CS 320 Programming Project 2

Closest Points Computational Problem
Using Divide and Conquer Algorithm

DUE DATE is posted on the BlackBoard web site for the course.

PROJECT NAME and What to Submit:

- Project Name MUST be **CS320P02LastName** according to your section number.
- Zipped file name must be named: CS320P02*LastName*.zip.
- Zipped file must contain **the entire** project folder containing your Visual Studio 2019 (Debug-x32 configuration) solution-project for Project 2. Also, any test cases (input and corresponding output) must be included in that zip file. Be sure to show correspondence through some consistent naming convention.
- Provide a word document describing the analysis of your algorithm. Use appropriate notation. That should be in the submitted zip file also.
- If your program does not work, please include a Word or notepad file explaining what works and what doesn't work.
- The zip file should be submitted to the assessment icon on Blackboard.

Description of Closest Points Computational Problem:

Given a finite set S of n ≥ 2 points p = (x, y) where $S \subseteq X \times Y$, you are to find two points in S with the minimum Euclidean distance between two points in S. In this project you will let X and Y be the set of representable real numbers in the C++ data type, `double`. The distance between two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is defined by the Euclidean distance metric:

$$Distance(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

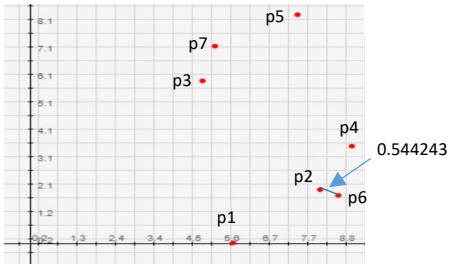
Example:

S = {p_1 = (5.60, 0.01), p_2 = (8.08, 1.93), p_3 = (4.80, 5.85), p_4 = (8.96, 3.50), p_5 = (7.47, 8.22), p_6 = (8.59, 1.74), p_7 = (5.14, 7.11)}

Example File Format

```
5.60 0.01
8.08 1.93
4.80 5.85
8.96 3.50
7.47 8.22
8.59 1.74
5.14 7.11
```

Cartesian Plane Depiction



Closest points are: p2 = (8.08, 1.93) and p6 = (8.59, 1.74) with distance = 0.544243

NOTE: You need not implement the graph shown on the upper right. However, you might find it to be a useful tool for examining your test data. This plot data was generated using http://www.shodor.org/interactivate/activities/SimplePlot/.

Brute Force Algorithm:

```
Read all points (n points) into an array
for each input point p_i = (x_i, y_i)
        for each other point p_j = (x_j, y_j)
                compute distance between p_i and p_j
                if ( distance < minimum distance ) minimum distance = distance
```

Divide and Conquer Approach: See your textbook for this approach to solve the problem. (Assume an vector, P, of size n and components of type Point. Also assume that the points in P have been sorted by x coordinate.) This pseudo-code is based upon, but modified, from Geek-for-Geeks. This code only returns the smallest distance. *Your implementation must return two Points: p = (x, y) and p' = (x', y') such that p ≠ p', distance(p, p') ≤ distance(p_i, p_j) for all p_i, pj and i≠j.* Also, your program must also print the distance between the p and p'.
https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/

***Note**: The algorithm and data structures used below must be convert to C++11 code and meet the following constraints: you must use the supplied Point class, and implement the sets of points as vector<Point> and you must return two Points representing the two points with the smallest distance between them rather than just the distance.*

1) Find the middle point in the sorted array, we can take P[n/2] as middle point.

2) Divide the given array in two halves. The first subarray contains points from P[0] to P[n/2]. The second subarray contains points from P[n/2+1] to P[n-1].

3) Recursively find the smallest distances in both subarrays. Let the distances be dl and dr. Find the minimum of dl and dr. Let the minimum be d.

4) From above 3 steps, we have an upper bound d of minimum distance. Now we need to consider the pairs such that one point in pair is from left half and other is from right half. Consider the vertical line passing through passing through P[n/2] and find all points whose x coordinate is closer than d to the middle vertical line. Build an array strip[] of all such points.

5) Sort the array strip[] according to y coordinates. This step is O(nLogn). It can be optimized to O(n) by recursively sorting and merging. You must use mergesort the points in your implementation and sort points in the strip based upon the y-coordinates.

6) Find the smallest distance in strip[]. From first look, it seems to be a O(n^2) step, but it is actually O(n). It can be proved geometrically that for every point in strip, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate). For more analysis, see http://people.csail.mit.edu/indyk/6.838-old/handouts/lec17.pdf

Here is sample pseudocode for steps 5 and 6: finding the min distance within strip and across midpoint. This is from https://www.youtube.com/watch?v=0W_m46Q4qMc. Also, see your textbook.

```
double closestAcrossStrip(Point strip[], unsigned size, double d)
{
        double min = d;
        sort(strip, sortOnYcoordinate);
        for(unsigned i = 0; i < size; i++)
                for(unsigned j = i + 1; j < size && (strip[j].y-strip[i].y) < min; j++)
                        if(distance(strip[j],strip[i]) < min )
                                min = distance(strip[j], strip[i]);
```

7) Finally return the minimum of d and distance calculated in above step (step 6). This is the minimum of the left subset, right subset, and strip.

NOTE: What is not clear from the above pseudo-code from steps 1-7 is the base case. Use the brute-force nested loop approach when the amount the number of points in the subset is 4 points or less.

Driver Algorithm:

1) Prompt user for filename according prompt specifications shown below.
2) Open file for reading. Use only text files (.txt file extensions).
3) Read file of n > 1unique points into a `vector<Point>` object.
4) Call closest() and have it return the closest pair, where the prototype of closest is:
   `pair<Point,Point> closest(vector<Point> points)`
5) Print the each point returned from `closest()` and distance between them according to output specifications shown below.

Input Specifications. Your program must read a .txt file containing the points as ordered pairs of literals of type double each separated by whitespace. Program must prompt the user for a filename containing the input points. The ***filename.txt*** must be replaced with an existing filename that represents a file containing the points.

```
Enter Filename: filename.txt
```

The prompt must be exactly as shown. One space after the colon is required.

The file must contain n>1 ordered pairs of literals of type double. (See example above with the file of 7 points.) Although only whitespace is required, it easier to see the points in files with new lines after point, i.e., after each pair of coordinates. You may assume that the input files have unique points --- no repetition of points.

Output Specifications. See below for format. Your output text and spaces must match exactly. Note that balanced parentheses and comma are added for the output presentation of points.

`Closest points are:`(x1, y1) `and` (x2, y2) `with distance =` *<double value representing distance between the closest points>*

Output Example. Using the above input example, the output from that problem instance is shown below.

```
Closest points are: (8.08, 1.93) and (8.59, 1.74) with distance = 0.544243
```

Note: you will not be able to use the "<<" operator or `print()`member function in the supplied Point class. You must implement a way to meet the above output specification.

Constraints on solution.

- You must you used the `Point` class supplied.
- You must use `vector<Point>` to represent the *set* of points.
- You must use `mergesort` on `vector<Point>` objects. You must also be able to modify the `mergesort` implementation in your textbook so that it accepts a function object parameter so that a `vector<Point>` parameter can be sorted on x-coordinates or y-coordinates.
- You must implement a divide and conquer algorithm based upon the algorithm presented in your textbook or as shown above.
- Use object-oriented design to solve this problem. If no classes are used, you will have points deducted.
- For full credit, your implementation must be able to compute problem instances faster than the brute force solution for n > 5000 points. Points will be deducted each solution not computed within an expected time limit.
- For full credit you must show that you tested your code by showing test files and results. Although they are not required, you it's helpful to create two additional project: Generate Points project and Brute Force project so you can make sure your Project 2 code is working.

```
#ifndef __POINT__
#define __POINT__
#include <iostream>
#include <fstream>
#include <math.h>
using namespace std;
class Point
{
public:
        Point()
        {
                this->x = 0.0;
                this->y = 0.0;
        }
        Point(double x, double y) {
                this->x = x;
                this->y = y;
        }
        Point(double x)
        {
                this->x = x;
                this->y = 0;
        }
        Point(const Point & p)
        {
                this->x = p.x;
                this->y = p.y;
        }
        const Point & operator=(const Point  & rhs)
        {
                if (this != &rhs) {
                        x = rhs.x;
                        y = rhs.y;
                }
                return *this;
        }
        bool operator==(const Point p) const
        {
                return (this->x == p.x && this->y == p.y);
        }
        bool operator!=(const Point p) const
        {
                return (this->x != p.x || this->y != p.y);
        }
        double distance(Point p) const
        {
         return sqrt((this->x - p.getX())*(this->x - p.getX()) + (this->y - p.getY())*(this->y - p.getY()));
        }
        double getX() const {
                return this->x;
        }
        double getY() const {
                return this->y;
        }
        void setX(double x)
        {
                this->x = x;
        }
        void setY(double y)
        {
                this->y = y;
        }
        class CompareXCoordinate {
        public:
                bool operator()(const Point & p1, const Point & p2) const
                {
                        return (p1.getX() < p2.getX());
                }
        };
        class CompareYCoordinate {
        public:
                bool operator()(const Point & p1, const Point & p2) const
                {
                        return (p1.getY() < p2.getY());
                }
        };

        private:
          double x;
          double y;
};
```

```
ifstream & operator>>(ifstream & in, Point & p) {
        double x;
        double y;
        in >> x;
        in >> y;
        p.setX(x);
        p.setY(y);
        return in;
}
ostream & operator<<(ost ream & outfile, const Point & p)
{
        outfile << p.getX() << ' ' << p.getY();
        return outfile;
}
#endif
```