CS325 programming assignment 1

January 14, 2025

1 General Instructions

- Students are encouraged to work in groups. Each group can contain up to 3 students. See "Guidance in forming groups and working in groups" for instruction if you are not familiar with canvas group work.
- Each group should create a single submission. In your submission, please explicitly state the names of all group members. Even though canvas will keep track everyone in the group, keeping it explicit in your submission helps avoid issues in case of mistakes in your canvas operations.
- Your report should be typed, properly formatted in PDF. Figures and tables should be have clear legends and captions.

2 Problem: Closest pair of points

In this assignment, you will solve the problem of finding the closest pair of points among a set of n points in the 2-d plane, which is an important computational geometry problem that has many applications in graphics, computer vision, GIS and air traffic control etc. Specifically, The input to your algorithm is a set of n points in a two-d array, where each row is a data point, and the first column stores the x-coordinate and the second column stores the y-coordinate. Your output should provide the shortest distance, and all pairs of points that have that shortest distance between them.

3 Algorithmic design: divide and conquer

The main idea you will explore for this problem is **divide and conquer**, which we briefly describe below.

The divide-and-conquer algorithm works by: dividing the point set into two nearly-equal halves, achieved by splitting the set at the median x-coordinate. Let x_m be the x-coordinate of the dividing line separating the two point sets.

We recursively compute the closest pair in each half, and then solve the merge step based on their results. More specifically, suppose d_1 and d_2, respectively,

are the closest pair distances for the left and right subproblems, and let $d = min(d_1, d_2)$. Determine the set of points whose x-coordinates lie in the range $[x_m - d, x_m + d]$. Call this set M, for the middle. Order the points of M in ascending order of their y-coordinates. We find the closest pair of points in M by comparing each point p to only those whose y-coordinate differs from p by at most d. Let their distance be d_m, we return $min(d, d_m)$.

4 Algorithms to be implemented

For this assignment, you will implement three different algorithms for this problem.

1. Brute-force. This algorithm works simply by computing the distance between all $\binom{n}{2}$ pairs of points and finding the closest pair. This algorithm will have $O(n^2)$ run time.

2. Naive Divide and Conquer. Implement the above outlined divide-and-conquer algorithm for computing the closest pair. In this naive version, you will sort the points within M based on y-coordinates in each recursive call from scratch.

3. Enhanced divide and conquer. In this version, you will eliminate the repeated sorting by pre-sorting all the points just once based on x-coordinates and once based on y-coordinates. All other ordering operations can then be performed by copying from these master sorted lists.

4.1 Implementation details

For this assignment, you will be completing the following three functions using in python in their respective python files: Do not deviate from these python function names, as they will be directly tested and expecting those inputs/outputs

```
# brute_force.py
def brute_force_closest_pair(points):
    ...
    return min_distance, sorted_points

# divide_and_conquer.py
def naive_divide_and_conquer_closest_pair(points):
    ...
    return min_distance, sorted_points

# enhanced_dnc.py
def enhanced_divide_and_conquer_closest_pair(points):
    ...
    return min_distance, sorted_points
```

Where:

- **points** is a list of lists (or tuples), where each nested tuple is a [x,y] point in a 2d graph
- **min_distance** is the minimum distance between any two sets of points. This value should be a float datatype that is rounded to 4 decimals
- **sorted_points** is the sorted list of pairs of points with the mim_distance (see details about formatting in the example below).

For example, the input **points** may be structured like:

```
points = [[0, 1], [2, 3], [4, 5], [5, 6], [8, 9]]
```

The corresponding outputs would be:

- **min_distance** is 1.4142
- **sorted_points**

  `=[ [[2, 3], [4, 5]], [[4, 5], [5, 6]] ]`

In this case, there are two pairs with the same min_distance of 1.4142. The output will list both pairs. To ensure consistency in the outputs, please sort your points both Within pair, and across pairs.

- **Within pair**: sort the two points first by the X-coordinate, and then by the Y-coordinate if the X-values are equal.
- **Cross pair**: Sort the pairs based on the X-coordinate of the first point in each pair, and if the X-values are equal, then by the Y-coordinate of the first point.

4.2 Starter files provided

The following starter files are provided:

- **brute_force.py**, **divide_and_conquer.py**, **enhanced_dnc.py**
- **ia1_utils.py** General utility functions for IA1 to read and write the data to a file, generating the random input of specified size, sorting a list of pairs in required order etc. You can complete and test your 3 algorithms using these functions. You do not need to strictly use these functions for your implementation, but we will be during our tests.
- **input1.txt** and **output1.txt** A small test input and output file

5 Experimenting with your implementation

The provided starter files contain the function name, input parameters, and the return arguments. If you add additional input parameters, make sure they are optional inputs. Aside from testing the correct outputs written to the output file, we will be testing the functions themselves for their estimated empirical time complexity.

5.1 Empirical testing of correctness

To ease the grading, please make a separate runnable python file for each algorithm (Brute Force, Divide and Conquer, Enhanced DnC).

So for example, we would run your homework like:

```
> python3 brute_force.py example.input
> python3 divide_and_conquer.py example.input
> python3 enhanced_dnc.py example.input
```

We expect each of these programs to output "output_bruteforce.txt", "output_divideandconquer.txt", and "output_enhanceddnc.txt" respectively.

The three algorithms should produce the same outputs. You are provided with sample input/output files (currently there is only one at the time of this assignment's release, more is forth coming) that can be used to verify if your program finds the correct solution. During grading, we will further assess the correctness of your algorithms by running them on additional test cases and comparing the results against the corresponding ground truth solutions.

Note that full credits for the correctness of the divide and conquer algorithms will also depend on the running time. For example, we expect to see $n \log n$ runtime for a **correct** implementation of the enhanced divide and conquer algorithm. You will not get full credit if your implementation does not achieve this runtime, even if your outputs are correct.

5.2 Empirical measure/analysis of run time.

For this part, you need to generate random input data of varying sizes using a random number generator and run your algorithms on these inputs to measure their empirical run time. We have provided a utility function, "generate_random_input_file", to help you generate input files. If you prefer, feel free to modify this function to directly generate inputs for your algorithms, which can help avoid unnecessary I/O operations.

The suggested range for input size n is $10^2, 10^3, 10^4, 10^5$. If time allows, you are encouraged to try even larger input sizes for the enhanced divide and conquer algorithms.For each input size, generate 10 random datasets (by varying the random seeds) and run each algorithm on them. Report the average runtime over these ten runs.

Additionally, generate a runtime plot for the three algorithms. Use a logarithmic scale for the input size on the x-axis. Plot the runtimes of all three

algorithms in a single figure (if the runtime of one algorithm is dominantly large making the small hard to see, you can use logarithmic scale for y-axis as well), using distinct colors and markers to differentiate them. Be sure to include appropriate labels and a legend in the figure.

6 Report

In addition to submitting the complete set of source code (with clear instructions for running them), you will also need to submit a typed report, which should include the following Sections:

- **Algorithm design and Pseudo-code.** Please describe your algorithm and provide the Pseudo-code for each of the three algorithms. The description could be brief but should be easily understandable.
- **Asymptotic Analysis of run time.** For each of the three algorithm, provide a asymptotic runtime analysis.
- **Empirical runtime.** Plot the empirically measured runtime of the three algorithms as a function of the input size. Your plot should have clearly labeled axes and legends.
- **Interpretation and discussion** Discuss the runtime plot in the context of the asymptotic runtime. Do the growth curves match your expectation based on their asymptotic bounds? Discuss and provide possible explanations for any discrepancy between the experimental runtime and the asymptotic runtime. For example,