By Moses Mbabaali and taught by Prof. Alesio Merlo and Mr. Davide Caputo.

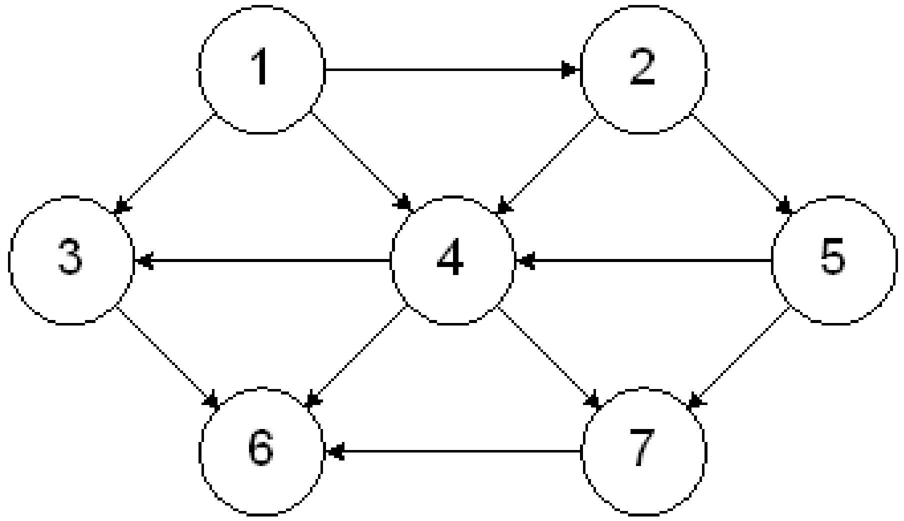
Graph data anonymization with K-degree Anonymity

What is a graph?

- This is a data structure with nodes and edges, G=(V,E).
- Often the nodes will be denoted as V and the edges as E.
- A graph can be directed or undirected.
- A directed graph has direction where it points and undirected graph is can travel either direction.
- A directed graph can be called a digraph.
- Graphs can be weighted or unweighted.
- Graphs are used very much in computer science to store data for example and now there are many variants of graph based databases like neo-4j and others.

Graph types. (Directed Graph)

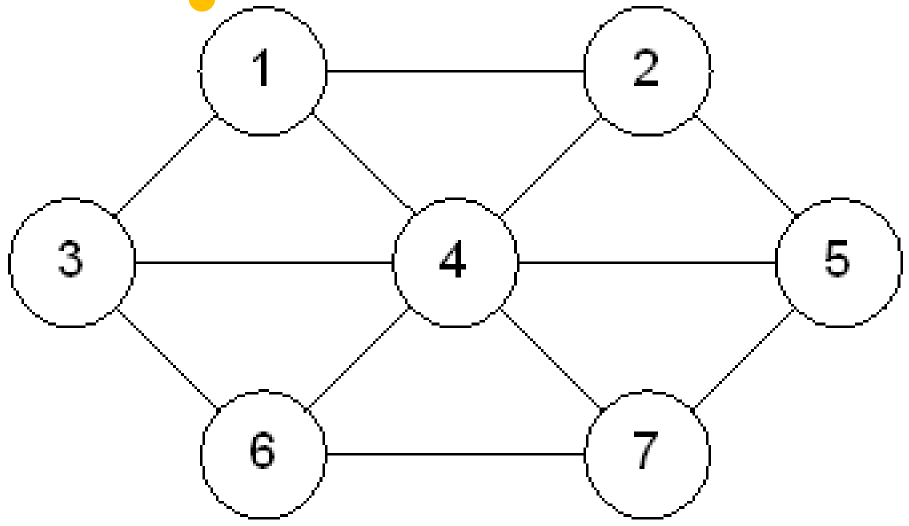
- Directed Graph – Has direction.
- The graph on the side has vertices v = { 1,2,3,4,5,6,7}
- Out degrees are {3,2,1,3,2,0,1}
- In degrees are {0,1,2,3,1,3,2}
- For a directed graph you can have both incoming edges and outgoing edges.



Graph types. (Undirected Graph)

- And undirected graph on the other hand has no direction. You can move from one side of the edge to the other.
- In this case the degrees for each of the nodes v= {1,2,3,4,5,6,7} is neither in or out.
- Degrees for this case will be

d= {3,3,3,6,3,3,3}



Graph, Use cases.

- Graphs are used to store data for social networks.
- Homeland security data.
- Electrical grids.
- Financial institutions.
- Genetics data
- Basically they are good at storing complex data.

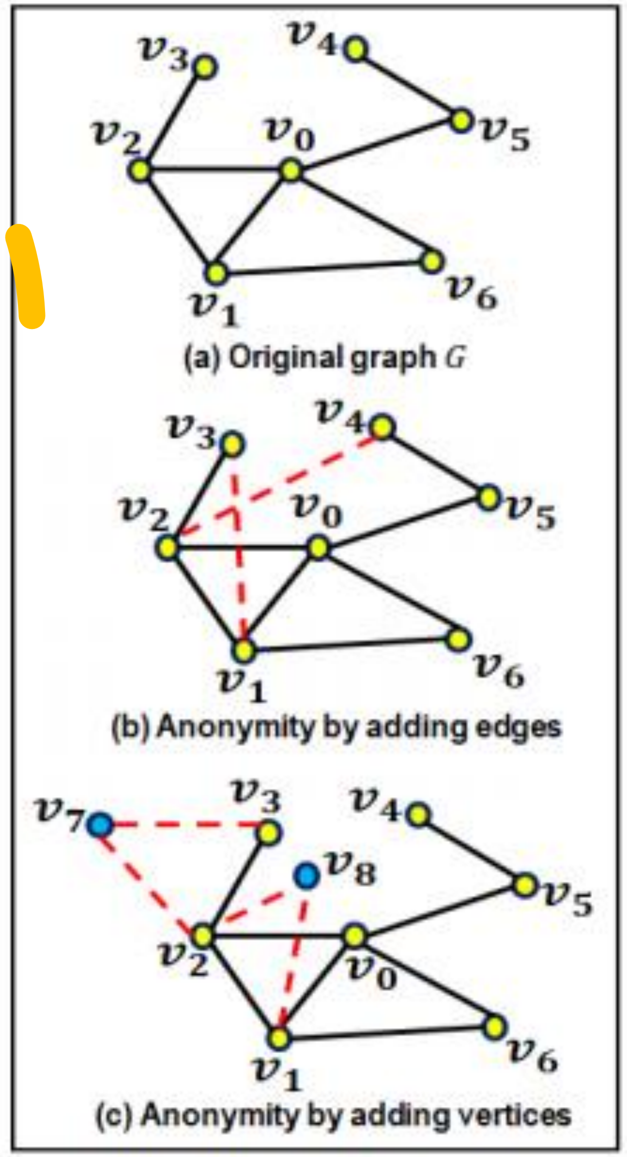
Given this data, how can it be accessed anonymously?

- There are several ways this can be done.

1. Graph modification. (Add or delete edges, switch edges, k-degree)
2. Graph generalization or clustering. (Degree vertex clustering, edge clustering etc)
3. Privacy aware graph computation. (Query analysis and evaluation)
4. Differential Privacy approaches. (Edge counting queries etc).
5. Hybrid anonymity methods. (Vertex edge clustering).

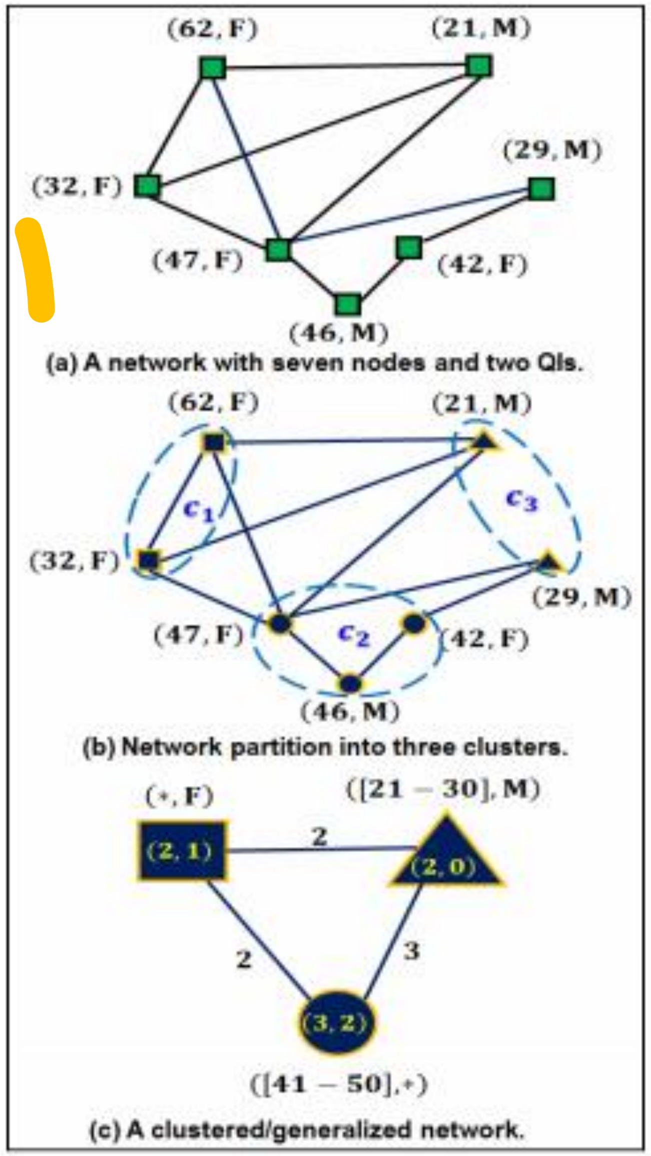
Adding Edges to a graph example.

- Given a graph you can add edges to it or vertices as a way of anonymizing it.



Clustering example of anonymization.

- In this case you can put nodes that have similar characteristics together.
- In a way you will get a generalized network.

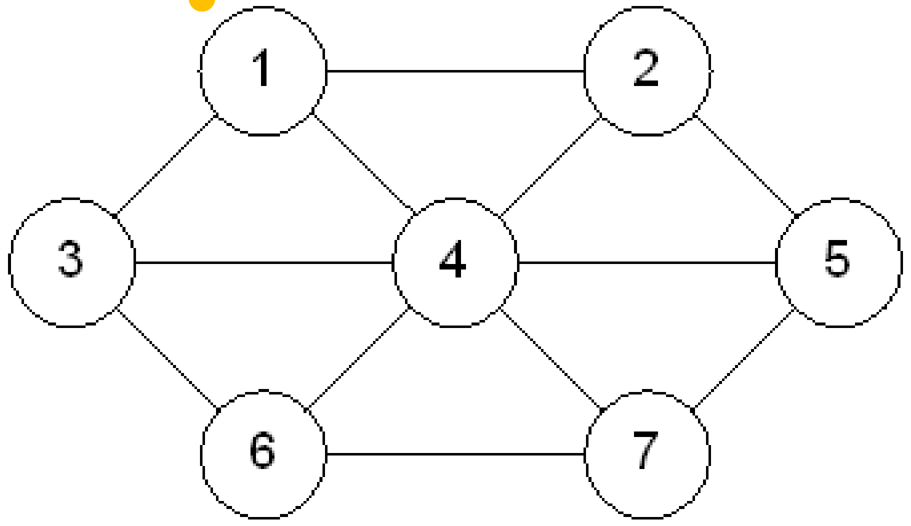


K-Degree Anonymity.

- Given a vector g = [1,1,1,1,2,2,2,3,3,3,4,4,4,4,4]
- The vector is said to be k-anonymous if every value x appears at least k times. In our case above the vector is 3 anonymous because every value appears at least 3 times.
- Now we are going to transfer the same concept to graph degrees.

K-Degree Anonymity.

- Given degrees D= {3,3,3,6,3,3,3} for the graph.
- The graph is said to be 1-Degree anonymous because every value appears at least once.
- If the 6 was out then it would be 6-degree anonymous.

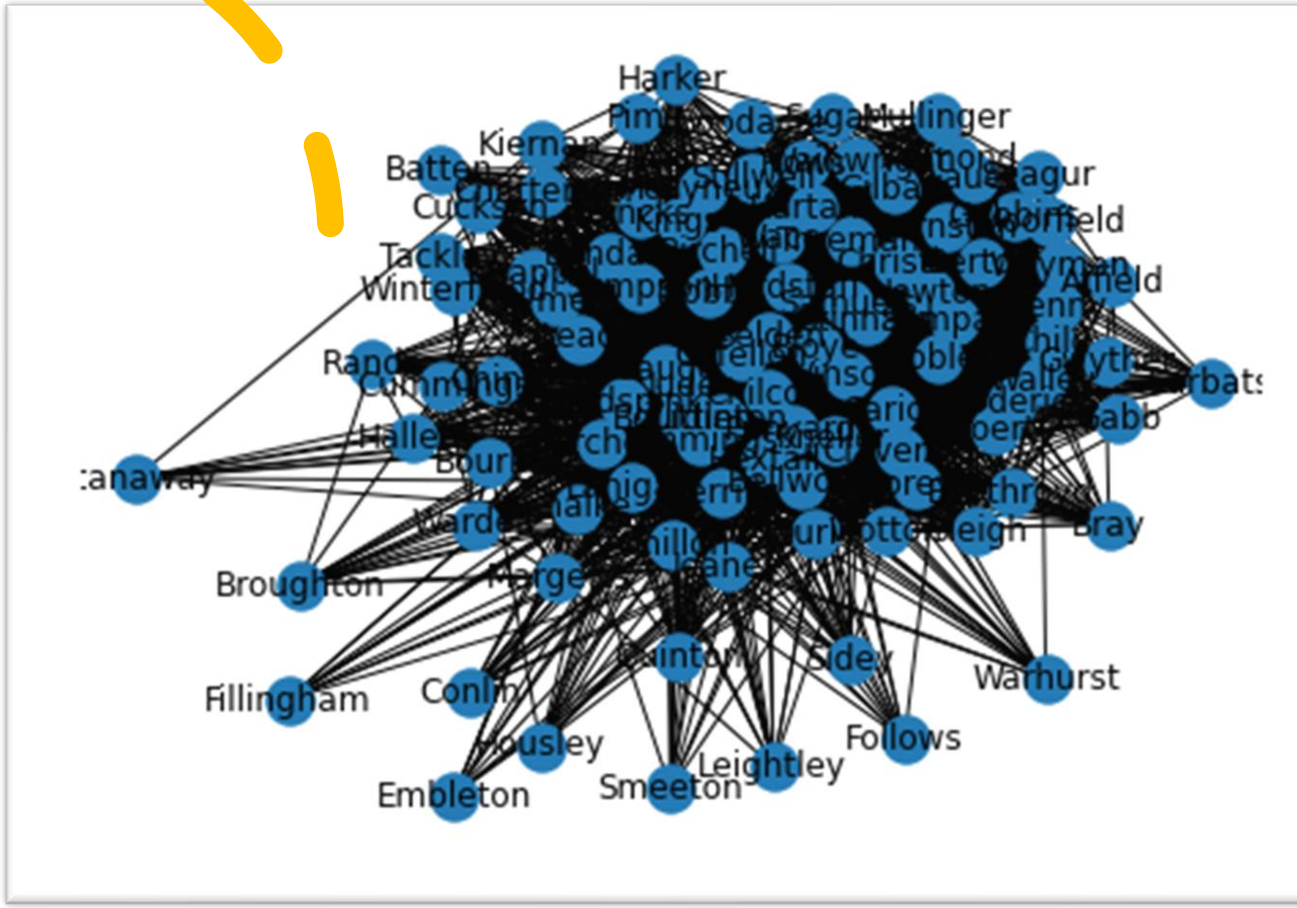


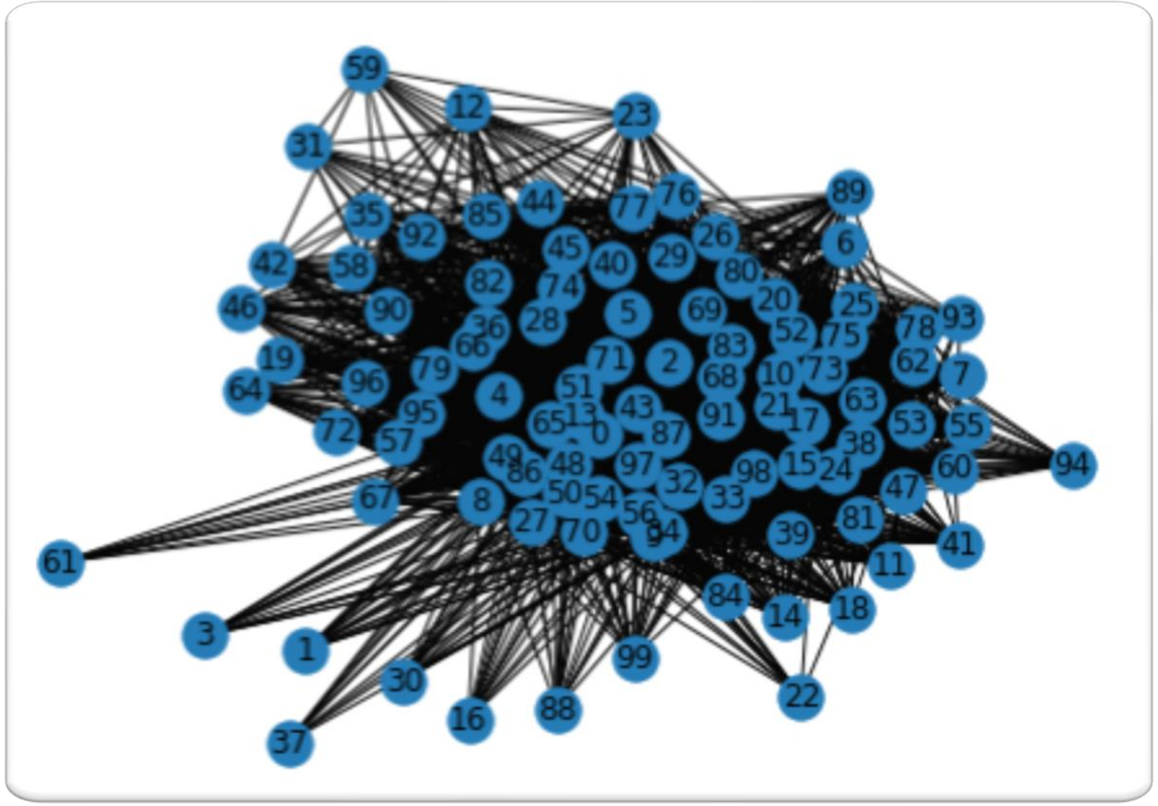
K-Degree Anonymity.

- The same similar concept was applied to the project based on Liu and Terzi’s paper.
- Two algorithms were suggested, using the greedy approach and also using the dynamic approach.
- And the following results were realised.

Greedy Algorithm. (Graph)

- With the greedy algorithm with a k-degree of 4. The following results were realized based on the friends-of-friends dataset.
- The graph on the right shows the original graph based on the degrees.



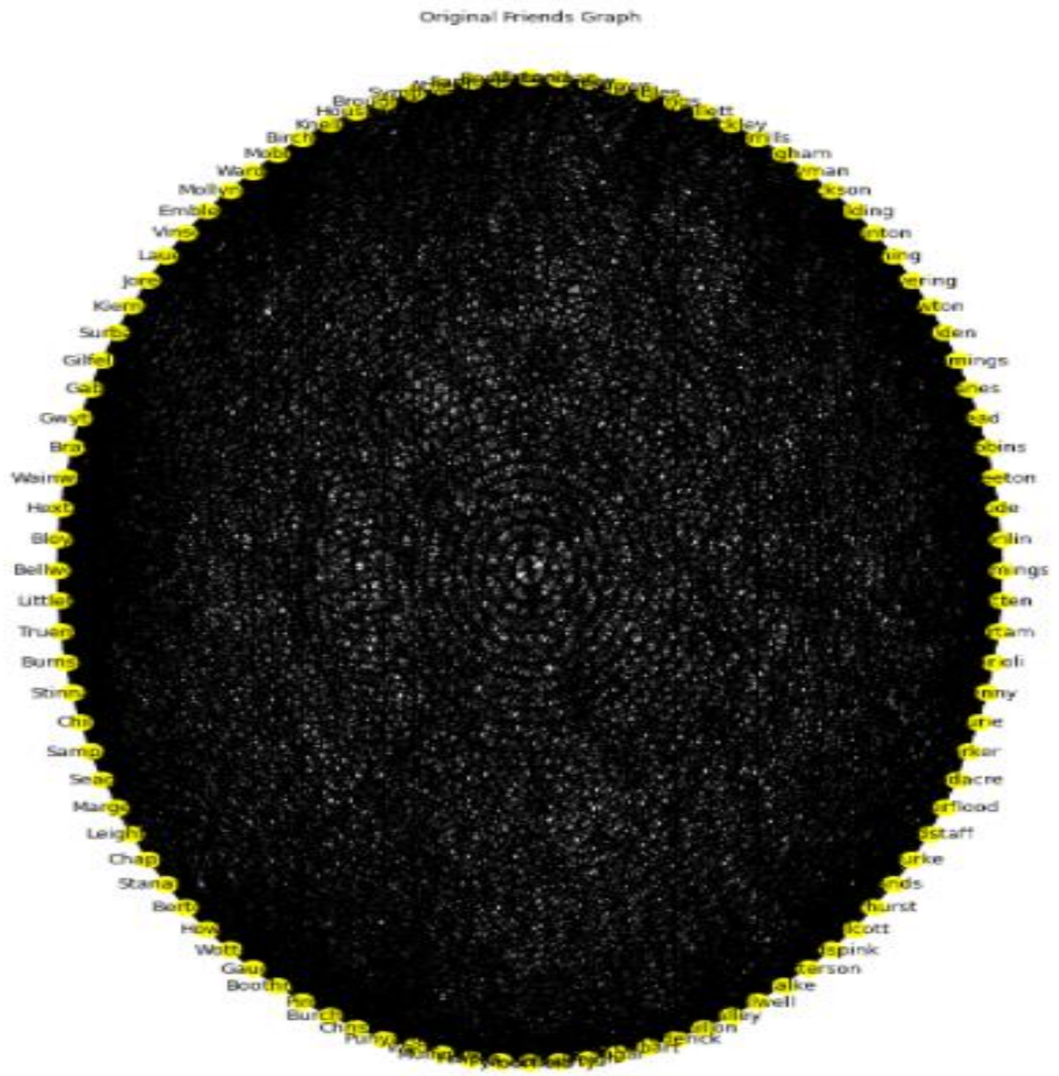


Greedy Algorithm. (Anonymized graph)

- With the greedy algorithm with a k-degree of 4. The following results were realized based on the friends-of-friends dataset.
- The graph on the left shows the anonymized data.

Greedy algorithm. Graphs.

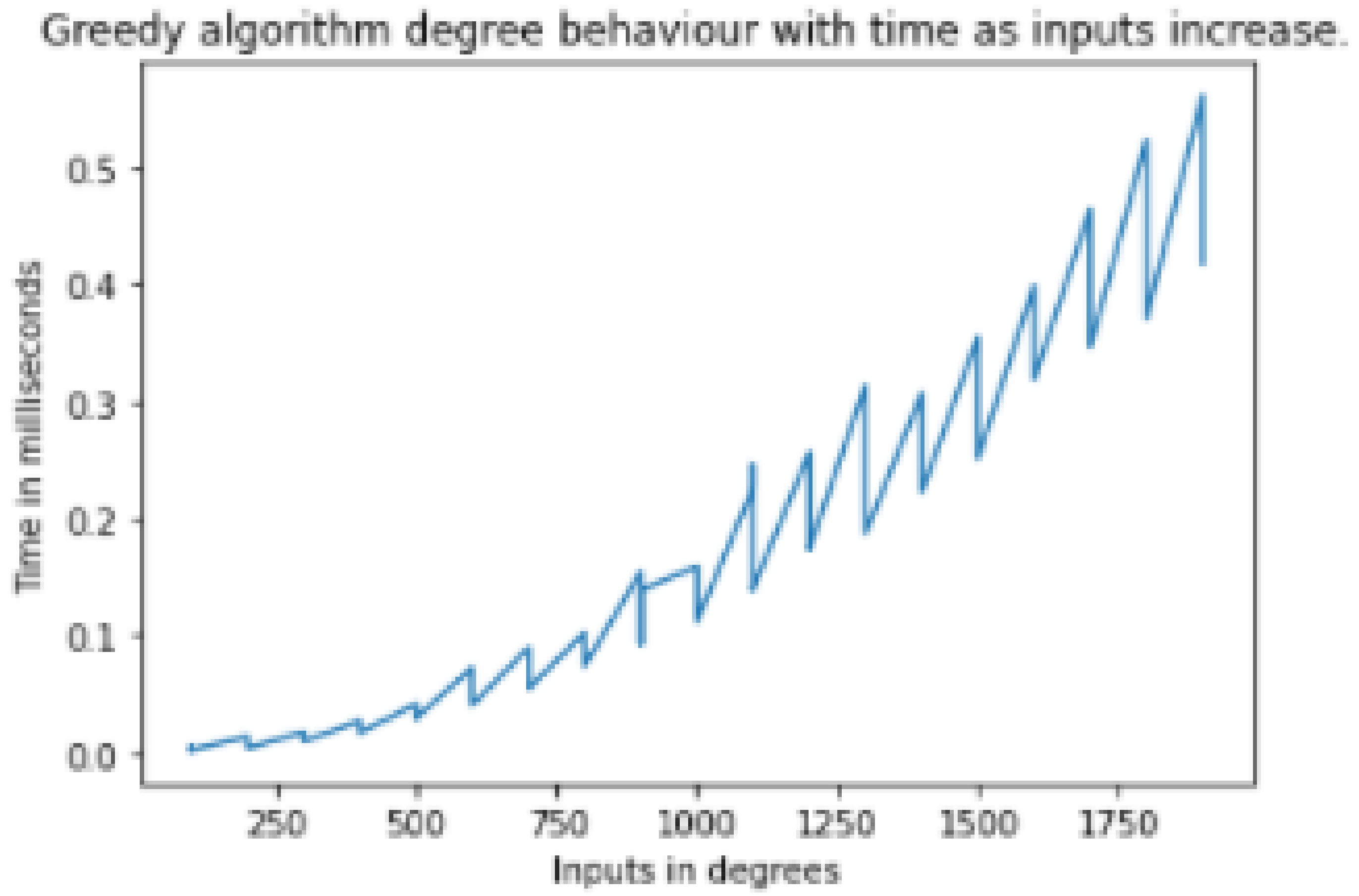
```
Out[290]: Text(0.5, 1.0, 'Anonymised Friends Graph Greedy')
```



Anonymised Friends Graph Greedy

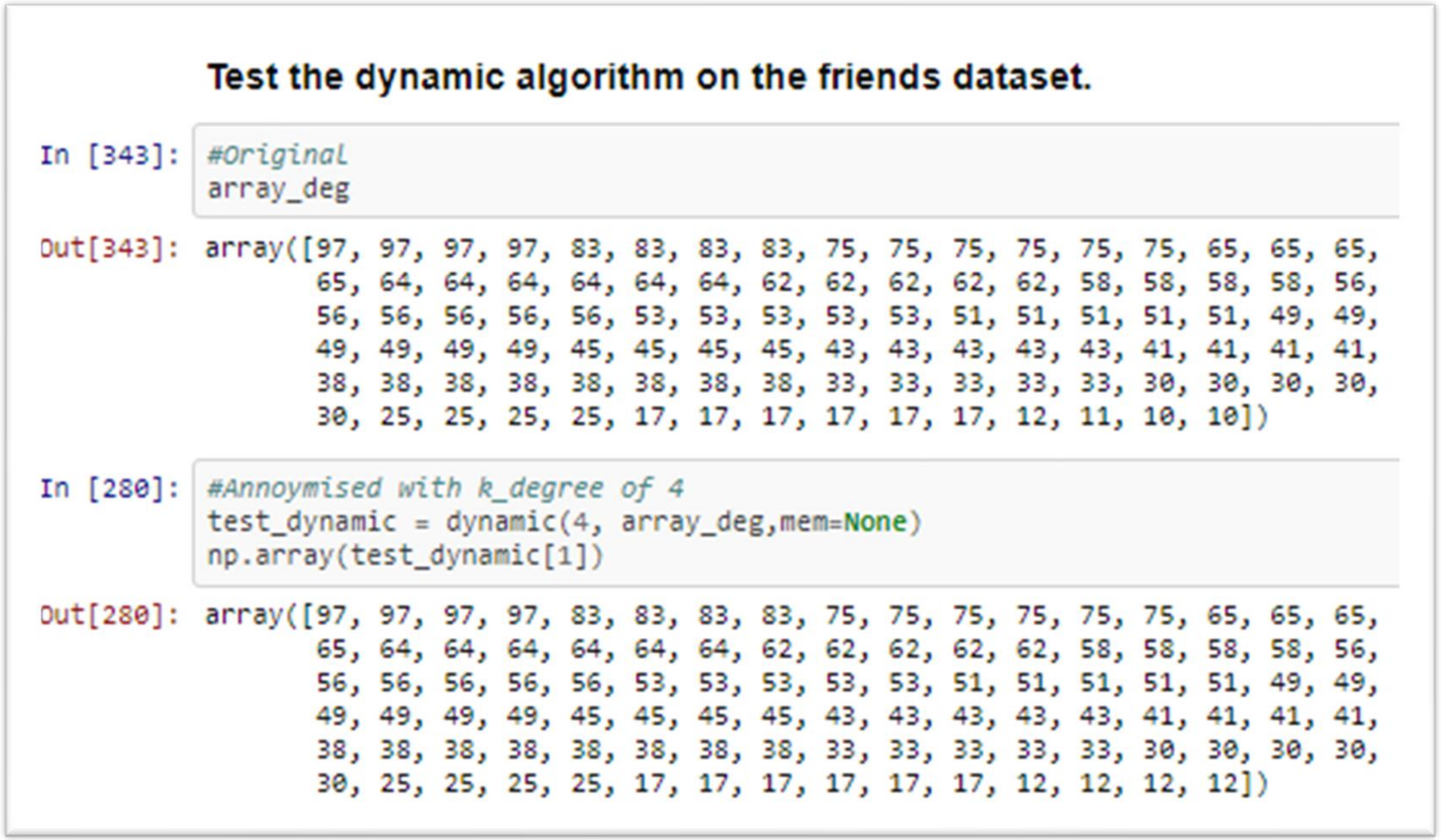
Time complexity of the algorithm. Greedy.

- While inputs increases the algorithm time of operation in terms of output also increases.
- In this case degrees varied from 100 to 2000.



Dynamic Algorithm.

- In this case also we used the same k-degree value of 4.
- Below is the output for the respective operation.

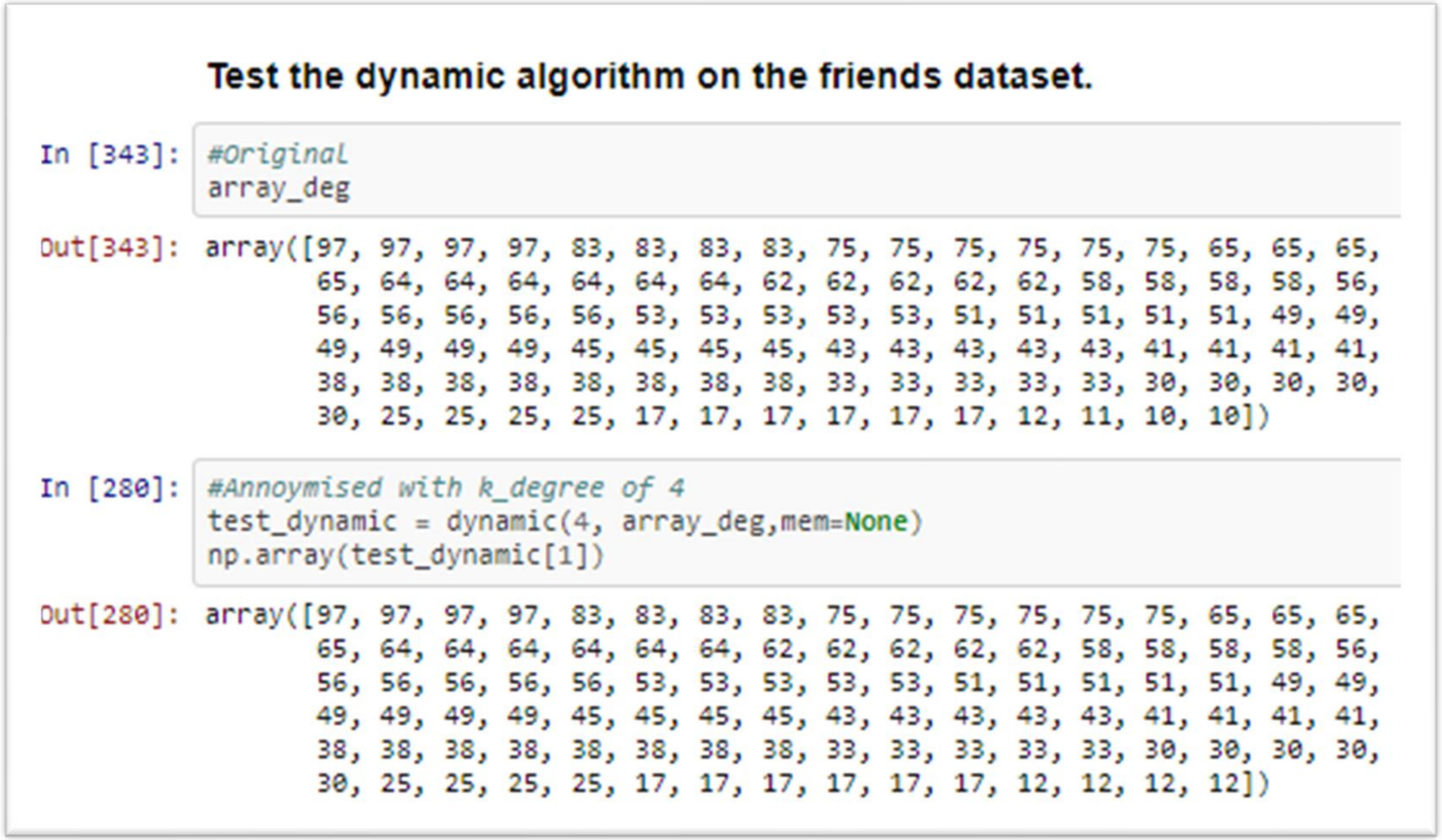


Test the dynamic algorithm on the friends dataset.

```
In [343]: #Original
          array_deg

Out[343]: array([97, 97, 97, 97, 83, 83, 83, 83, 75, 75, 75, 75, 75, 75, 65, 65, 65,
                 65, 64, 64, 64, 64, 64, 64, 62, 62, 62, 62, 62, 58, 58, 58, 58, 56,
                 56, 56, 56, 56, 56, 53, 53, 53, 53, 53, 51, 51, 51, 51, 51, 49, 49,
                 49, 49, 49, 49, 45, 45, 45, 45, 43, 43, 43, 43, 43, 41, 41, 41, 41,
                 38, 38, 38, 38, 38, 38, 38, 38, 33, 33, 33, 33, 33, 30, 30, 30, 30,
                 30, 25, 25, 25, 25, 17, 17, 17, 17, 17, 17, 12, 11, 10, 10])

In [280]: #Annoymised with k_degree of 4
          test_dynamic = dynamic(4, array_deg,mem=None)
          np.array(test_dynamic[1])

Out[280]: array([97, 97, 97, 97, 83, 83, 83, 83, 75, 75, 75, 75, 75, 75, 65, 65, 65,
                 65, 64, 64, 64, 64, 64, 64, 62, 62, 62, 62, 62, 58, 58, 58, 58, 56,
                 56, 56, 56, 56, 56, 53, 53, 53, 53, 53, 51, 51, 51, 51, 51, 49, 49,
                 49, 49, 49, 49, 45, 45, 45, 45, 43, 43, 43, 43, 43, 41, 41, 41, 41,
                 38, 38, 38, 38, 38, 38, 38, 38, 33, 33, 33, 33, 33, 30, 30, 30, 30,
                 30, 25, 25, 25, 25, 17, 17, 17, 17, 17, 17, 12, 12, 12, 12])
```

Dynamic Algorithm.

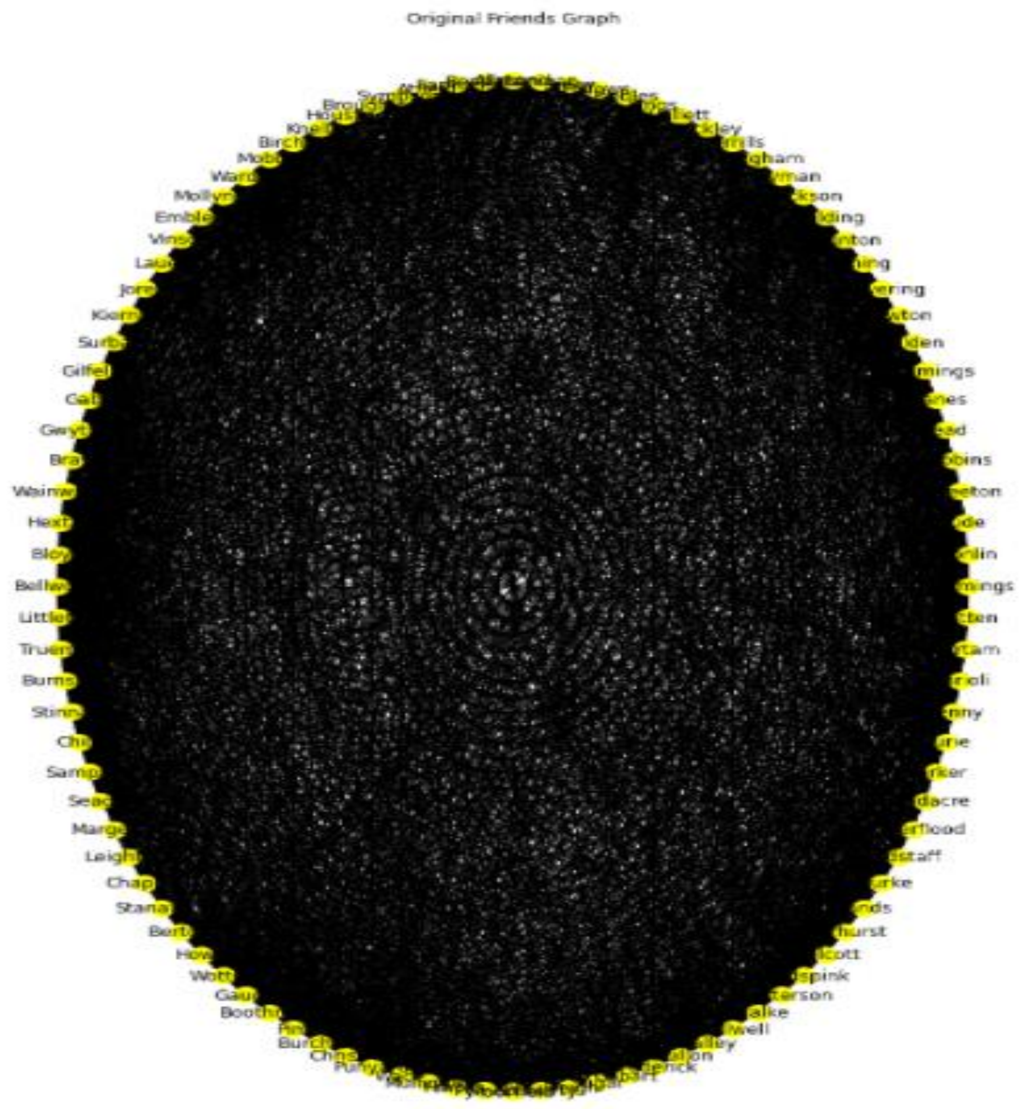
- In this case also we used the same k-degree value of 4.
- Below is the output for the respective operation.

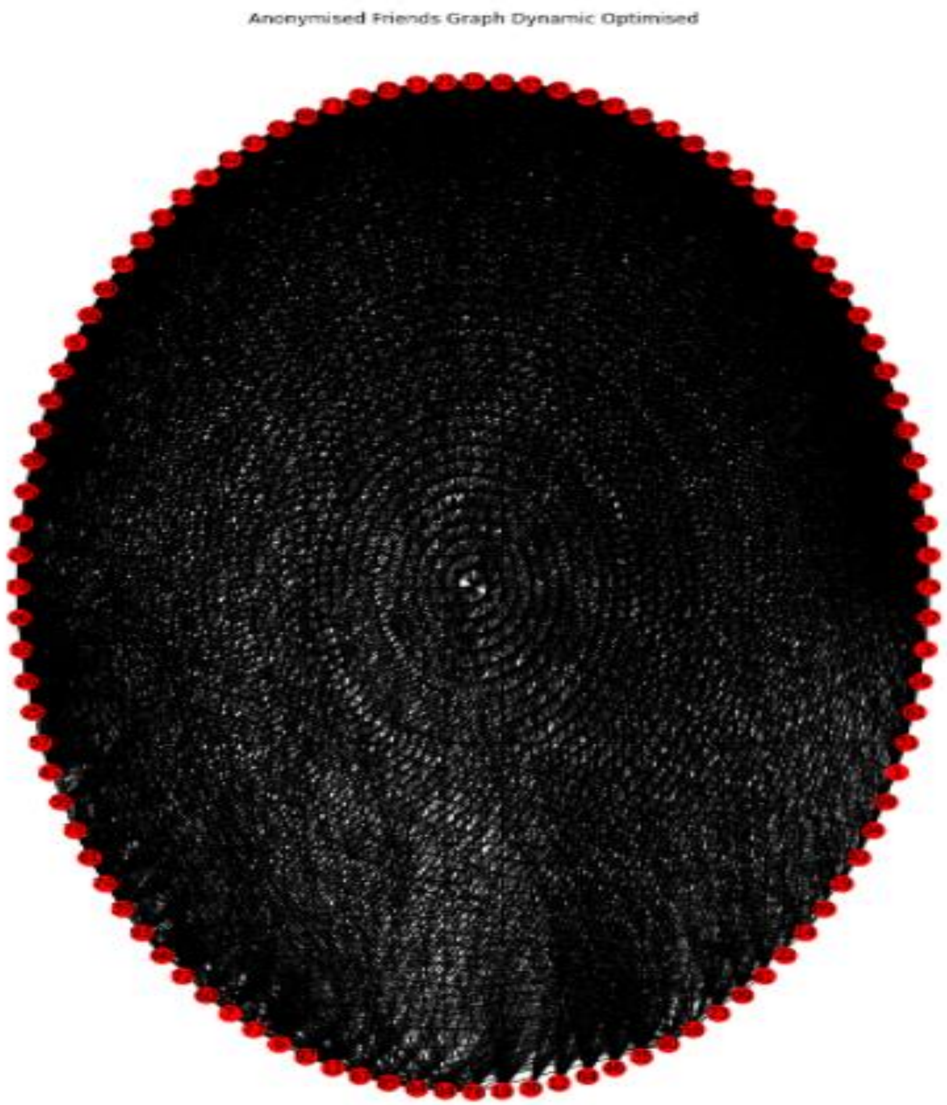


Test the dynamic algorithm on the friends dataset.

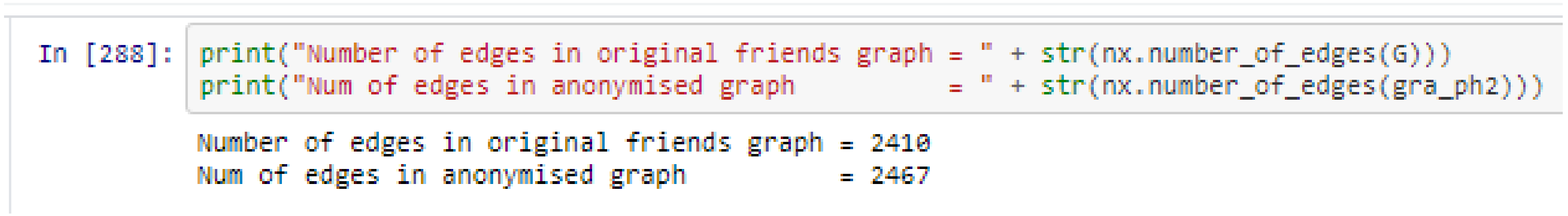
```
In [343]: #Original
          array_deg

Out[343]: array([97, 97, 97, 97, 83, 83, 83, 83, 75, 75, 75, 75, 75, 75, 65, 65, 65,
                 65, 64, 64, 64, 64, 64, 64, 62, 62, 62, 62, 62, 58, 58, 58, 58, 56,
                 56, 56, 56, 56, 56, 53, 53, 53, 53, 53, 51, 51, 51, 51, 51, 49, 49,
                 49, 49, 49, 49, 45, 45, 45, 45, 43, 43, 43, 43, 43, 41, 41, 41, 41,
                 38, 38, 38, 38, 38, 38, 38, 38, 33, 33, 33, 33, 33, 30, 30, 30, 30,
                 30, 25, 25, 25, 25, 17, 17, 17, 17, 17, 17, 12, 11, 10, 10])

In [280]: #Annoymised with k_degree of 4
          test_dynamic = dynamic(4, array_deg,mem=None)
          np.array(test_dynamic[1])

Out[280]: array([97, 97, 97, 97, 83, 83, 83, 83, 75, 75, 75, 75, 75, 75, 65, 65, 65,
                 65, 64, 64, 64, 64, 64, 64, 62, 62, 62, 62, 62, 58, 58, 58, 58, 56,
                 56, 56, 56, 56, 56, 53, 53, 53, 53, 53, 51, 51, 51, 51, 51, 49, 49,
                 49, 49, 49, 49, 45, 45, 45, 45, 43, 43, 43, 43, 43, 41, 41, 41, 41,
                 38, 38, 38, 38, 38, 38, 38, 38, 33, 33, 33, 33, 33, 30, 30, 30, 30,
                 30, 25, 25, 25, 25, 17, 17, 17, 17, 17, 17, 12, 12, 12, 12])
```

Dynamic algorithm.



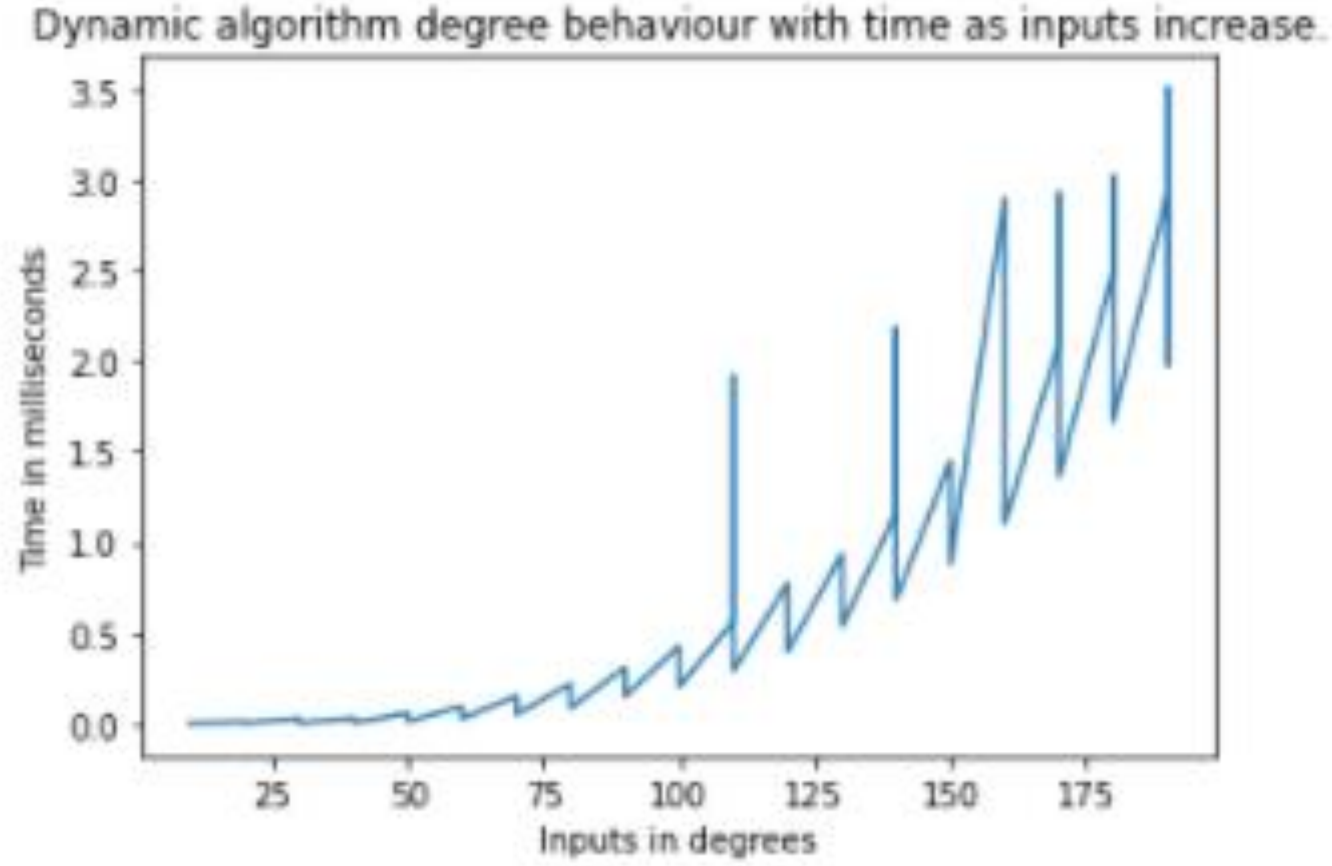


Dynamic Algorithm, change in graph structure.



```
In [288]: print("Number of edges in original friends graph = " + str(nx.number_of_edges(G)))
          print("Num of edges in anonymised graph          = " + str(nx.number_of_edges(gra_ph2)))

          Number of edges in original friends graph = 2410
          Num of edges in anonymised graph          = 2467
```

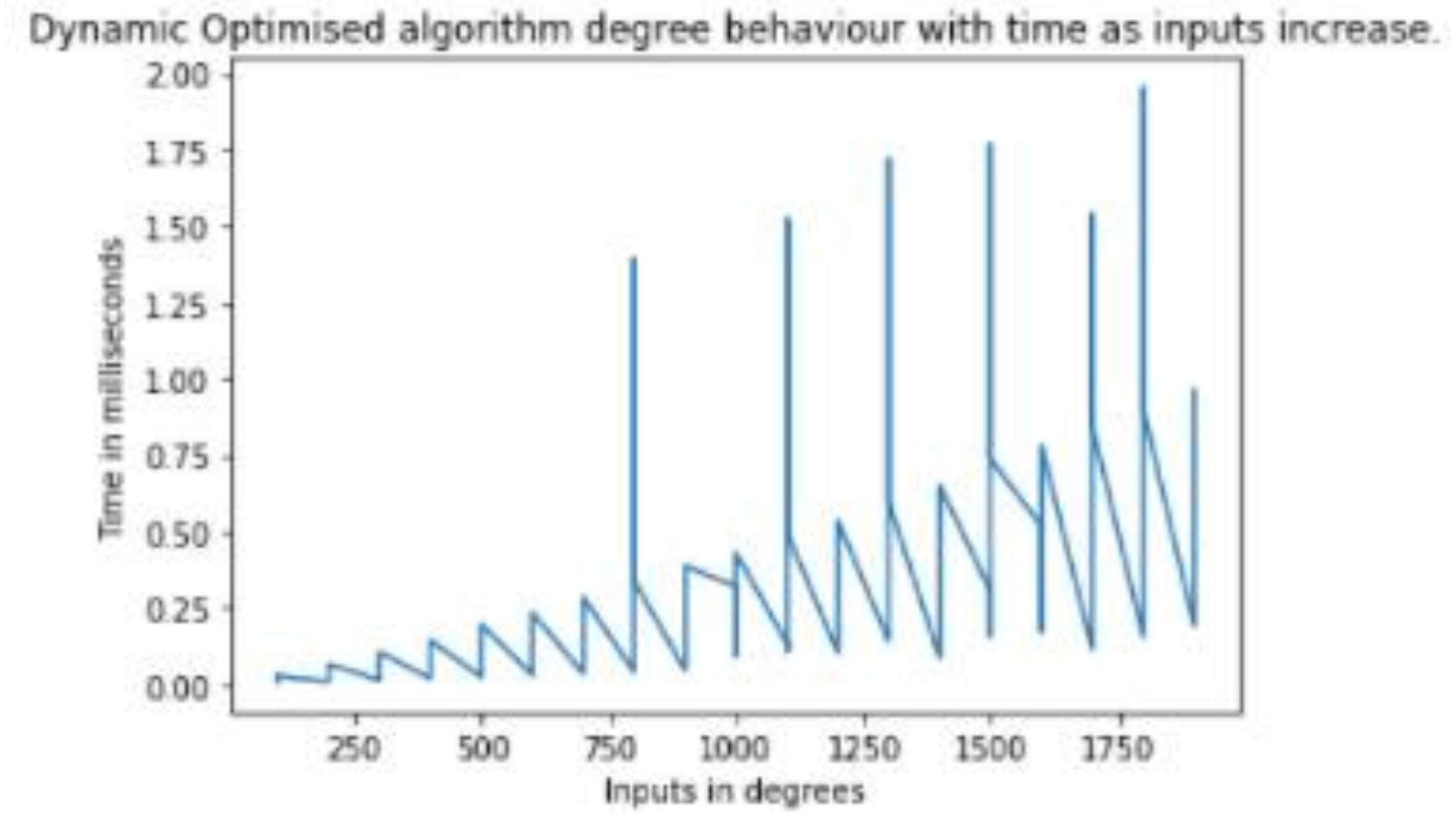
Dynamic algorithm(Time complexity)



- While inputs increase time also increase the inputs ranged from 10 to 200 degrees. It was very slow.

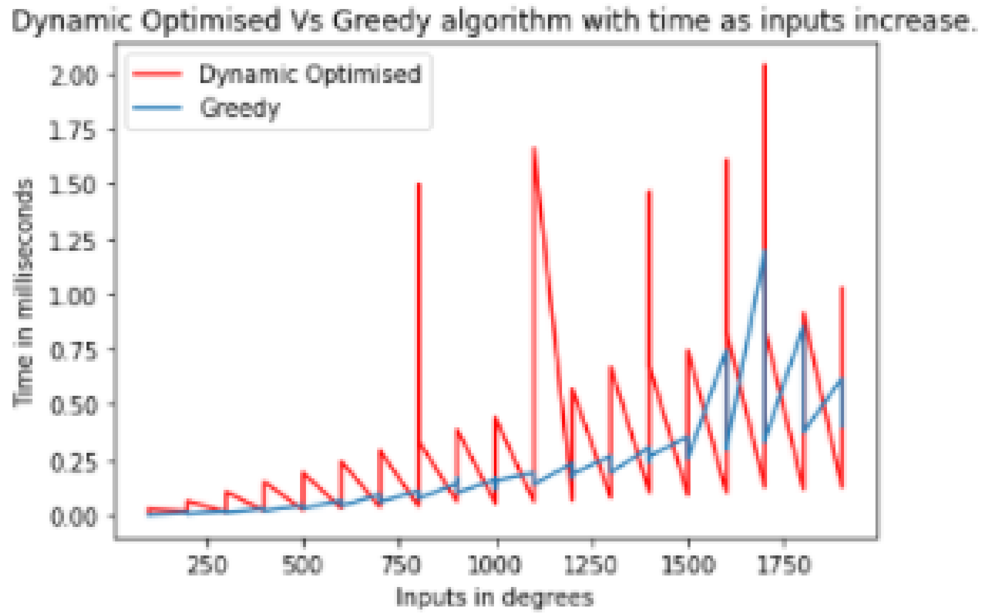
Dynamic Algorithm Optimized.

- As the inputs increased
- The time of operation also increased.
- But faster than non optimized version.



Compare operation of both algorithms.

```
Out[249]: <matplotlib.legend.Legend at 0x7fa71000aa30>
```



```
In [247]: print("Time in seconds for the Dynamic Optimised algo = " + str(sum(time_co)))
          print("Time in seconds for the Greedy algo            = " + str(sum(time_co1)))
```

```
Time in seconds for the Dynamic Optimised algo = 43.95826172828674
Time in seconds for the Greedy algo            = 27.5326189994812
```

- Overall the greedy algorithm did better than the dynamic while using barabasi albert dataset.