<u>Implementation of Closest-Pair Algorithm</u>

<u>Overview</u>

This lab has several goals:

- Ensure that you understand the closest-pair algorithms studied in class.
- Provide first-hand experience with the practical benefits of nontrivial algorithm design, in particular the divide-and-conquer methodology

Part One: Implement the Algorithms (75%)

To complete this section, you must implement the two closest-pair algorithms discussed in class: the naïve algorithm, and the divide-and-conquer algorithm. Your implementation will take as its input a set of n points Your algorithms should find the closest pair of points in the input and print their coordinates, along with the distance between them

<u>Part Two: Do the Comparison (25%)</u>

The following two studies look at some aspects of the performance of the naive and divide-and-conquer

closest pair algorithms. Once you have working implementations of the naive and divide-and-conquer closest-pair algorithms, you should first compare their running times on inputs of the following sizes: 10000, 20000, 40000, 60000, 80000, 100000. You should produce and turn in a single graph plotting the running times of both algorithm versus input size. If your naive implementation takes more than about ten minutes for a given size n, you need not plot its running time for larger input sizes

Pseudocode

Pseducode- Naïve Algorithm

ALGORITHM *BruteForceClosestPoints*(P)

//Input: A list P of n ($n \geq 2$) points $P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n)$
//Output: Indices $index1$ and $index2$ of the closest pair of points
$dmin \leftarrow \infty$
for $i \leftarrow 1$ **to** $n-1$ **do**
  for $j \leftarrow i+1$ **to** n **do**
    $d \leftarrow sqrt((x_i - x_j)^2 + (y_i - y_j)^2)$ //$sqrt$ is the square root function
    if $d < dmin$
      $dmin \leftarrow d$; $index1 \leftarrow i$; $index2 \leftarrow j$
return $index1, index2$

Pseducode- Divide & Conquer

```
Closest-Pair(p1, …, pn) {
   Compute separation line L such that half the points
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Delete all points further than δ from separation line L

   Sort remaining points by y-coordinate.

   Scan points in y-order and compare distance between
   each point and next 11 neighbors. If any of these
   distances is less than δ, update δ.

   return δ.
}
```