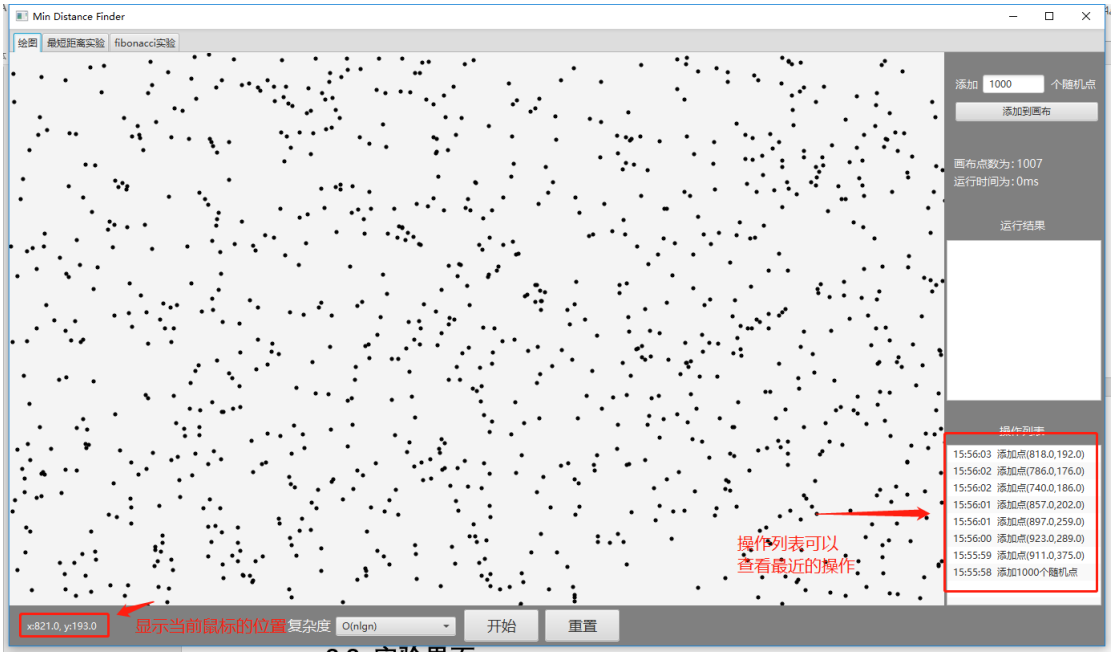
Java 1.8
Windows 10 64 位，　CPU i7-8700

2、实验界面

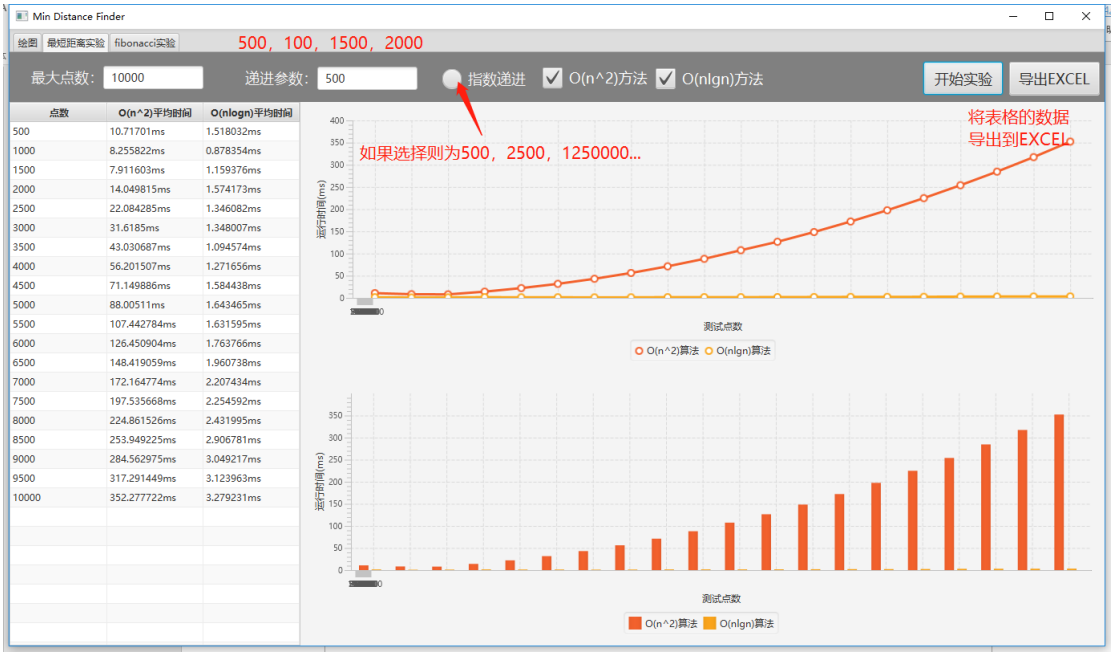
2.1 画布界面

画布界面可以用鼠标在画布上选择随机点，也可选添加随机生成的点。选择复杂度并计算后会显示最小两点距离的连线。

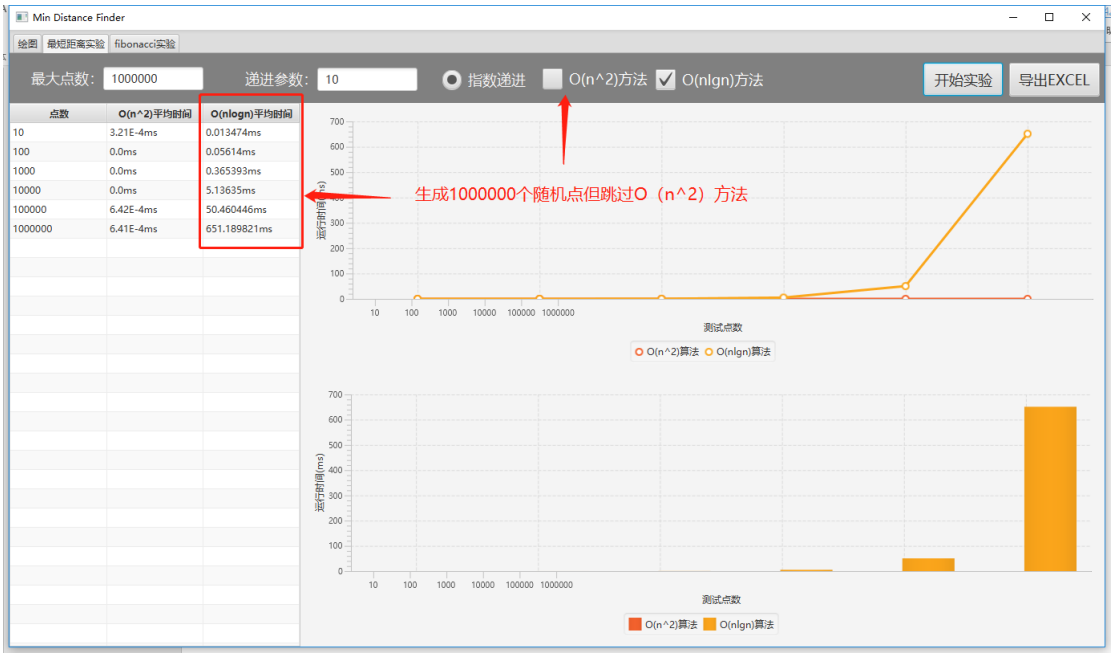
生成随机点与鼠标画点可以以任意顺序进行



2.2 实验界面



实验界面可以比较不同方法的执行时间，最大点数限制了递进的最大范围，选择指数递进则将递进方式由加法变为指数，由于 $O(n^2)$方法后期太慢影响实验，也可以只对 O(nlgn)方法进行实验。



可以将实验的结果导出到 Excel 以备后续分析

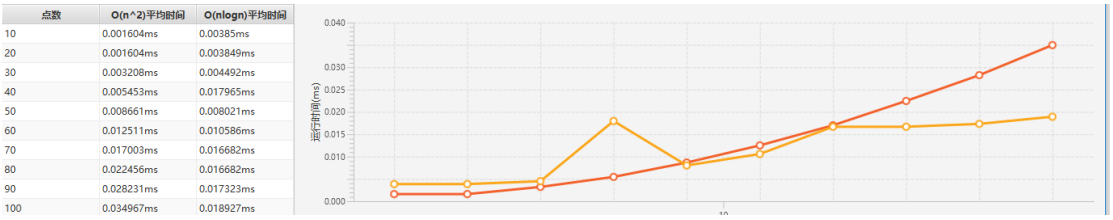
	A	B	C
1	点数	O(n^2)平均时间	O(nlogn)平均时间
2	500	0.878354ms	0.17195ms
3	1000	3.511171ms	0.345503ms
4	1500	7.921227ms	0.523227ms
5	2000	14.318647ms	0.586104ms
6	2500	22.200736ms	0.708008ms
7	3000	31.829909ms	0.846595ms
8	3500	43.676781ms	1.025602ms
9	4000	56.433446ms	1.177983ms
10	4500	72.538635ms	1.387466ms
11	5000	87.933892ms	1.529902ms
12	5500	106.275067ms	1.705059ms
13	6000	126.500308ms	1.839154ms
14	6500	148.465254ms	2.027465ms
15	7000	172.161245ms	2.207754ms
16	7500	197.6765ms	2.371043ms
17	8000	224.887832ms	5.199548ms
18	8500	254.040012ms	2.844545ms
19	9000	284.758665ms	3.123963ms
20	9500	317.056622ms	3.233356ms
21	10000	351.317243ms	3.383491ms

三、实验结果

1、运行时间的对比

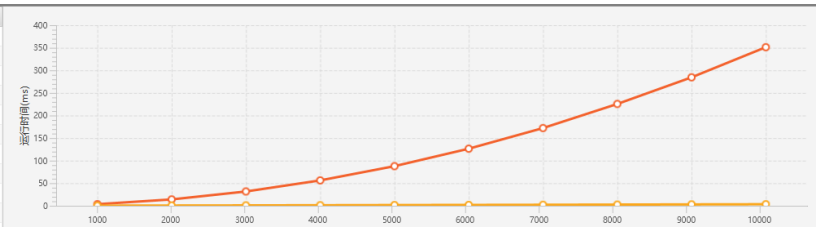
点数	O(n^2)时间	O(nlogn)时间
10	0.002567ms	0.005453ms
20	0.001924ms	0.003529ms
30	0.003208ms	0.006095ms
40	0.005774ms	0.018607ms
50	0.008983ms	0.008982ms
60	0.012512ms	0.011228ms
70	0.017003ms	0.012832ms
80	0.022456ms	0.014757ms
90	0.028551ms	0.017002ms
100	0.034967ms	0.017965ms
1000	3.529136ms	0.26434ms
2000	14.092802ms	0.534455ms
3000	31.672074ms	0.854936ms
4000	56.215301ms	1.146223ms
5000	87.843426ms	1.528618ms
6000	126.651405ms	1.826964ms
7000	173.072962ms	2.169579ms
8000	224.972524ms	2.542029ms
9000	284.700599ms	2.960355ms
10000	351.4924ms	3.378038ms
20000	1405.664839ms	7.522149ms
30000	3164.562103ms	14.743388ms
40000	5624.246037ms	16.780156ms
50000	8788.42799ms	21.447815ms
100000	35274.818314ms	46.101718ms
500000		302.246598ms
1000000		704.853789ms
5000000		6632.300716ms

10 到 100 的折线图为：



1000 到 10000 的折线图为

点数	O(n^2)平均时间	O(nlogn)平均时间
1000	3.507642ms	0.251187ms
2000	14.072913ms	0.541513ms
3000	31.637749ms	0.846595ms
4000	56.21466ms	1.162263ms
5000	87.762584ms	1.555887ms
6000	126.438713ms	1.839154ms
7000	172.163491ms	2.192677ms
8000	225.562156ms	2.577318ms
9000	284.596018ms	3.01072ms
10000	351.367287ms	3.363923ms



2、实验结果

在点数小于 50 时，O(nlgn)算法由于本身的代码复杂性，运行时间大于循环查找暴力求解的方法，在点数为 50~70 之间两种方法运行时长相似，但在这之后分治法方法的运行时间则小于暴力求解方法的时间，两者运行时间的差距不断扩大，在点数超过 100000 时包里求解的方法已经过长而没有统计必要，运行时间大约为 35s，而此时分治法的运行时间只有 46ms，在 5000000 个点的情况下分治法的运行时间为 6632ms，仍然在可以接受的范围。

四、关键代码

分治法的核心代码如下：

```
//在分治开始之前对 points 所有点进行排序
private Comparator<Point> xCompare = Comparator.comparingInt(o -> o.x);
private DistanceResult divideDistance() {
    long start = System.nanoTime();
    points.sort(xCompare);
    DistanceResult result = divideDistance(0, points.size()-1);
    long end = System.nanoTime();
    result.setTime(end-start);
    return result;
}
//分治法主方法
private DistanceResult divideDistance(int start, int end) {
    double minDistance = Double.MAX_VALUE;
    List<Pair<Point,Point>> result = new ArrayList<>();
    if (end - start == 0) {
        return new DistanceResult(result, minDistance);
    }
    if (end - start == 1) {
        result.add(new Pair<>(points.get(start),points.get(end)));
        return new DistanceResult(result, distance(points.get(start),points.get(end)));
    }
    //划分左右
    int half = (end+start)>>1;
```

```
DistanceResult resultLeft = divideDistance(start, half);
DistanceResult resultRight = divideDistance(half, end);
//划分中间分区
double minHalf = Math.min(resultLeft.getMinDistance(), resultRight.getMinDistance());
List<Point> middlePoints = new ArrayList<>();
for (int i = start; i < end; i++) {
    if (Math.abs(points.get(i).x - points.get(half).x) < minHalf)
        middlePoints.add(points.get(i));
}
//计算中间分区的距离最小值
double minMiddleDistance = Double.MAX_VALUE;
List<Pair<Point,Point>> middleResult = new ArrayList<>();
if (middlePoints.size() > 1 ) {
    middlePoints.sort(yCompare);
    for (int i = 0; i < middlePoints.size(); i++) {
        for (int j = i+1; j < middlePoints.size() && j < i+8; j++) {
            if (middlePoints.get(j).y-middlePoints.get(i).y>minHalf) break;
            double distance = distance(middlePoints.get(i), middlePoints.get(j));
            if (distance < minMiddleDistance) {
                middleResult.clear();
                middleResult.add(new Pair<>(middlePoints.get(i), middlePoints.get(j)));
                minMiddleDistance = distance;
            } else if (distance == minMiddleDistance) {
                middleResult.add(new Pair<>(middlePoints.get(i), middlePoints.get(j)));
            }
        }
    }
}
DistanceResult resultMiddle = new DistanceResult(middleResult, minMiddleDistance);
//合并结果
if (resultLeft.getMinDistance() < resultRight.getMinDistance()) {
    minDistance = resultLeft.getMinDistance();
    result = resultLeft.getPoints();
} else if (resultLeft.getMinDistance() == resultRight.getMinDistance()) {
    minDistance = resultLeft.getMinDistance();
    result = resultLeft.getPoints();
    result.removeAll(resultRight.getPoints());
    result.addAll(resultRight.getPoints());
} else {
    minDistance = resultRight.getMinDistance();
    result = resultRight.getPoints();
}
if (minDistance == resultMiddle.getMinDistance()) {
    result.removeAll(resultMiddle.getPoints());
```

```
            result.addAll(resultMiddle.getPoints());
        } else if (minDistance > resultMiddle.getMinDistance()) {
            minDistance = resultMiddle.getMinDistance();
            result = resultMiddle.getPoints();
        }
        return new DistanceResult(result, minDistance);
}
```