COP 4930/5930 – Computational Geometry[1]
Homework 1
Full grade: 20 points

☞ *You can collaborate with **at most** one other student from this class.*

1 Problem description

In this assignment, you need to ***engineer*** (implement in the most efficient possible way) the $O(n \log n)$-time divide-and-conquer algorithm for finding closest pair in C++ using the CGAL library. Put this implementation inside a new header file named **`LinearithmicTimeClosestPair.h`**. Name your function **`findClosestPairLinearithmic`**. It should accept just one parameter **`const std::vector<Point> &P`** and return a pair of type **`std::pair<unsigned, unsigned>`** that contains the two indices of the closest pair in **`P`**. An algorithm is said to have *linearithmic* time complexity if its runtime is $O(n \log n)$.

Every single step should be implemented carefully so that the whole algorithm runs in $O(n \log n)$ time. ***Please go through the algorithm and understand it clearly before you start coding. Otherwise, you will lose time and may miss the deadline.***

- For ***sorting*** you must use the built-in sorting function **`std::sort`**. See here for more information: `https://www.cplusplus.com/reference/algorithm/sort/`. Make sure that you are using the correct comparison operator when you are sorting the points based on x or y-coordinates. The default comparison operator for the `Point_2` class will sort the points lexicographically, which is not needed in this assignment. For your knowledge: given two points $p = (p_x, p_y)$ and $q = (q_x, q_y)$, we say that p is lexicographically small than q if $p_x < q_x$ or $p_x = q_x$ but $p_y < q_y$.
- Make sure that you are not sorting anything once the recursion starts. Otherwise, your implementation will not run in $O(n \log n)$ time anymore!
- Use **`std::vector<Point>`** when you need to create new arrays in order to avoid memory leaks. If you have an estimate on the size of a vector in advance, use the **`reserve`** function on it to allocate space in advance. Then you can insert items into it. This may speed up your code in many cases.
- Since your code will be recursive, be careful about the base case. Otherwise, your code may crash during execution.
- Use the random point generator supplied to you for generating random point sets for testing and experiments.
- For correctness checking, make sure that for every test input, the answers returned by this algorithm and the quadratic time algorithm are exactly the same. If not, you need to debug your code. Feel free to use the Qt GUI we have used in our lectures for visualizations.

[1] LaTeX *compiled on February 3, 2022 at 5:38pm*

- Make sure that you are using the **`-O3`** flag in your CMake file inside CLion to optimize your code.
- If you are using a specific constant expression over and over again, please pre-compute and store in some **`const`** variable. For instance, if you are using $\sqrt{3}/2$ multiple times in your code, just do **`const double sqrt2 = std::sqrt(3)/2;`**. This will help your reduce execution times.

Fun fact. How to test experimentally if your code is running in $O(n \log n)$ time? For all values of n which are sufficiently large enough, running time divided by $n \log_2 n$ (other bases of the log can be used as well) should always be at most some fixed positive constant.

Plotting and comparing the runtimes. You need to use **GnuPlot** (`http://www.gnuplot.info/`) to plot the runtime graphs of the two algorithms. This software can be easily installed on your Linux by running **`sudo apt-get install gnuplot`** from terminal.

For this assignment, use $n = 1K, 2K, \ldots, 20K$. For every value of n, use 5 random samples (pointsets in this case) and take the average running times of the five runs. But make sure that you feed the same pointset to both the algorithms otherwise comparison would not make any sense. You need to send the experimental results to a text file, say **`a.txt`** (see `https://www.cplusplus.com/doc/tutorial/files/` to learn how to redirect output items to a file). However, this file needs to be formatted in a specific way. See next.

Here is one example for you (created by hand). The first column must contain the n-values. To save space on this document I have used $n = 1K, 2K, \ldots, 5K$. The second column must contain the runtimes (averaged over five samples for every value of n) for the quadratic time algorithm. Finally, the third column must contain the runtimes (averaged over five samples for every value of n) for the $O(n \log n)$-time algorithm.

```
1000 5 9.9
2000 10 10.1
3000 15 17
4000 20 19
5000 25 50
```

Your header file must also contain a no-argument function **`void plot()`** that can create this text file. This function may be used to see the plots from the grader's end. To plot the two runtime curves, just type the following from your terminal. Make sure the text file is present in your current working directory. Command for generating the plots from file (***retype the single quotes before and after the two file names in this command on your terminal before you execute it***): **`gnuplot -p -e "plot 'a.txt' using 1:2 with linespoints lw 3 pt 7, 'a.txt' using 1:3 with linespoints lw 3 pt 7"`**

A window should pop up. There should be a button on this window to save the plots to a PDF. Go ahead and save it before your close this window.

2 What to deliver?

Upload just the two files **`LinearithmicTimeClosestPair.h`** (containing the two above-mentioned functions: **`plot`** and **`findClosestPairLinearithmic`**) and the GnuPlot PDF to Canvas.

Commenting is highly encouraged. Please test thoroughly before submission. You will be given zero if your program does not compile or gives unnecessary warnings. You are allowed to submit your solutions multiple times. We will grade your latest submission only. Your code will be graded using the CLion IDE on macOS/Linux.