Programming implementation:

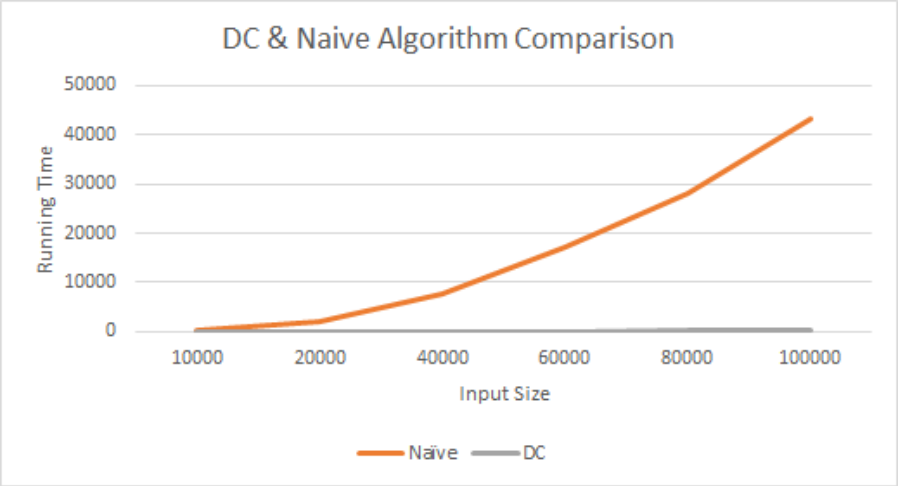
In ClosestPairNaive, I create a global variable minDist with value of infinity to store the smallest distance and two global variable p1 and p2 to store the indices of the 2 corresponding points. Then, I used a loop within a loop to iterate through each pair of points and compare their distance to the minDist, and set minDist to whichever is smaller.

In ClosestPairDC, I created a global variable minDist with value of infinity to store the smallest distance and two global variable p1 and p2 to store the the 2 corresponding points. I implemented the divide-and-conquer algorithm in this class, and make change to the global variables every time the divideNConquer method is called. The divideNConquer method is the one that I use for accomplishing the recursion of the algorithm and enable the findClosestPair method to print the correct results after calling divideNConquer.

One thing that I noticed in the process of implementation is that when I was debugging and printing variable values in the middle of the divideNConquer algorithm, the DC class ran a lot longer than the NAÏVE class to reach the result. The printing cost a significant amount of time comparing to the other parts of the program.

Comparison of two algorithms:

Firstly, the running time comparison with the input sizes of 10000, 20000, 40000, 60000, 80000 and 100000:



Input Size	Naïve	DC
10000	450	26
20000	1943	41
40000	7598	63
60000	17122	78
80000	28002	122
100000	43386	141

As we can see, the DC Algorithm runs in significantly shorter time comparing to the Naïve Algorithm. We even cannot really see the change in running time of DC Algorithm in the plotting because the values are too small to be shown within the same graph of the Naïve Algorithm.

Running 100 times of same input of size 50000, the DC Algorithm has an average running time of 24.08, with a maximum value of 75 and a minimum value of 21. However, the Naïve Algorithm has an average running time of 10925.32, with a maximum value of 11939 and a minimum value of 10676.

Running 100 times of different input of size 50000, the DC Algorithm has an average running time of 45.3, with a maximum value of 89 and a minimum value of 20. However, the Naïve Algorithm has an average running time of 11212.6, with a maximum value of 11930 and a minimum value of 10813.

We can see that when given different inputs each time, both algorithms' running times vary more from one input to the next comparing to given same inputs. This is because each time different sets of points are given, and the cost of each step/line becomes more inconsistent because the x and y coordinates are changing substantially, while the program uses slightly different times to process different values.

The Crossover Point:

I programmed the two algorithms to run through each input 120 times and output the average running time as below:

Input Size	Naïve	DC
5	0.0083	0.03
10	0.0083	0.058
20	0.017	0.075
40	0.03	0.092
100	0.067	0.015
150	0.133	0.175
175	0.167	0.2
200	0.208	0.208
210	0.242	0.225
250	0.317	0.258
300	0.483	0.3

As we can see, when the size of input is small, the Naïve Algorithm runs faster than the DC Algorithm because the structure of the DC Algorithm makes running small input size clumsy. However, as the size of input increases, the difference between the running times of the 2 algorithms becomes smaller and smaller, and at the input size of 200, the running times of the 2 algorithms become the same. As the size of input exceeds 200, the DC Algorithm becomes faster than the Naïve Algorithm, and the difference becomes larger and larger as the size of input increases.