ReadMe

Part 1

In this lab I implemented two algorithm, the Naïve algorithm and the Divide-and-Conquer algorithm to find the closest points and the distance between them. The Naïve algorithm is relatively easier compared with the other one. But with one thing I think I should talk about is I didn't call the distance method in the **XYPoint** class, because I think in the Naïve algorithm implementation it is no necessity in calling the method and it would affect the timing part.

While the implementation of Divide-and-Conquer algorithm did cost me much time, almost 24 hours. Firstly I make a method called "**closestPoints**", which is used to find the closest points respectively in the left part and the right part. In this method I also called another method called **" combine"** which is used to find the closest pairs in the divided part. The reason why it took me too much time is I made two big mistake in my implementation. The first is after I divided all of the points into two parts, I stupidly used the Naïve algorithm to solve each part and then combined the divided part. So no matter how big my input is, the running time of the Divide-and-Conquer program needed more time than the Naïve one. It was not RECURSIVE! I simply made the Divided-and-Conquer algorithm into two Naïve algorithm!

The second mistake which was also the biggest bug in my program was I could only check half of the input. Like, when I put ten points, only five of them can be compared, and when goes to the sixth one, a kind of "null" error was displayed. Even with two TAs' kind help, it kept still unfixed. After many times checking and all kinds of searching and asking google, I finally fortunately fixed it with adding another condition to the **if** statement in the part of copying **pointsByY** to **YL** and **YR**. it goes like:

```
if((pointsByY[i].isLeftOf(pointsByX[mid]))|| pointsByY[i] == XL[mid] ){
                                              YL[yLLength] = pointsByY[i];
                                              yLLength += 1;
}
```

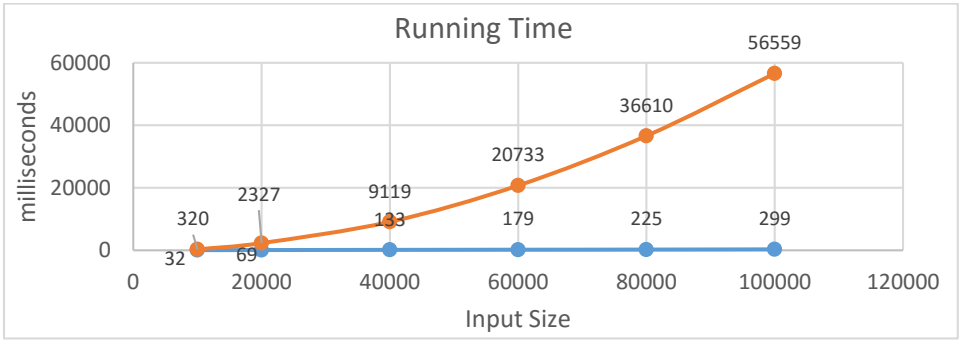
Previously, I missed this part

But I still cannot figure out why only half of the input could be detected by only missing this part, because there would be only few points fell on the divide line coincidently. I will go to TA for this problem afterward.

Thank the two TAs very much who was on duty on Saturday!

Part two

The comparison of running times of the Naïve algorithm and the Divide-and-Conquer algorithm with the increasing of input size



Time 100 runs with the same input of size 50000. I added a for loop in **Main.java** like this:

```
if (points == null)
   points = genPointsAtRandom(nPoints, prng);

for ( int i = 0; i <100; i++){

// run the DC algorithm
{
   XComparator lessThanX = new XComparator();
   YComparator lessThanY = new YComparator();

   // DC CLOSEST-PAIR ALGORITHM STARTS HERE
```

The results of each time listed in the following table. The maximum running time is 219 milliseconds, the minimum running time is 34 milliseconds, and the mean running time is 41.34 milliseconds. A regularity is found as in the first runs, the running time decreases dramatically, and afterwards it becomes stationary in some extend. Why would this happen? Is there something wrong with my coding? I am very interested in it.

NO.01-10	NO.11-20	NO.21-30	NO.31-40	NO.41-50	NO.51-60	NO.61-70	NO.71-80	NO.81-90	NO.91-100
219	35	35	38	35	44	35	38	34	35
126	40	35	38	36	49	37	38	34	34
129	35	35	38	35	43	39	34	36	34
89	35	40	39	36	37	39	35	35	34
77	34	36	38	35	36	38	34	34	37
62	36	35	43	35	34	37	36	35	35
37	39	36	37	35	36	38	35	34	35
38	39	35	34	38	35	39	35	38	35
41	35	34	35	37	35	37	33	34	38
34	34	40	34	40	37	39	36	36	38

Run 100 randomly generated inputs of same size 50000. I added a for loop and called **genPointsAtRandom()** each time in the **Main.java** class.

```
if (points == null)
        points = genPointsAtRandom(nPoints, prng);
for ( int i = 0; i <100; i++){
        points = genPointsAtRandom(nPoints, prng);
// run the DC algorithm
{
   XComparator lessThanX = new XComparator();
   YComparator lessThanY = new YComparator();
```

The results is listed in the following table. The maximum running time is 250 milliseconds, the minimum running time is 31 milliseconds, the mean running time is 41.54 milliseconds. The same regularity of the variation of running time was observed as in the above part. I am very interested!

NO.01-10	NO.11-20	NO.21-30	NO.31-40	NO.41-50	NO.51-60	NO.61-70	NO.71-80	NO.81-90	NO.91-100
250	38	36	41	36	33	38	48	35	35
128	36	38	39	34	35	44	38	35	35
123	35	34	42	34	37	46	34	34	33
88	46	34	34	35	33	41	36	33	33
47	40	35	34	32	40	44	37	31	33
59	44	37	33	34	42	33	34	33	35
37	35	34	35	32	43	33	37	38	35
38	34	40	37	35	40	37	37	32	35
35	36	42	46	34	41	43	34	37	34
34	39	43	37	36	43	39	35	35	33

Part 3

The graph showing the crossover running time is showed below. From the graph it can be concluded that before the first five points, the Naïve algorithm runs faster than the Divide-and-Conquer algorithm. But, afterwards with the growing of the input size, Divide-and-Conquer algorithm walks over the Naïve algorithm.