605.621 Foundations of Algorithms Spring 2024
Project Assignment #1

1. **Natural Language Description of Algorithm**

   First step the algorithm would take is to take input for the algorithm. Then it would need to calculate distances using the provided Manhattan distance equation between each pair of points. Once that is done the algorithm needs to sort the list of point pairs by their Manhattan distance in ascending order. Once that is completed it will select the closes m pairs of points. Finally it will return the selected m pairs of points.

2. **Pseudocode**

```
FindClosestPairs($P, m$)
distanceList=[]                          //Create new blank array
for $i = 0$ to $P.length - 1$
    for $j = i + 1$ to $P.length$
        distance = $|P[i].x - P[j].x| + |P[i].y - P[j].y|$
        distanceList.append(distance, P[i], P[j])
MERGE-SORT(A, p, r)                      //CLRS page 39, Use this function to sort the list in ascending order
ClosestPairs = Select m entries of distanceList
return ClosestPairs
```

3. **Analytically derived Asymptotic family for algorithm**

FindClosestPairs(P, m)

1	distanceList=[]	c_1	1
2	**for** $i = 0$ to $P.length - 1$	c_2	n
3	**for** $j = i + 1$ to $P.length$	c_3	n
4	distance = $\lvert P[i].x - P[j].x\rvert + \lvert P[i].y - P[j].y\rvert$	c_4	$\sum_{i=0}^{n-1}(n-i-1)$
5	distanceList.append(distance, P[i], P[j])	c_5	$\sum_{i=0}^{n-1}(n-i-1)$
6	MERGE-SORT(A, p, r)	c_6	$n\log(n)$
7	ClosestPairs = Select m entries of distanceList	c_7	1
8	**return** ClosestPairs	c_8	1

We can simplify $\sum_{i=0}^{n-1}(n-i-1)$ to $\frac{n(n-1)}{2}$

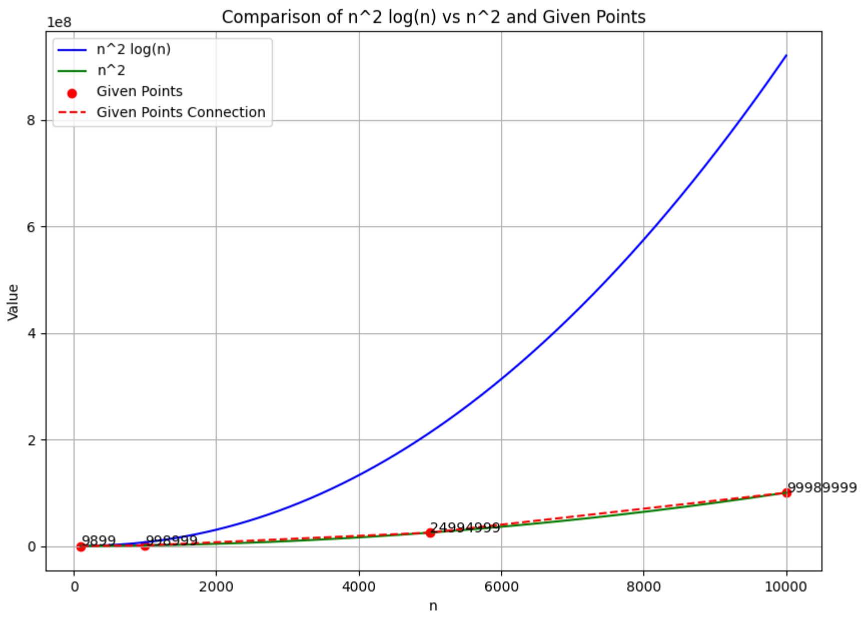
$T(n) = c_1(1) + c_2(n) + c_3(n) + c_4(\frac{n(n-1)}{2}) + c_5(\frac{n(n-1)}{2}) + c_6(n\log(n)) + c_7(1) + c_8(1)$

$T(n) = (c_2 + c_3) * n + (\frac{c_4}{2} + \frac{c_5}{2}) * n^2 + c_6(n\log(n))$

$T(n) = \Theta(n^2\log(n))$

4. **Analysis**

   Looking at we can get an overview of the codes actual runtime. We can see that as n increases the number of calculations seems to be roughly $\Theta(n^2)$.



This leads to the question, why is the calculated worst case runtime worse than the runtime found when running the program. The simple answer is that there are other variable besides the two loops that I added counters for, every line or item modified causes an increase in running time. More than likely my codes running time is $\Theta(n^2 \log(n))$. An example of something that can't really be taken into account is line 58 in my code to the compiler this is only ran once, however in actuality it is running through the list and chopping part of it off. It would be impossible to show the running time it is actually taking up by a simple print statement. In summation, while I found the code is running at more or less $\Theta(n^2)$ time, in actuality it is probably closer to estimated run time of $\Theta(n^2 \log(n))$.

5. **Retrospection**

   There are two major aspects of my code that I could change and reduce the worst case running time of my code. The first would be utilizing a more efficient sorting algorithm. In my current approach I use CLRS provided Merge Sort algorithm with a know runtime of $\Theta(n^2 \log(n))$. However, since we only care about m closest pairs, it would be possible to utilize a partial sorting method that stops once the m smallest distances are found would greatly reduce running time. The other aspect of my code that I could change is in the distance calculation loop. Currently my loop finds every single distance between every single point, which exponentially grows. Utilizing something such as a h-d tree to filter and find pairs that are more likely to be closer could drastically reduce the number of distance caluclations performed and thus reducing the running time.