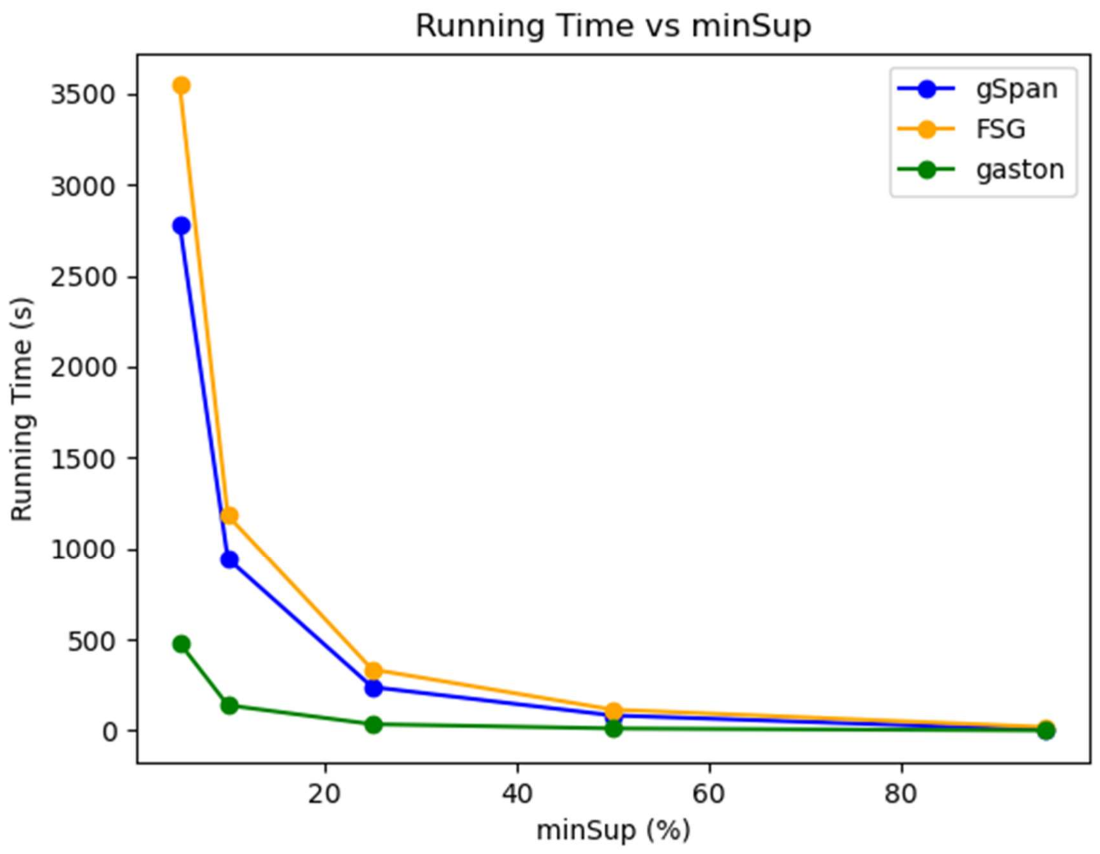
Data Mining (COL 761)

Assignment: 2

Team: Data Crashers		
Ajay Tomar	Anshul Patel	Pulkit Singal
2023AIB2071	2023AIB2072	2023AIB2064

Question 1:

The plot for running times for gSpan, FSG and gaston algorithms for minSup = 5%, 10%, 25%, 50% and 95% is as shown below:



The running time for different algorithms is shown in the table below:

minSup (%)	Running Time (s)		
	gSpan	FSG	gaston
5	2782.96	3548.02	479.62
10	940.74	1182.88	138.46
25	238.66	335.93	35.18
50	83.05	115.00	11.98
95	4.21	19.53	0.72

For a particular algorithm, it is observed that the running time decreases with increasing value of minSup. This is because the data follows a power-law distribution, i.e., much of the data has very low support.

For a particular value of minSup, it is observed that the running time follows the following trend:

$$\text{gaston} < \text{gSpan} < \text{FSG} \qquad \text{....(1)}$$

gaston is fastest because it uses the "quick-start principle", exploiting the fact that the various sub-structures are contained in each other. In the search for sub-structures, first paths are considered, then paths are transformed into trees and finally, trees are transformed into graphs.

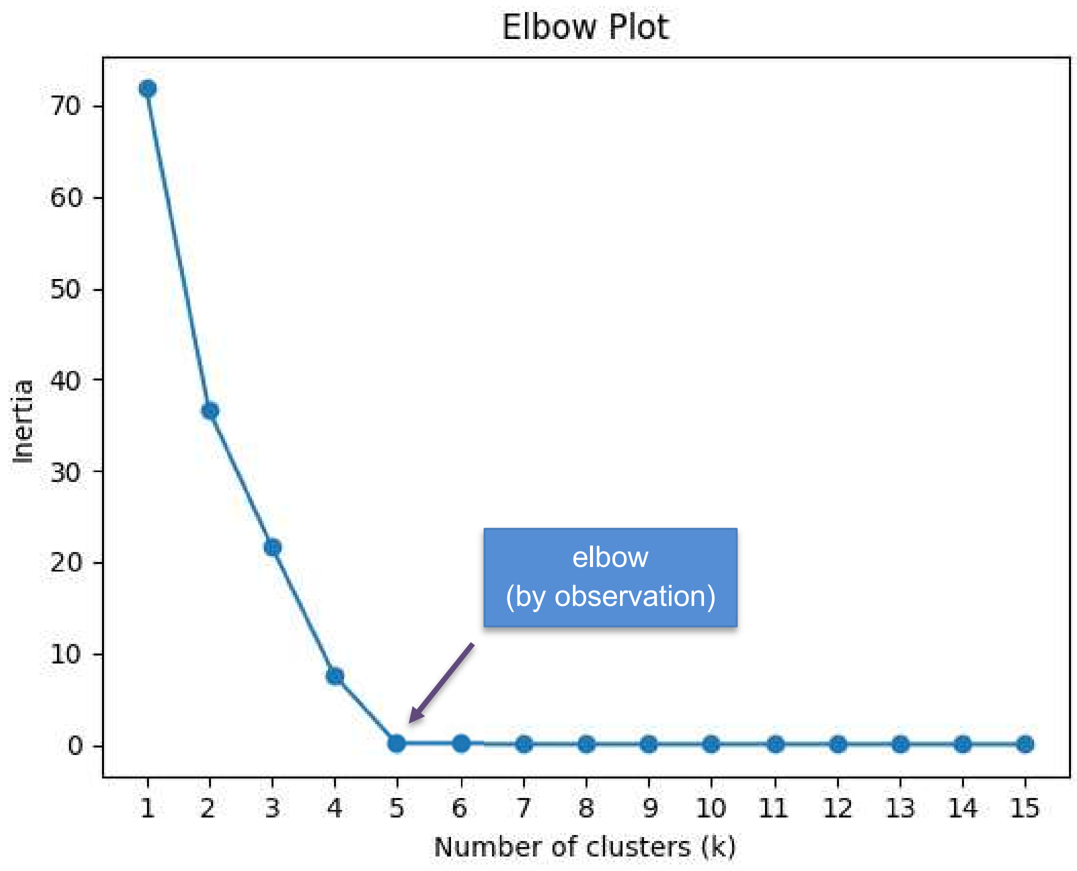
Next is gSpan, which uses a depth-first approach. It is faster than FSG, as it avoids redundant candidate generation and graph isomorphism testing.

FSG is the slowest, which uses a breadth-first approach. It generates all possible candidates, many of which are redundant. It also performs graph isomorphism testing (which takes exponential time, although not proven to be NP-complete) for the candidate generation and pruning steps.

In the worst case, all three algorithms grow exponentially. However, for the above-mentioned reasons, the average running time order (for a particular value of minSup) for the different algorithms follows the trend in equation (1).

Question 2:

Dimension = 6
Suitable value of k = 5 (by observation)



Question 3:

Step 1: Initially we have six clusters C1 (P1), C2 (P2), C3 (P3), C4 (P4), C5 (P5), C6 (P6). The below table shows the data points corresponding to the six clusters as given in Q3:

Cluster No.	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

Step 2: The distance between each pair of clusters is calculated, which is tabulated below (arranged in increasing order of distance values):

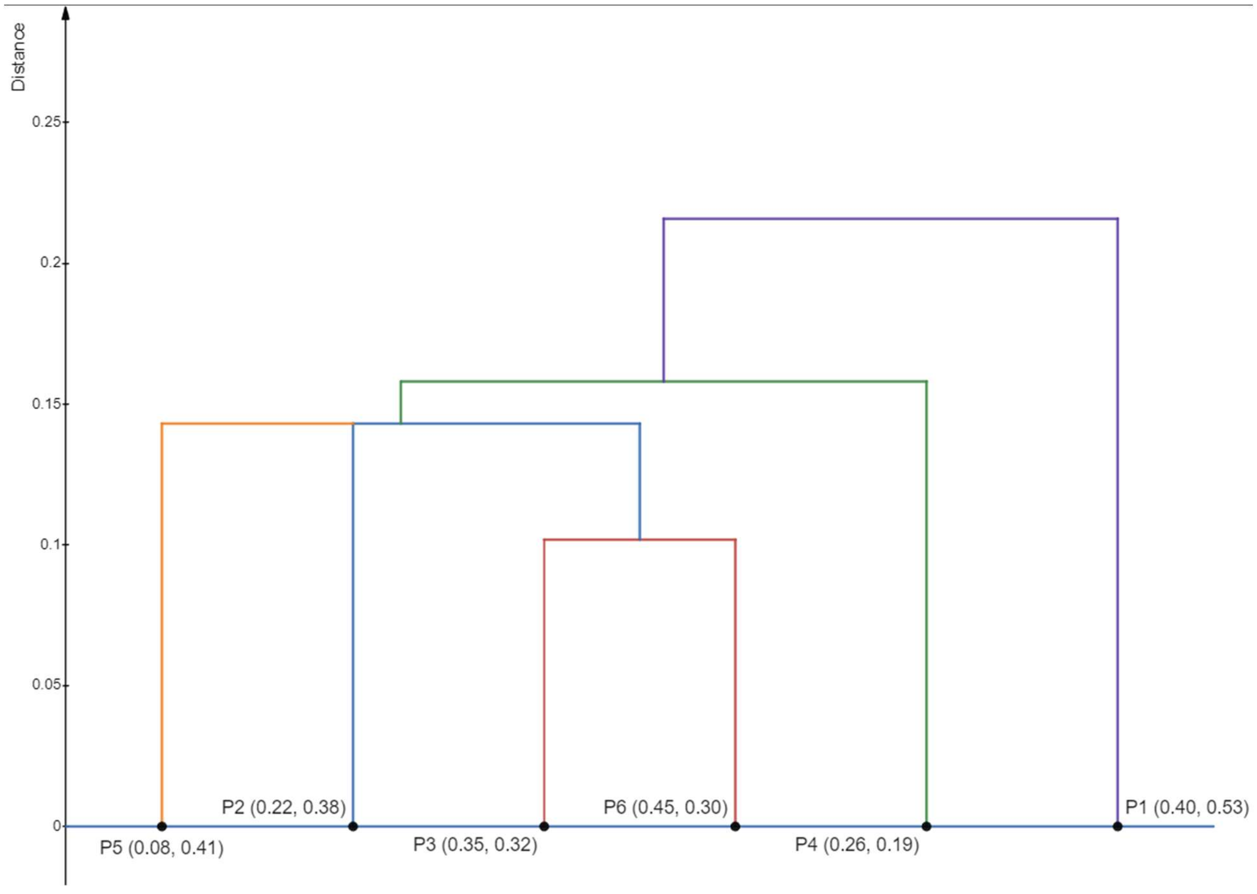
Cluster A	Cluster B	d (P1, P2)
3	6	0.10198
2	3	0.143178
2	5	0.143178
3	4	0.158114
2	4	0.194165
1	3	0.21587
4	6	0.219545
1	2	0.234307
1	6	0.235372
2	6	0.243516
4	5	0.284253
3	5	0.284605

1	5	0.34176
1	4	0.367696
5	6	0.386005

Step 3: The pair of clusters that have the minimum (single linkage) distance between them (C3 and C6) are merged to form a new cluster C7, and connected in the Dendogram, with the height on the Y-Axis taken as the distance between the two clusters (0.10198). Now we have 5 clusters remaining: C1 (P1), C2 (P2), C4 (P4), C5 (P5) and C7 (P3, P6).

Step 4: Again, the pair of clusters that have the minimum single linkage distance between them (C2 and C7) are merged to form a new cluster C8, and connected in the Dendogram, with the height on the Y-Axis taken as the distance between the two clusters (0.143178). Now we have 4 clusters remaining: C1 (P1), C4 (P4), C5 (P5) and C8 (P2, P3, P6).

Step 5: The above step is repeated till all the points belong to the same cluster. Hence the Dendogram gets completed.



The complexity of the fastest possible algorithm is O(n^2). The pseudocode for the algorithm is given below:

```
C[] ← All point IDs                                                    O(n)
D[i][j] ← distance (i, j)                                              O(n²)
Av[i] ← nearest neighbor (i)                                           O(n²)
Ad[i] ← distance (i, Av[i])                                            O(1)

function SINGLE-LINKAGE (clusters C)                                   O(n²)
      id ← index (min(Ad[]))                                           O(n)
      remove i, Av[i] from C[]                                         O(1)
      add cluster (i, Av[i]) to C[]                                    O(1)
      for k in range (n):                                              O(n)
            D[id][k], D[k][id] = min (D[id][k], D[Av[id]][k])          O(1)
      end for
      remove row Av[id], column Av[id] from D                          O(n)
      Ad[id] ← min (D[id][m]), d != m, m = 1, 2, ...n                  O(n)
      Ad[Av[id]] ← inf                                                 O(1)
      Av[id] ← index (min(D[id][m])), d != m, m = 1, 2, ...n           O(1)
      for c in C:                                                      O(n)
            if Av[c] = id or Av[c] = Av[id]:                           O(1)
                  Av[c] ← id                                           O(1)
            end if
      end for
      if size (C) = 1                                                  O(1)
            return
      end if
      return SINGLE-LINKAGE (C)
```

The above pseudocode shows the corresponding time complexity on the right side of each step. From the above analysis, it is evident that the time complexity of the above algorithm is O(n^2).

Reference for Q3:

R. Sibson, SLINK: An optimally efficient algorithm for the single-link cluster method, *The Computer Journal*, Volume 16, Issue 1, 1973, Pages 30–34, https://doi.org/10.1093/comjnl/16.1.30