Q1: gSpan vs FSG vs Gaston

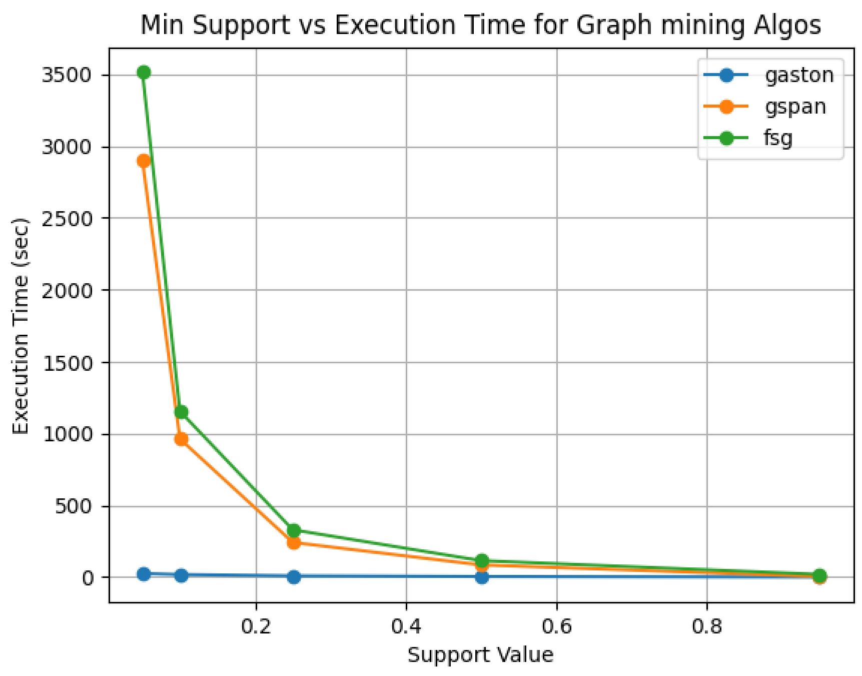


Fig 1 :Gaston vs gSpan vs fsg plot

Growth rate: In the above graph as we can, as the minimum support for the algorithms decreases, time increases exponentially. Which is obvious because chances of search getting pruned are decreasing, algos are moving towards finding frequency for every graph in the database.

Algo speed comparison: Overall order of speed is (fastest to slowest), Gaston>gSpan>FSG

Gaston makes use of one characteristic of practical graph databases that free trees are the most frequent. So, it makes sure that it is mining frequent free trees first.

On the other hand, gSpan looks for "minimum DFS code" to avoid duplicates and mine in the same Depth first order. As backward edges are preferred in rightmost extension to get minimum DFS code, cyclic graphs are possible which may not be that frequent in practical graph databases.

FSG takes the most time, which was expected, as it uses breadth first approach to mine frequent patterns, also it uses merging of frequent graphs to get candidates.

A similar comparison of these three algos has been made in the Gaston paper on different databases.

References: https://liacs.leidenuniv.nl/~nijssensgr/gaston/gaston-april.pdf

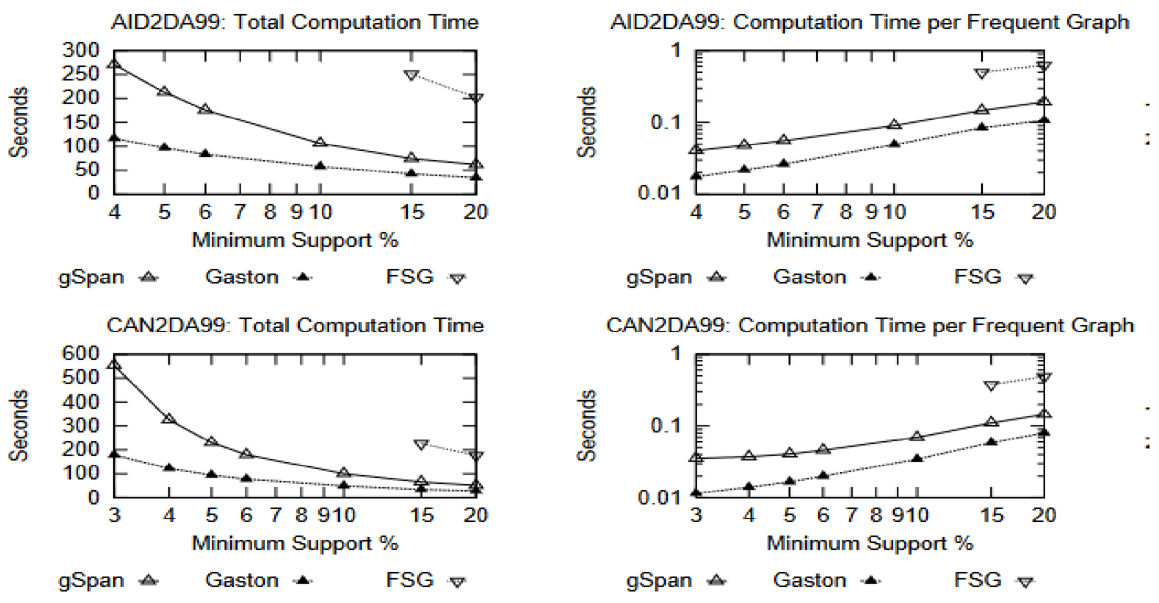


Fig.2: Plot of Comparison in Gaston paper

Q2: K-Means Clustering

K-Means is an algorithm for clustering the data into k clusters. It takes the k data points arbitrarily and makes them as cluster centers and then classifies each point to the cluster center to which the point is closest to and then finds the centroid of the points in the cluster and repeats the process.

Our goal is to find the optimal value of k so that we have the optimal number of clusters in the data. We do this by finding the mean distance from the points to their cluster centers(mean_distance) and comparing it. We compare this by plotting the graph of mean distance vs number of clusters(k).
For each value of k, the ratio of mean_distance and after a certain value of k, we will see insignificant changes (I.e., negligible) in the ratio. We would call this point as "elbow point" in the plot and the value of k at which elbow plot occurs is the optimal value of k.

In our code, we are reading the file which is in ".dat" format and storing the data using pandas library and passing the data to the KMeans function which is available in the library sklearn (scikit-learn) which can efficiently find the clusters. The following is the sample statement which we use for generating the clusters

```
kmeans = KMeans(n_clusters=5, n_init=20, random_state=42)
```

It means that we are clustering the data into 5 clusters (n_clusters=5) and running it for 20 iterations I.e., first, we are generating 5 clusters and then finding their cluster centers, and using these new centers, we will find the clusters again. We repeat this process for 20 iterations. We have declared random_state as 42 because we generate the points at random and for us to reproduce the same thing again if done again. We are then adding this whole model to variable "kmeans"

```
kmeans.fit(data)

distances = kmeans.transform(data)

mean_distance = distances.min(axis=1).mean()

mean_distances.append(mean_distance)
```

Then, we are giving our data to the kmeans model and then we are finding the distance of the point from every cluster centroid. The distances store the data from every point to every cluster centroid in a 2-D array. In this, we calculate the Euclidean distance between the points. Then, we find the minimum distance of a point to the centroids (the centroid to which the point is closest to), store the information as a 1-D matrix and find the mean of all those minimum distances and then add them to a mean_distances matrix. After this, using this data, we plot the graph of the mean_distances vs the k value and using this data, we find the elbow_point and the optimal value of k from the plot as shown below.

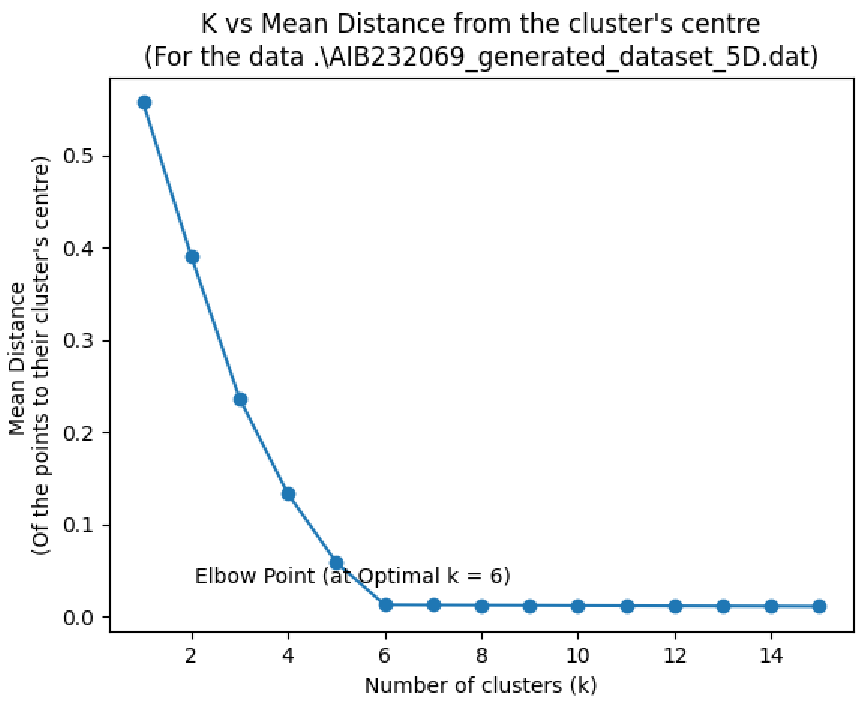
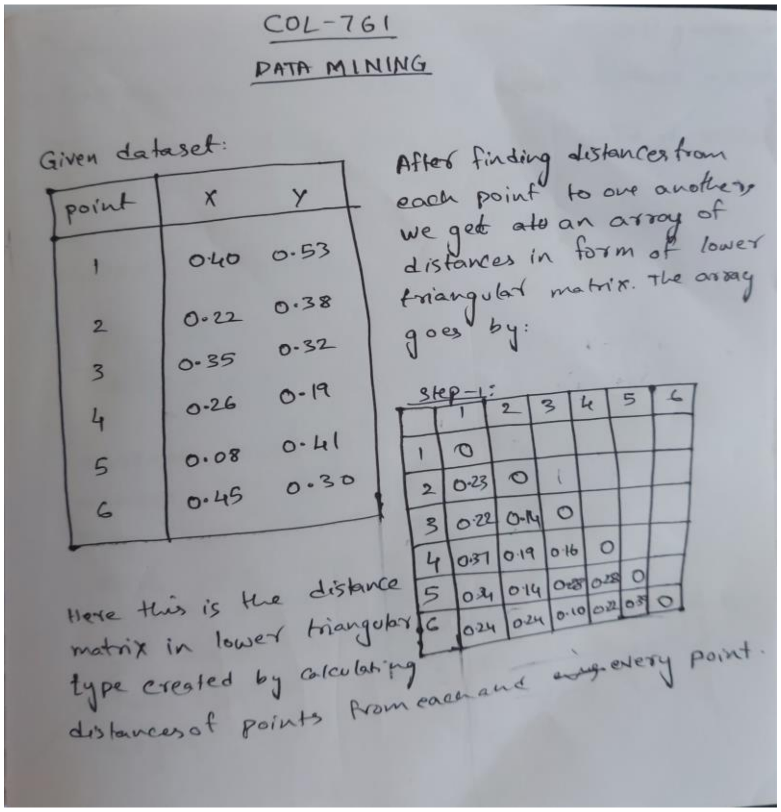


Fig.3: Plot of Mean Distance vs Number of clusters(k)

We have plotted the graph for the dataset “AIB232069_generated_dataset_5D.dat” and have found the “optimal value of k=6”.

Q3 Single Linkage Clustering

Draw Dendrogram for single linkage clustering



COL-761

DATA MINING

Given dataset:

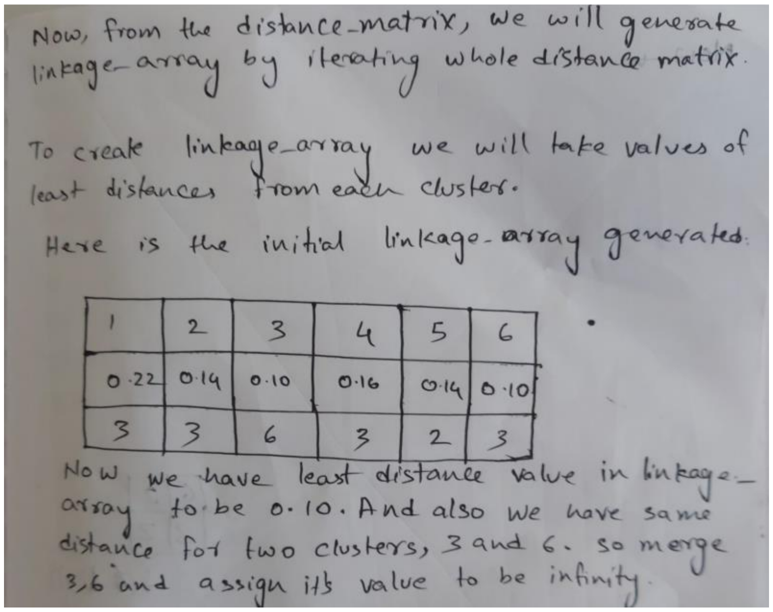
point	X	Y
1	0.40	0.53
2	0.22	0.38
3	0.35	0.32
4	0.26	0.19
5	0.08	0.41
6	0.45	0.30

After finding distances from each point to one another we get ~~ato~~ an array of distances in form of lower triangular matrix. The array goes by:

step-1:

	1	2	3	4	5	6
1	0					
2	0.23	0				
3	0.22	0.14	0			
4	0.37	0.19	0.16	0		
5	0.34	0.14	0.28	0.28	0	
6	0.24	0.24	0.10	0.22	0.3[illegible]	0

Here this is the distance matrix in lower triangular type created by calculating distances of points from each and ~~[illegible]~~ every point.



Now, from the distance_matrix, we will generate linkage_array by iterating whole distance matrix.

To create linkage_array we will take values of least distances from each cluster.

Here is the initial linkago-array generated.

1	2	3	4	5	6
0.22	0.14	0.10	0.16	0.14	0.10
3	3	6	3	2	3

Now we have least distance value in linkage_ array to be 0.10. And also we have same distance for two clusters, 3 and 6. so merge 3,6 and assign it's value to be infinity.

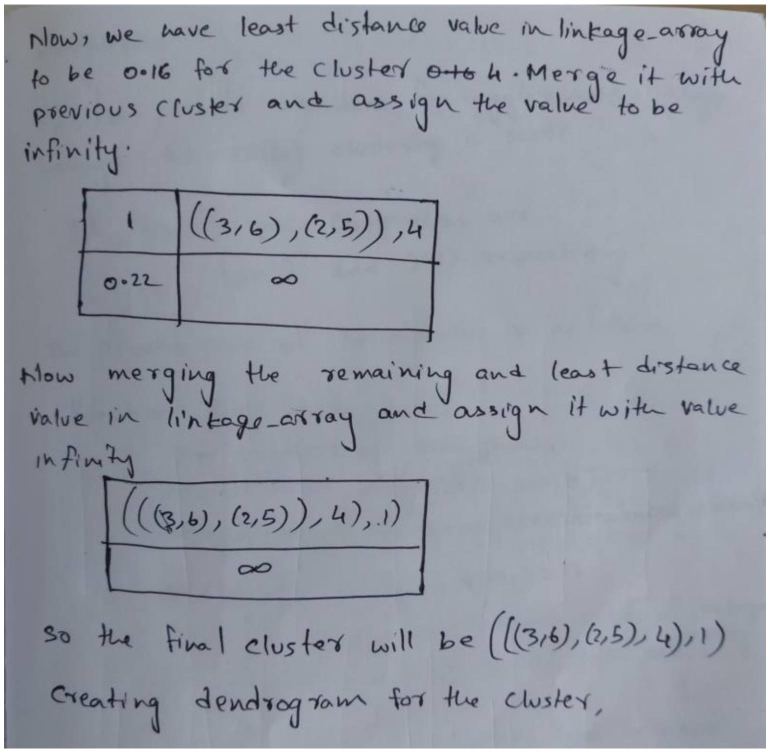
1	2	(3,6)	4	5
0.22	0.14	∞	0.16	0.14

Now we have the least distance value in linkage-array is 0.14. And also we have two clusters 2,5. so merge 2,5. But this time ~~leave t~~ assign the value to be 0.14 and not infinity. Because since weare doing hierarchical clustering, we need to merge all clusters one by one.! And this time we created new cluster without merging with existing one. So we need to merge it in next move and assign it's value to be 0.14 this time.

1	(2,5)	3,6	4
0.22	∞	∞	0.16

Now merging both clusters and assigning it's value to be infinity.

1	(3,6),(2,5)	4
0.22	∞	0.16



Now, we have least distance value in linkage_array to be 0.16 for the cluster ~~0 to~~ 4. Merge it with previous cluster and assign the value to be infinity.

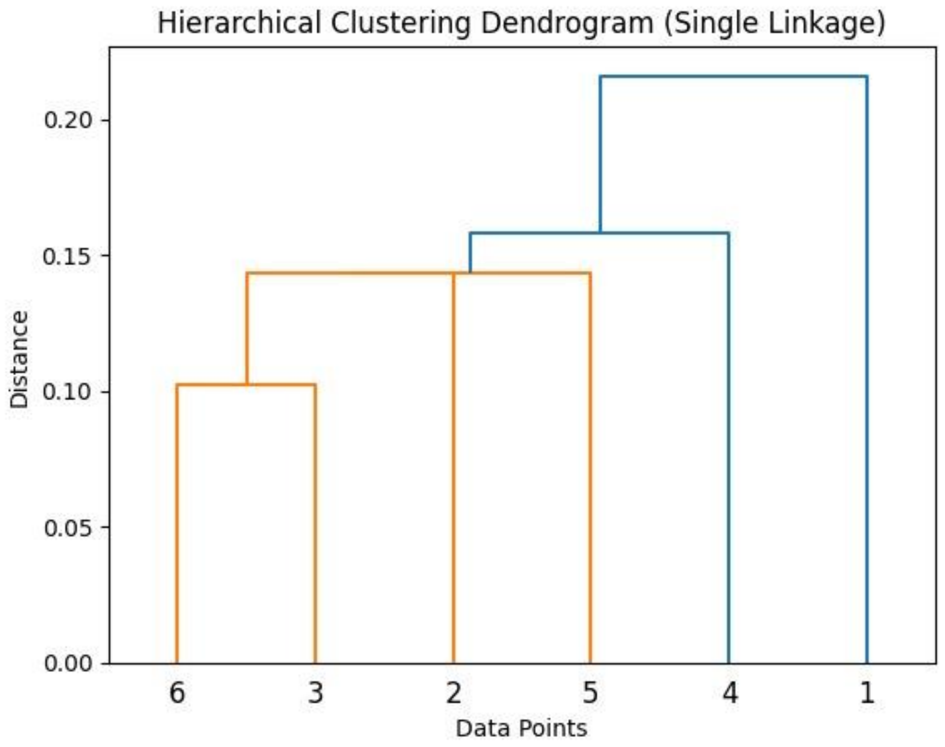
1	((3,6),(2,5)),4
0.22	∞

Now merging the remaining and least distance value in linkage_array and assign it with value infinity

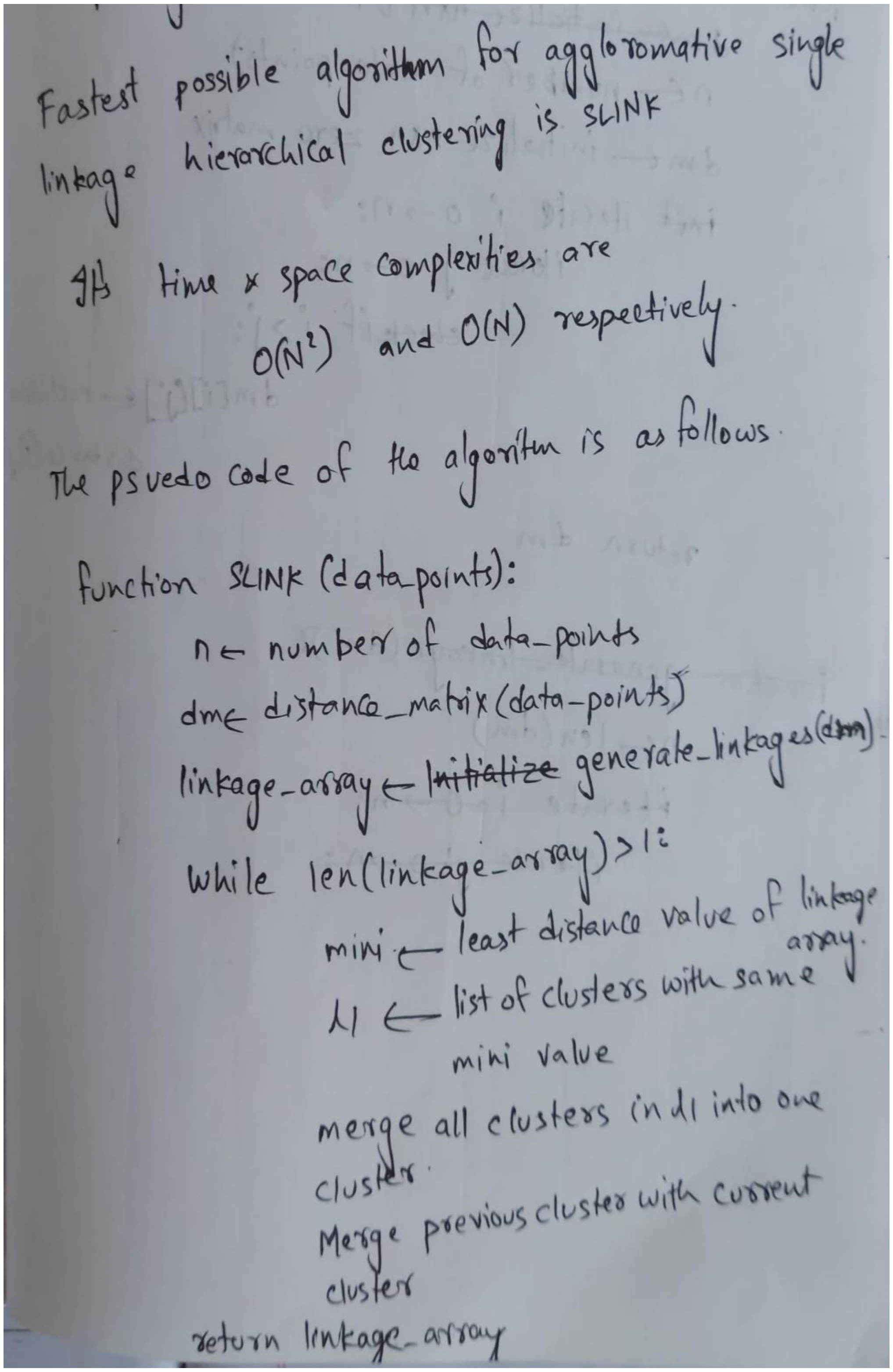
(((3,6),(2,5)),4),1)
∞

So the final cluster will be (((3,6),(2,5)),4),1)

Creating dendrogram for the cluster,



Fastest possible algorithm for single linkage clustering.

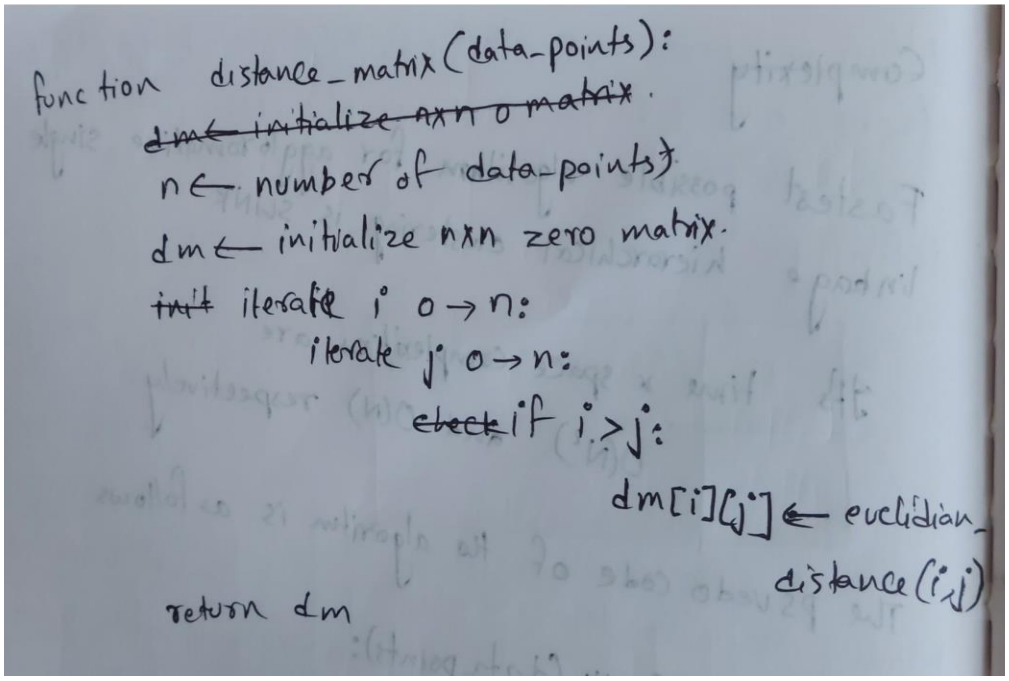


Fastest possible algorithm for agglomative single linkage hierarchical clustering is SLINK

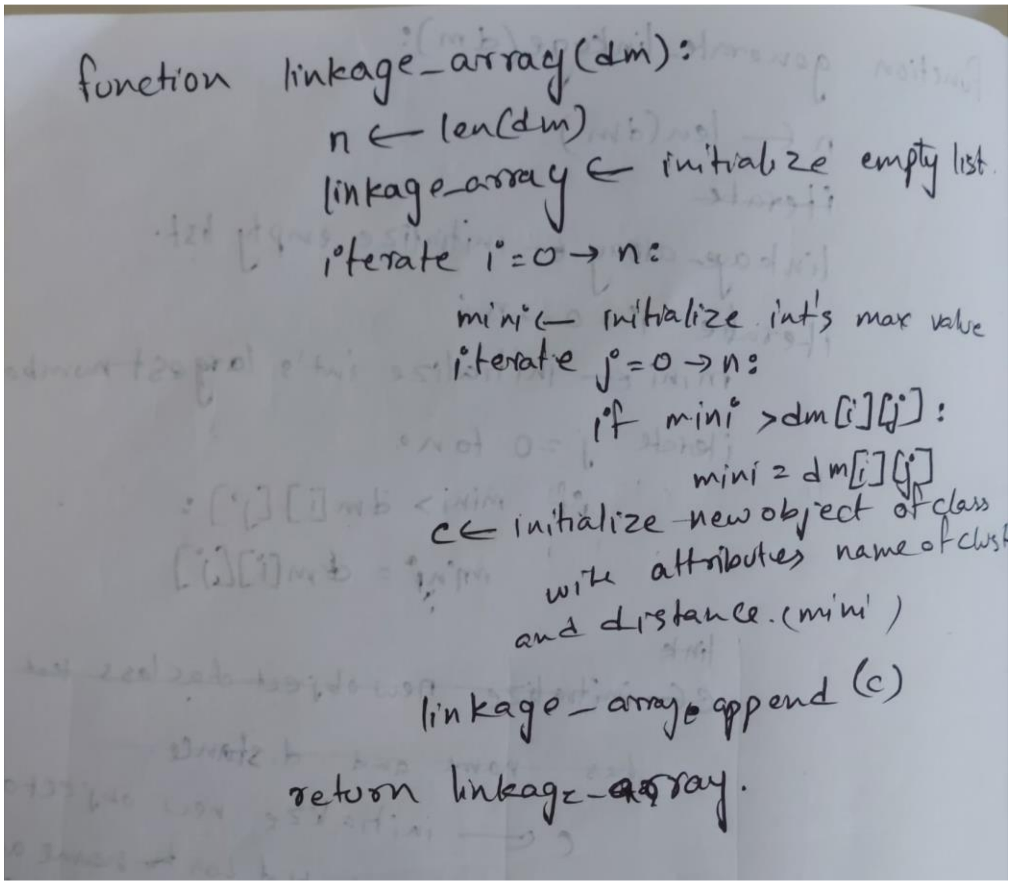
Its time & space complexities are
O(N^2) and O(N) respectively.

The psuedo code of the algoritm is as follows.

```
function SLINK (data_points):
        n ← number of data_points
        dm ← distance_matrix(data_points)
        linkage_array ← ~~Initialize~~ generate_linkages(dm).
        While len(linkage_array) > 1:
                mini ← least distance value of linkage array.
                l1 ← list of clusters with same
                        mini value
                merge all clusters in l1 into one
                cluster.
                Merge previous cluster with current
                cluster
        return linkage_array
```



```
function distance_matrix(data-points):
    ~~dm← initialize nxn 0 matrix~~.
    n← number of data-points}
    dm← initialize nxn zero matrix.
    ~~init~~ iterate i 0→n:
        iterate j 0→n:
            ~~check~~ if i>j:
                dm[i][j] ← euclidian_
                              distance(i,j)
    return dm
```



```
function linkage_array(dm):
    n ← len(dm)
    linkage_array ← initialize empty list.
    iterate i=0→n:
        mini ← initialize int's max value
        iterate j=0→n:
            if mini >dm[i][j]:
                mini = dm[i][j]
        c← initialize new object of class
            with attributes name of clust
            and distance.(mini)

        linkage_array.append (c)

    return linkage_array.
```

Complexity Analysis:

We have 3 functions. Out of which SLINK is main function. And remaining two functions distance_matrix and generate_linkage are helping functions.

The time and space complexities of above functions are as follows:

Function Name	SC	T C
SLINK	$O(N)$	$O(N^2)$
distance_matrix	$O(N^2)$	$O(N^2)$
generate_linkage	$O(N)$	$O(N^2)$