FACULTY OF ENGINEERING AND APPLIED SCIENCE

SOFE 2715 - Data Structures

Course Project

Group 7

Dr. Shahryar Rahnamayan

MEMBER(S)			
#	Last Name:	First Name:	Student ID:
1	Bhavsar	Karn	100557957
2	Gege	Paul	100584697
3	Patel	Savan	100583384
4	Pedamallu	Sai	100598060

Table of Contents

Introduction:

The two topics we worked on are **K-Means** and **Convex-Hull**. We chose the two topics because both concepts are involved in many of our everyday activities and they are also easy research topics which are easy to implement. However we would have loved to work on the other harder topics such as the computer vision topics. But due to the heavy course-load of the semester our results would not have been as good.

The K-Means Clustering is about taking “n” observations and making “k” clusters where each observation belongs to the cluster with the nearest mean, also known as centroid. Some of the applications for K-Means can be seen in data mining, signal processing, computer vision. One can use K-Means Clustering to figure the skin color segmentations from a color image to help determine faces, bringing us back to the point made earlier on how the topics chosen are linked with the other project topics.

Convex-Hull is about taking a set of “x” points and finding the smallest convex shape that contains all points in “x” with the fewest number of perimeter nodes. To picture this, imagine having a rubber band wrap around the farthest points, thus containing all points of x. An application of Convex-Hull is collision detection of complex object. With Convex-Hull you can determine a much simplified geometry for a quick phase collision detection. Convex hull can also be used in making a planned route for a robot to follow.

Problems faced:

Earlier in the semester we chose to work on K-Means and Facial detection. Due to problems we faced while working on the Facial detection project, we switched to work on Convex-Hull. One of the problems with the facial detection was, we could not find an algorithm that was efficient and simple to implement, most of the solutions we found required the use of cascades. This didn't stop us from coming up with our own solution, we chose to make our solution based on the knowledge we had gathered about faces. What we initially came up with was to;

1. Scan image for skin color range in HSV
2. Have a threshold based on the human skin HSV values
3. Gather the clusters on the mask created by the threshold and use this to draw our squares to show our solution

This was not accurate enough, many other issues such as the noise in the picture and accuracy in predicting what was actually a face versus what was not. These problems listed led us to switch to the Convex-Hull topic. When working on the K-Means algorithm, we faced the issue of implementing our own self-made data structure, this is because we did not understand what exact functions we would include in the data structure class. For our Convex-Hull implementation for the algorithm we did not know how to efficiently store the points. To solve our issue of not knowing how to store the points, we created a "Points" class and the rest was a breeze to implement. However the one problem we never expected to face was the plotting of solutions in Java. We did not know how to plot points in Java, we tried using a tool called Jfree Charts to plot it but we were unsuccessful so we chose to just output the hull points computed from Java to a .txt file and then use python to plot it.

K-means: Explanation and Pseudocode

- Lloyd's Algorithm:
  - Works by selecting k data points as centers and choosing the closest cluster for these centers
  - If assigned clusters are not ideal, it reassigns the center
  - With new centers it chooses closest cluster again
  - Keeps on repeating until ideal centers are found/total amount of iterations are reached

Algorithm: Lloyd's Algorithm

Input: $E = \{e_1, e_2, ..., e_n\}$ (set of points to be clustered)

k (numbers of clusters)

$MaxIterations$

Output: $C = \{c_1, c_2, ..., c_n\}$ (set of centroids)

foreach $c_i \in C$ **do**	n
$c_i \leftarrow e_i \in E$ (assigning initial points from set as centroids)	1
end	
foreach $e_i \in E$ **do**	n
$label[i] \leftarrow minimumDistance(e_i, c_j)$	1
end	
changed ← false;	1
iter ← 0;	1
repeat	n
foreach $c_i \in C$ **do**	$n + (n-1) + (n-2) + ... + 1$
UpdateCluster(c_i);	1
end	
foreach $e_i \in E$ **do**	$n + (n-1) + (n-2) + ... + 1$
$minDistance \leftarrow minimumDistance(e_i, c_j)$	1
if $minDistance \neq label[i]$ **then**	1
$label \leftarrow minDistance$;	1
$changed \leftarrow true$;	1
end	
end	
iter ++;	1

Lloyd's algorithm takes $O(ndki)$time to run, where "n" is the number of points given, "i" is the number of iterations, "k" is the number of clusters and "d" is the dimensional vector. With the simple approach we were taught to analyse complexity with, we can conclude that this algorithm takes $O(n^2)$ time.

Convex-Hull: Explanation and Pseudocode

- **Gift Wrapping Algorithm (also known as Jarvis March)**
  - Choose leftmost point
  - Iterate through all points
  - Pick point with greatest angle
  - Point with greatest angle is now what we use to get the next point on the hull
  - Repeat this process until you get the first point on hull as another hull point

Algorithm jarvis(S)

Input: data points S $\{(x_1, y_1)\ldots(x_n, y_n)\}$ where n is the number of points

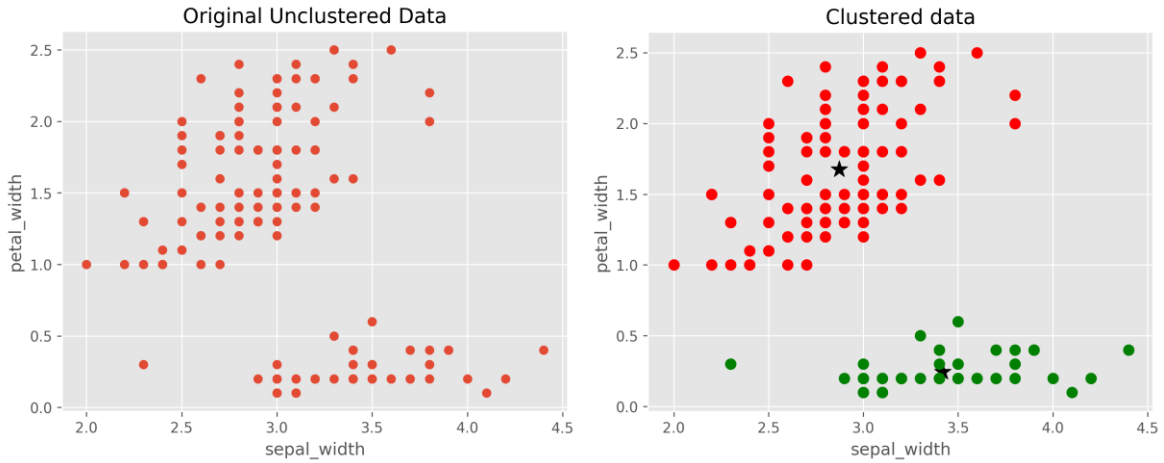
Output: set of points containing vertices $\{(x_1, y_1)\ldots(x_n, y_n)\}$ v is the number of vertices

vertex ← leftmost point in S	1
i ← 0	1
repeat	n
P[i] ← vertex	1
endpoint ← S[0]	1
for j from 1 to |S|	$\boldsymbol{n + (n-1) + (n-2) + \ldots + 1}$
if(endpoint = vertex || S[j] is on left of P[i] to endpoint)	1
endpoint ← S[j]	1
i ← i+1	1
vertex ← endpoint	1
until endpoint == P[0]	

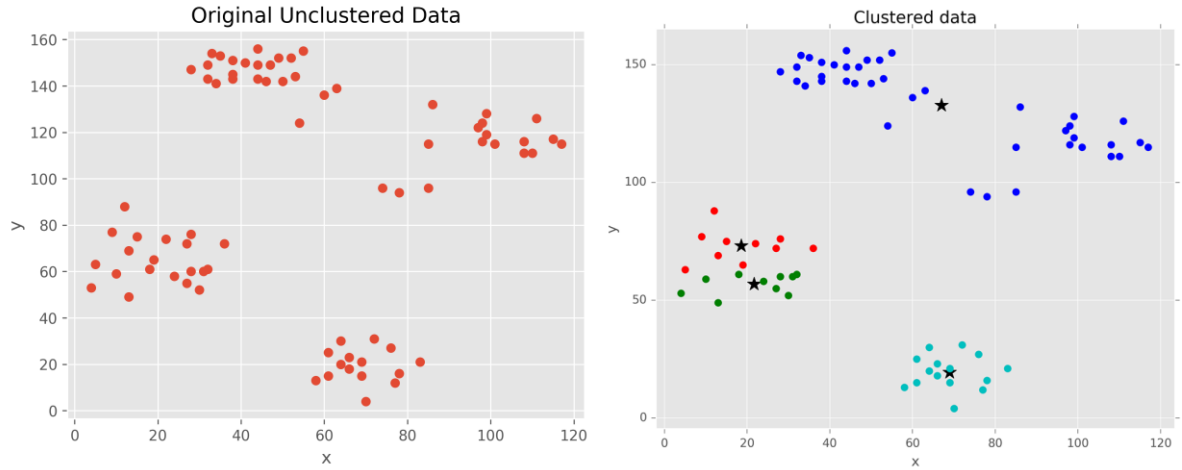
The complexity of the Jarvis March algorithm is $O(nh)$, where n is the number of points given and h is the number of points on the hull. Since algorithms are analyzed in terms of worst case we can see that the worst it can become is $O(n^2)$.

Solutions

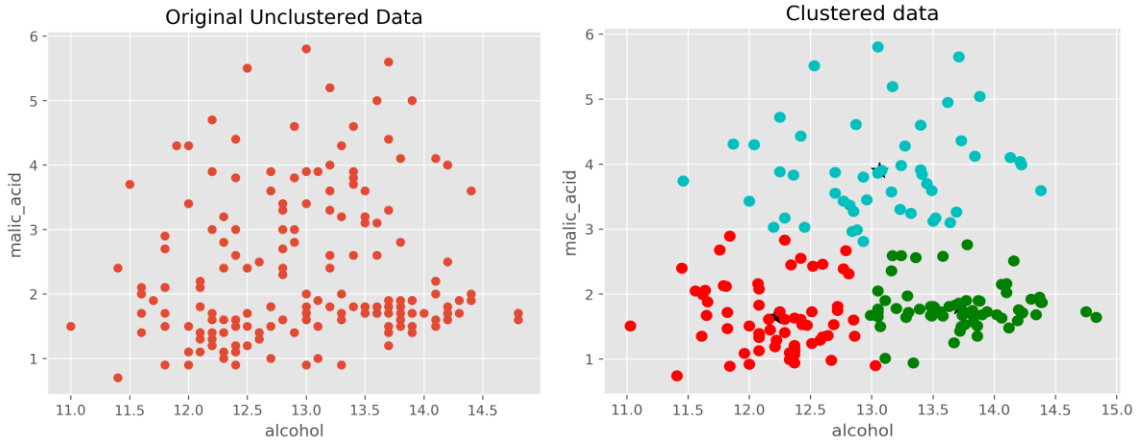
1) K-Means



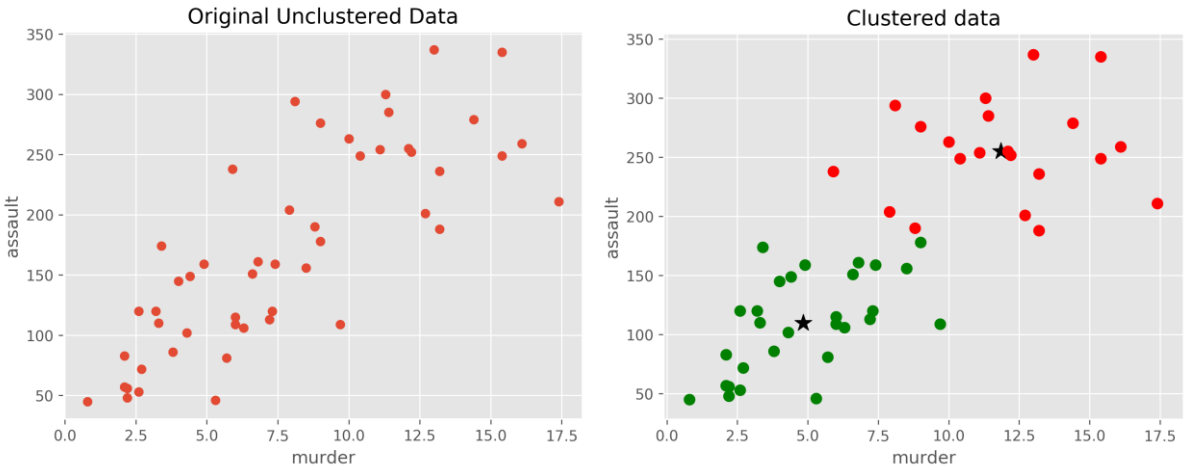
Plots corresponding to exercise 1 -- with 2 clusters



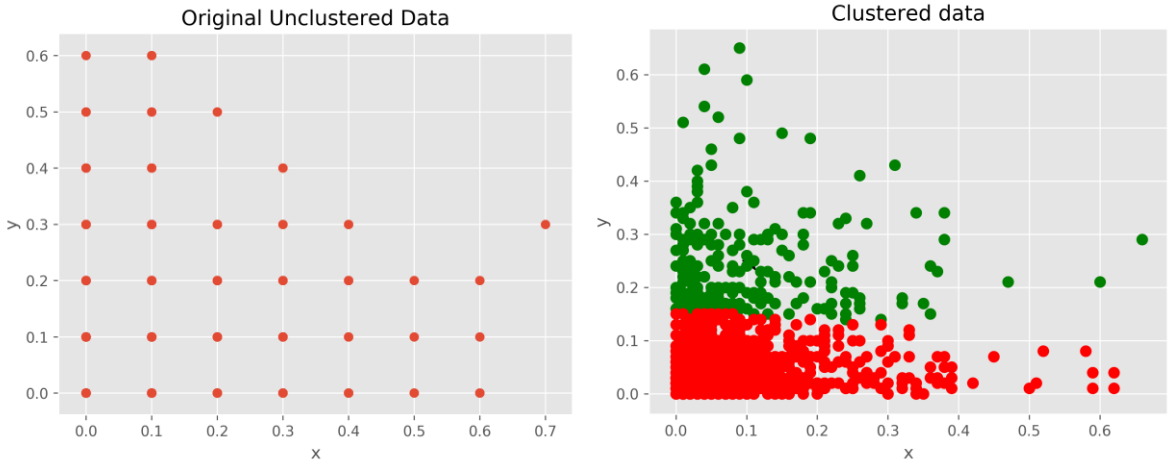
Plots corresponding to exercise 2 -- with 4 clusters



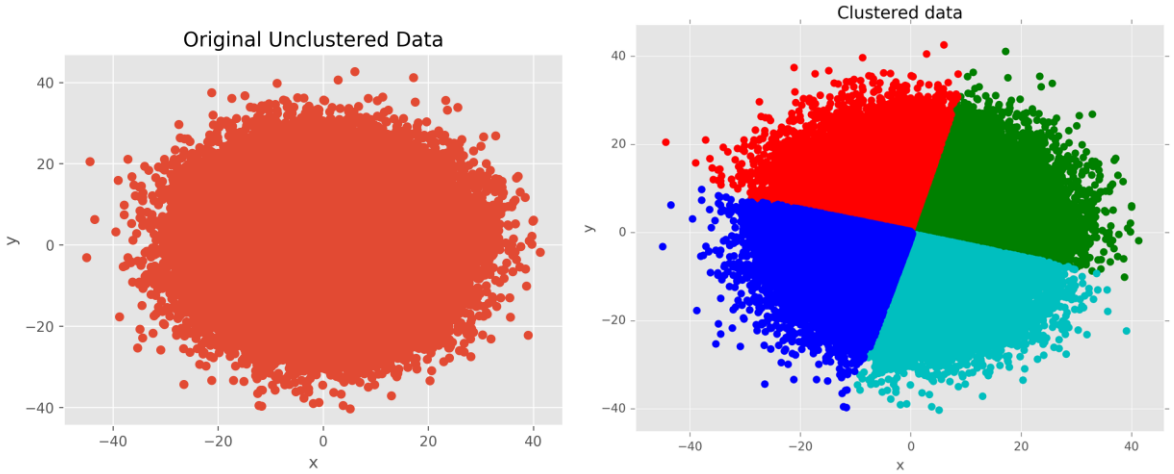
Plots corresponding to exercise 3 -- with 3 clusters



Plots corresponding to exercise 4 -- with 2 clusters

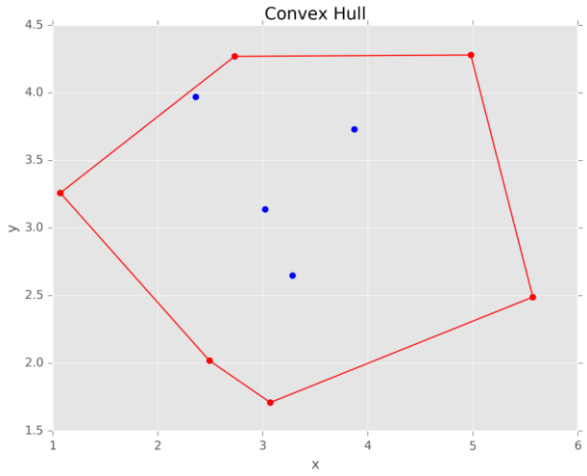


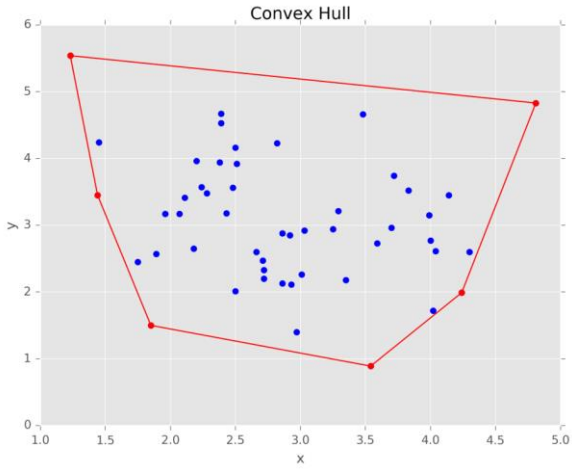
Plots corresponding to exercise 5 -- with 2 clusters



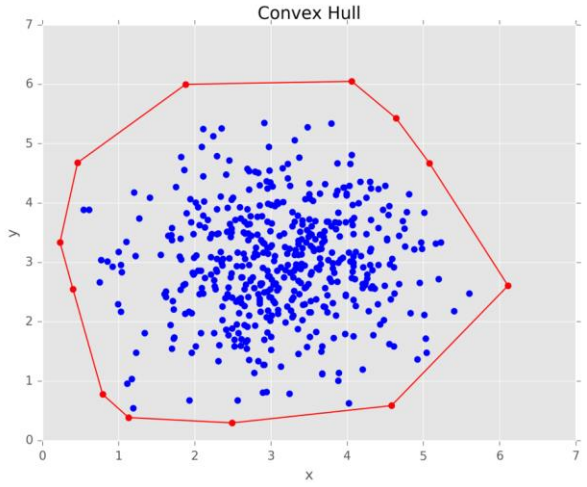
Plots corresponding to exercise 6 -- with 4 clusters

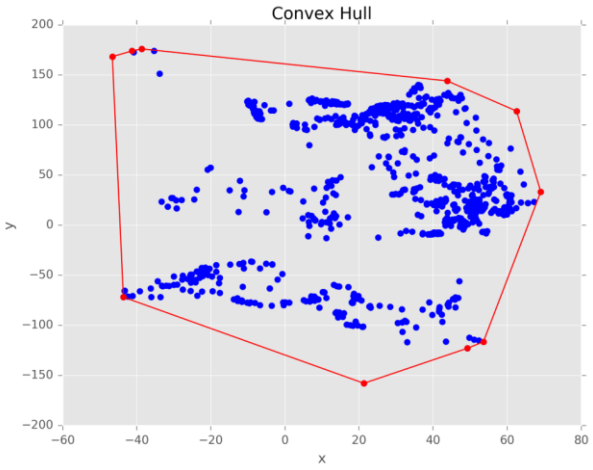
2) Convex Hull



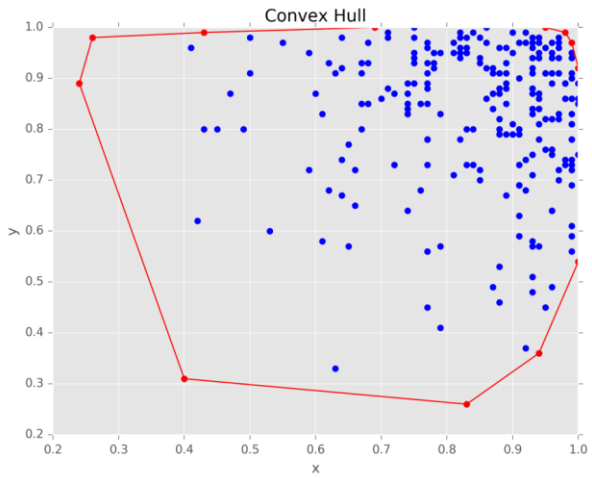


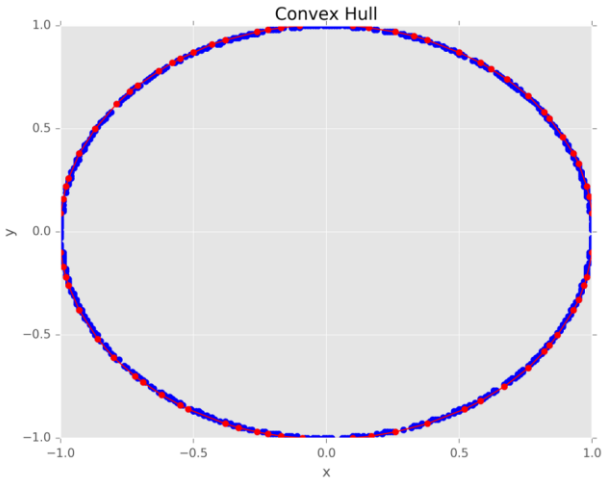
Solutions to Exercise 1 and Exercise 2 respectively





Solutions to Exercise 3 and Exercise 4 respectively





Solutions to Exercise 5 and Exercise 6 respectively

Time Complexity Analysis

K-Means:

Time without plot (just algorithm):

Exercise-1.csv K = 2	Exercise-2.csv K = 4	Exercise-3.csv K= 3	Exercise-4.csv K=2	Exercise-5.csv K=2	Exercise-6.csv K=4
3.219 s	2.445 s	4.749 s	1.034 s	17.784 s	3197.286 s

Total time:

Exercise-1.csv K = 2	Exercise-2.csv K = 4	Exercise-3.csv K= 3	Exercise-4.csv K=2	Exercise-5.csv K=2	Exercise-6.csv K=4
8.223 s	5.848 s	9.248 s	4.079 s	31.816 s	5465.544 s

Convex-Hull:

Time without plot (just algorithm):

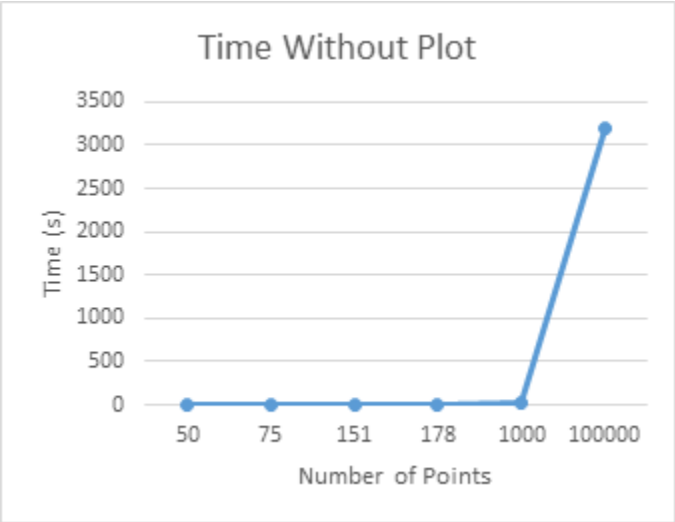
Exercise-1.csv	Exercise-2.csv	Exercise-3.csv	Exercise-4.csv	Exercise-5.csv	Exercise-6.csv
0.0001 s	0.0002 s	0.0480 s	0.0880 s	0.0270 s	0.620 s

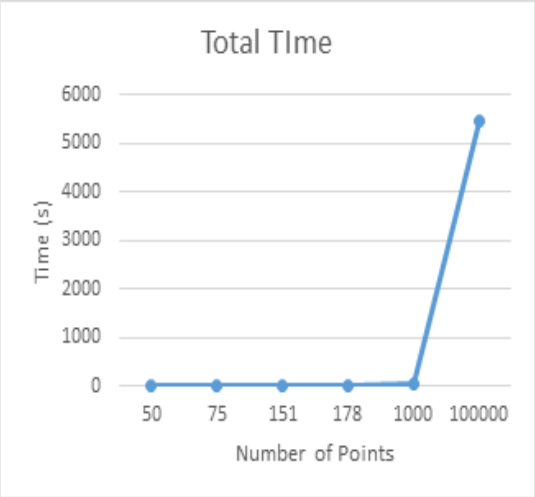
Time with plot:

Exercise-1.csv	Exercise-2.csv	Exercise-3.csv	Exercise-4.csv	Exercise-5.csv	Exercise-6.csv
2.632 s	2.766 s	2.840 s	3.109 s	2.865 s	2.884 s

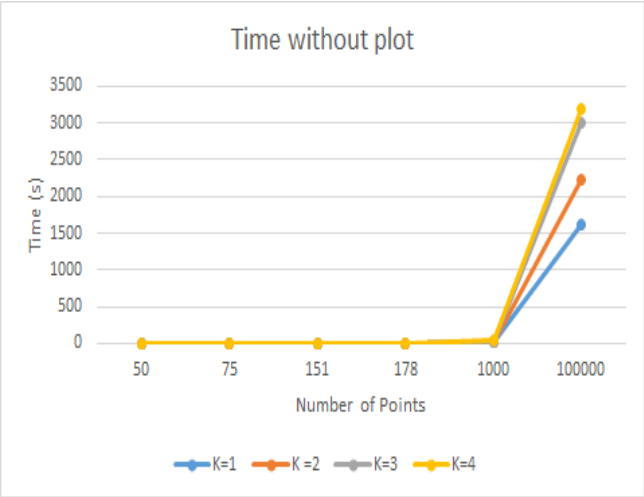
Performance Plots:

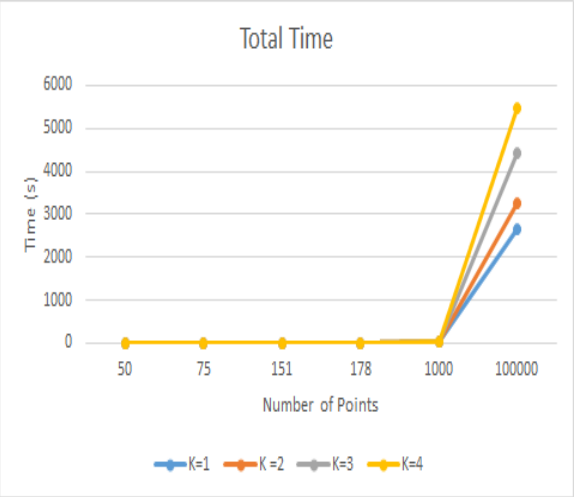
1) K-Means





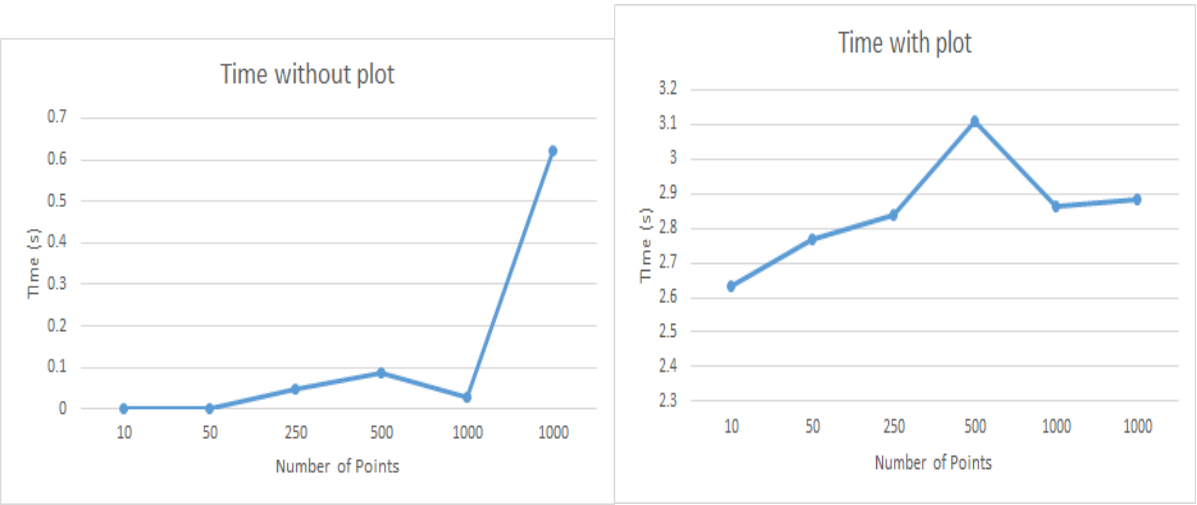
The two plots compares the number of points vs time to compute the results. The first plot is the the time without plot so just testing the algorithm and the second plot is the total time it takes to plot the two graphs. This graph shows an exponential curve because it's gradually increase as you increase the number of points





The two plots compares number of points vs time to compute the results including the k value. The first plot is the the time without plot so just testing the algorithm and the second plot is the total time it takes to plot the two graphs. It also follows an exponential curve. The difference between the above two plots and this is that as you increase the k it takes more time to plot.

2) Convex Hull



The two plots compares number of points vs time to get the results. The first plot is the time without plot so just testing the algorithm alone and second plot is the time with plot. It follows fluctuating curve; as number of points increases, it takes more time to plot the graph but for some data sets with low points it takes longer time because the points are complex, they are positive and negative values in the data file along with them being decimals and very close to each other. So if you have more data points but are all positive numbers that are distinct it will take less time to compute the solutions.

Conclusion:

Learning about various Data Structures and Algorithms while working on this project simultaneously gave us the necessary tools to tackle this project. We were successfully able to complete 2 topics which were K-Means and Convex Hull as a team. With respect to K-Means, we were able to use Lloyd's algorithm along with an implementation of the Linked List data structure. We decided to use a linked list because it was easy to implement since we had done it in tutorial too, and it made sense to use a linked list than any other data structure. We just needed to add and traverse through the points and linked list accomplishes this effectively than a hash table or a tree which would be irrelevant to our problem. This gave us all a good understanding on developing our own Linked List class and implementing it in our solution for the K-Means clustering. For our Convex Hull solution, we were able to use the Gift Wrapping algorithm along with an implementation of ArrayLists data structure. The ArrayList data structure was specifically chosen because as opposed to the basic array, the ArrayList expands in size as data is added to it. Also the ArrayList explicitly shows what object it is storing and in our case the object we stored are "Points" objects. The points object is a class we made to handle coordinates given to us.

All the plot solutions for K-Means came out just like we expected. Using different number of clusters k, the algorithm was able to plot a 1,2,3 or 4 cluster plot corresponding to the number of points n and number of iterations d. The time complexity results were reasonable as the first few exercises did not take much time to complete but as we got to exercise 5 and 6 the time to compute the results did take a spike but was as a result of handling a larger number of points n, which was expected. The plots for the Convex Hull came out perfectly. However we used python to plot it and java to compute the hull points. The cross-programming is something we could have avoided however due to little experience with java's graphical interfaces, we could not plot using java and had to result to python. The results for the Convex Hull were appropriate to the complexity of the algorithm. We observed that the runtime of each code on each of our group members computers was different and researched on that as well. Some reasons why the solutions can take time to generate can be due to performance of the machine that you are running it on, amount of programs that are open during the execution of the program, and other factors such as battery level, hardware specs and much more. These results will vary on each computer it is run on, however the growth in time will be very similar for all solutions, which is why big O notation is provided for us.

Both these topics have given us a good understanding of implementing algorithms and implementing them in the best way possible. Alongside the programming experience it has given us, we also learned the concept of working as a team on a specific code to derive a solution.