Name: Longchang Cui

ID: 16242D

Course: EN.605.621.81.SP21 Foundations of Algorithms

Programming Assignment #1

1. Conditions:

Consider a set, *P*, of *n* points, (*X*1*, Y*1)*,…,*(*Xn, Yn*), in a two dimensional plane.

Distance of two points = $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

2. Closest Pairs

a.

i. Algorithm for finding m closest pairs of points:

Approach:

The easiest way to find the closest pairs would be looping through each point in the set P and comparing the distance between points. Then, once we get the minimum distance of pairs, store them into a dictionary with distance value as a key. Next, sort the dictionary by the key. At last, return the m size of the pairs and distance from the sorted dictionary.

Pseudo Code is on next page

Finding-Closest-Pairs(P, m)

1 Let $D = \{key: value\}$ be a new dictionary

2 Let *min-distance* be a new float

3 **for** $i = 1$ **to** $P.length$ - 1

4 *min-distance* = ∞ // use min-distance to store the minimum distance of pairs

5 **for** $j = 1$ **to** $P.length$ - 1

6 **if** $i \neq j$ // avoid calculating the distance of self

7 *distance* = $\sqrt{(P[i].x * P[j].x)^2 + (P[i].y * P[j].y)^2}$

8 **if** *min-distance > distance*

9 *min-distance = distance*

10 // use min-pair to store the minimum distance pairs

11 Let *min-pair* be a new array

12 *min-pair.append*($P[i],\ P[j]$)

13 // Add the minimum distance as a key, and minimum pairs as a value to the dictionary *D*

14 *D.add*(*min-distance, min-pair*)

15 // Sort the dictionary D by the key with TimSort

16 *D.sort*(*min-distance)*

17 // Loop through the sorted dictionary and return the m number of distance and pair*s*

18 Let *result-distance* be the new array

19 Let *result-pairs* be a new array

20 **for** $k = 1$ **to** *m*

21 *result-distance.append*(*D.keys[k]*)

22 *result-pairs.append(D.values[k])*

23 **return** *result-distance, result-pairs*

ii. Determine the worst-case running time of the algorithm.

The running time for the algorithm is very slow.

Comparing distance of points in worst-case will take $O(n^2)$

Sorting the closest pairs distance in the dictionary in worst-case will take $O(n \lg n)$ with TimSort.

Obtaining the sorted result in worst-case will take $O(n)\ if\ m = n$

The overall time complexity for this algorithm will be

$$T(n) = n^2 + n\ lg\ n + n$$

Thus, the worst-case running time of this algorithm is

$$T(n) = O(n^2)$$

b.

The source code file is named "*LongchangCui_PA1.py*"

It is written in Python 3.7.4 on Windows 10.

To execute the program, run the command "*python LongchangCui_PA1.py*" in the terminal/command prompt.

c.

The output file of trace run is "TraceRunReport.txt"

The trace run output was created by running the following command on the terminal.

python -m trace --count --summary LongchangCui_PA1.py

d.

I added integer variables in the source code to test the asymptotic behavior.

The testing variables I used are

1. *test_i_loop*: This variable will be incremented on each step we loop through a point.

2. *test_j_loop*: This variable will be incremented when we compute distance between two points on each step of inner for loop.

3. *test_k_loop*: This variable will be incremented as m number of pairs we wish to find increases.

The *test_i_loop, and test_j_loop* are initialized on line 17 and line 18 in the source code "LongchangCui_PA1.py"

The *test_k_loop* is initialized on line 48 in the source code "LongchangCui_PA1.py"

Test Cases:

Test Case 1. *m* = 2 and *Length of P* = n = 1000

```
Number of times test_i_loop executed: 1000
Number of times test_j_loop executed: 1000000
Number of times test_k_loop executed: 2
```

Test Case 2. *m* = 4 and *Length of P* = n = 2000

```
Number of times test_i_loop executed: 2000
Number of times test_j_loop executed: 4000000
Number of times test_k_loop executed: 4
```

Test Case 3. *m* = 8 and L*ength of P* = n = 4000

```
Number of times test_i_loop executed: 4000
Number of times test_j_loop executed: 16000000
Number of times test_k_loop executed: 8
```

e.

As I increase the size of n on each test case, the execution number of the inner for loop also increases exponentially. By observing the nested for loop execution number of each test case, I can conclude that the code's worst-case running time is indeed $O(n^2)$ if we have enormously large size of n.

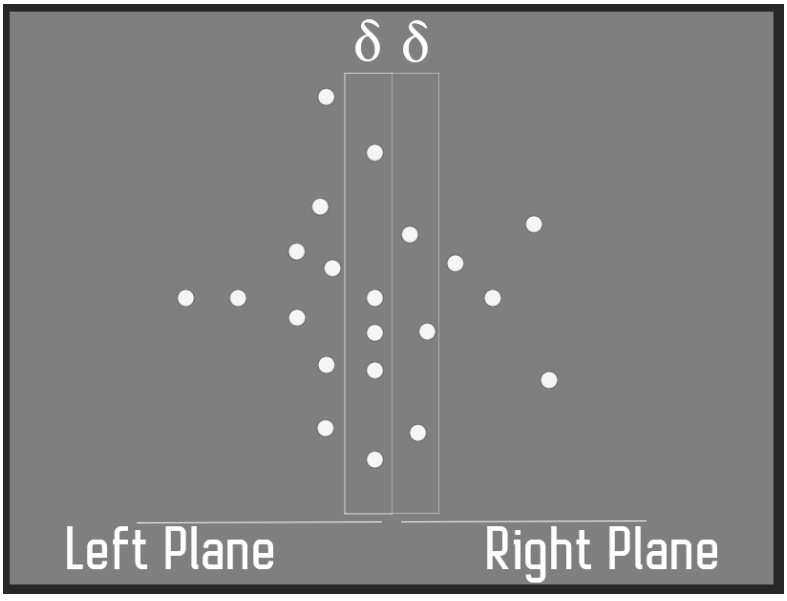
3. Retrospection

Looping through each point and checking the distance of each pair is very time consuming and not necessary. Changing part of my code by using better data structure and the sorting algorithm will not increase the worst-case running time.

After doing some research, I've learned that I can use Dived and Conquer approach to significantly lower the overall running time. I can draw a vertical line in the plane defined by a median x-coordinate. Then, dived the plane into half, and recursively compute the closest points on the left and right planes to get the minimum pairs. Once the minimum pairs on each plane are computed, I need to find all points that are closer to the middle vertical line which is within a certain distance δ. After that, I can recursively sort and merge the points, then, finally return the *m* number of smallest distance pairs.

If I use this approach, I can reduce the worst-case running time from $O(n^2)$ to $O(n\ lg\ n)$

Illustration:



References:

Subhash Suri. (2002, November 2). *Closest Pair Problem*. University of California, Santa Barbara. https://sites.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf

Wikipedia contributors. (2020, July 10). *Closest pair of points problem*. Wikipedia. https://en.wikipedia.org/wiki/Closest_pair_of_points_problem

Larry Ruzzo. (2011). *Algorithms: Divide and Conquer*. University of Washington. https://courses.cs.washington.edu/courses/cse421/11su/slides/05dc.pdf

Wikipedia contributors. (2020b, December 31). *Timsort*. Wikipedia. https://en.wikipedia.org/wiki/Timsort