CS2223 - Introduction to Algorithms
Worcester Polytechnic Institute **BTerm 2017**

Project 2 - Comparing Divide and Conquer with Brute Force Algorithms

This project is due **Tuesday, Nov 21** at **11:59PM**. Upload a zipped file named `yourlastnameFirstinitial_proj2` to Canvas. The file must include a(n):

1. Executive Summary Report in PDF format
2. Python (.py) file
3. readme.txt file with instructions on how to run your code

Implement a divide and conquer (recursive) and brute force (non-recursive) algorithm in Python for calculating the Closest-Pair Problem. This problem calls for finding the two closest points in a set of n points.

You can assume all points are represented by (x, y) Cartesian coordinates and the distance between two points $p_i(x_i, y_i)$ and $p_j(x_j, y_j)$ is the standard Euclidean distance $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

Here are the steps to complete the project:

- Step 1. Write the brute force algorithms in Python. The brute force algorithm checks every combination using the above calculation.
- Step 2. Write the recursive algorithm. The recursive algorithm pseudo code is in Figure 1 below.
- Step 3. Write functions to time the execution of each of the algorithms by creating two additional functions:

```
effRec(n)
Start the clock
Call your recursive function
Stop the clock
Print the two closest points and time to calculate

effBF(n)
Start the clock
Call your brute force function
Stop the clock
Print the two closest points and time to calculate
```

```
ALGORITHM  EfficientClosestPair(P, Q)
    //Solves the closest-pair problem by divide-and-conquer
    //Input: An array P of n ≥ 2 points in the Cartesian plane sorted in
    //       nondecreasing order of their x coordinates and an array Q of the
    //       same points sorted in nondecreasing order of the y coordinates
    //Output: Euclidean distance between the closest pair of points
    if n ≤ 3
        return the minimal distance found by the brute-force algorithm
    else
        copy the first ⌈n/2⌉ points of P to array P_l
        copy the same ⌈n/2⌉ points from Q to array Q_l
        copy the remaining ⌊n/2⌋ points of P to array P_r
        copy the same ⌊n/2⌋ points from Q to array Q_r
        d_l ← EfficientClosestPair(P_l, Q_l)
        d_r ← EfficientClosestPair(P_r, Q_r)
        d ←min{d_l, d_r}
        m ← P[⌈n/2⌉ − 1].x
        copy all the points of Q for which |x − m| < d into array S[0..num − 1]
        dminsq ← d^2
        for i ← 0 to num − 2 do
            k ← i + 1
            while k ≤ num − 1 and (S[k].y − S[i].y)^2 < dminsq
                dminsq ← min((S[k].x − S[i].x)^2+ (S[k].y − S[i].y)^2, dminsq)
                k ← k + 1
    return sqrt(dminsq)
```

Figure 1: Recursive Closest Pair Pseudo Code

- Step 3. Conduct a thorough experiment to compare the time efficiency of your recursive and non-recursive algorithms. Test at least 3 sets of points and provide the point sets in your readme file.

  Include the following point set to test your code:
  $[(0,0),(7,6),(2,20),(12,5),(16,16),(5,8),(19,7),(14,22),(8,19),(7,29),(10,11),(1,13)]$

- Step 4. Write your professional report. Answer the following questions:

  What is the time efficiency for your brute force algorithm? (set up equation and solve)

What is the time efficiency for your recursion algorithm? (set up equation and solve)

Grading Rubric Can find on Canvas