Mini Project

CS478

Made by:

Abdul Rafey Zafar (2019008)

Muhammad Afzal (2019279)

Instructor:

Sir Ahsan Shah

List of Contents

1. Introduction……..……3
2. Implementation……4
   2.1 Explanation……..…..6
3. Experimental Setup…….……..7
4. Results…….……8
   4.1 Plot……8
   4.2 Time……14
5. Discussion……15

1. Introduction:

Prim's and Kruskal's Algorithm are minimum spanning tree algorithms. They both use undirected graphs to create a minimum spanning tree that includes every vertex. Prim's algorithm uses starts making its tree from a specific vertex known as the root node and it grows the tree outwards. Whereas Kruskal's algorithm starts with all the edges in the graph and join the vertices in order of increasing weight of the edges. Prim's algorithm adds an edge to the tree assuming that it associates the tree to a vertex that isn't yet in that tree, while Kruskal's algorithm adds an edge to the tree on the off chance that it doesn't make a cycle.

The time complexity for Prim's algorithm for dense and sparse graph to create a minimum spanning tree is O((V+E)logV). The time complexity for Kruskal's algorithm for dense graph is $O(V^2)$ and sparse graph is O(VlogV) in best case scenario, whereas in worst case the time complexity is O(ElogE) to create a minimum spanning tree. Where (E) represents the edges of graph and (V) represents vertices of graph.

Prim's algorithm is generally considered faster for dense graphs having great number of edges. Kruskal's algorithm is generally considered faster for sparse graph having less number of edges.

2. Implementation:

```
import matplotlib.pyplot as plt
import networkx as nx
from itertools import combinations
from random import randint,random
import time

def DrawGraph(G):
    elarge = [(u, v) for (u, v, d) in G.edges(data=True) if d["weight"] > 0.5]
    esmall = [(u, v) for (u, v, d) in G.edges(data=True) if d["weight"] <= 0.5]
    pos = nx.spring_layout(G, seed=7)  # positions for all nodes - seed for
reproducibility
    # nodes
    nx.draw_networkx_nodes(G, pos, node_size=700)
    # edges
    nx.draw_networkx_edges(G, pos, edgelist=elarge, width=6)
    nx.draw_networkx_edges(
        G, pos, edgelist=esmall, width=6, alpha=0.5, edge_color="b",
style="dashed"
    )
    # node labels
    nx.draw_networkx_labels(G, pos, font_size=20, font_family="sans-serif")
    # edge weight labels
    edge_labels = nx.get_edge_attributes(G, "weight")
    nx.draw_networkx_edge_labels(G, pos, edge_labels)
    ax = plt.gca()
    ax.margins(0.08)
    plt.axis("off")
    plt.tight_layout()
    plt.show()


kdata=dict()
pdata=dict()
kdata1=dict()
pdata1=dict()
for i in range(5):
    G = nx.Graph()
    n = 50*(i+1)
    p = 0.1
    V = set([v for v in range(n)])

 for combination in combinations(V, 2):
        a = random()
```

```
        if a < p:
            G.add_edge(combination[0],combination[1],weight=randint(1,10))
    DrawGraph(G)
    starttime=time.time()
    nx.minimum_spanning_tree(G,algorithm='kruskal')
    endtime=time.time()
    kdata[n]=endtime-starttime
    kdata1[G.number_of_edges()]=endtime-starttime
    starttime=time.time()
    nx.minimum_spanning_tree(G,algorithm='prim')
    endtime=time.time()
    pdata[n]=endtime-starttime
    pdata1[G.number_of_edges()]=endtime-starttime

print("Fully Connected")
print('Time Taken Kruskal')
print(kdata.values())
print('Time Taken Prim')
print(pdata.values())

print("Dense")
print('Time Taken Kruskal')
print(kdata.values())
print('Time Taken Prim')
print(pdata.values())

print("Sparse")
print('Time Taken Kruskal')
print(kdata.values())
print('Time Taken Prim')
print(pdata.values())

plt.plot(*zip(*sorted(kdata.items())))
plt.xlabel("Total Nodes")
plt.ylabel("Time Taken")
plt.title('Kruskal sparse')
plt.show()
plt.plot(*zip(*sorted(kdata1.items())))
plt.xlabel("Total Edges")
plt.ylabel("Time Taken")
plt.title('Kruskal sparse')
plt.show()

plt.plot(*zip(*sorted(pdata.items())))
plt.xlabel("Total Nodes")
```

```
plt.ylabel("Time Taken")
plt.title('Prim sparse')
plt.show()
plt.plot(*zip(*sorted(pdata1.items())))
plt.xlabel("Total Edges")
plt.ylabel("Time Taken")
plt.title('Prim sparse')
plt.show()
```

2.1 Explanation:

We implemented Prim's and Kruskal's algorithm in python with the use of the networkx library. The first step was to generate a random graph given the number of vertices in said graph and the probability that there is an edge between two vertices. The number of vertices is denoted by (n) in the code and the probability of an edge is denoted by (p). By increasing and decreasing the value of (p) between 0 and 1 the number of edges in the graph increases likewise a probability of 0 will give a graph with no edges and a probability of 1 will generate a fully connected graph.

After generating a random graph we used the minimum_spanning_tree() function with parameters 'kruskal' and 'prim ' to generate minimum spanning trees using both the algorithms. These algorithms are implemented in networkx such that minimum spanning edges are generated first with the use of both Prim's and Kruskal's algorithms and then these edges are used to generate new graphs that are the minimum spanning trees.

We generated graphs of sizes 50, 100, 150, 200 and 250 and calculated the time taken to generate the minimum spanning trees using both prim's and Kruskal's algorithms. After which we plotted the graphs of time taken against the number of nodes and time taken against the number of edges in the graph.

3. Experimental Setup:

Approach: Greedy.

Language: Python (**Library used**: networkx).
https://networkx.org/

Platform: Windows 11.

System specification: PC used: Afzal's.

Device name: DESKTOP-IRPS92D.

Processor: 12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz.

Installed RAM: 16.0 GB (15.7 GB usable).

Device ID: F5AFA096-1292-417D-9B57-B0DA1F3B01D4.

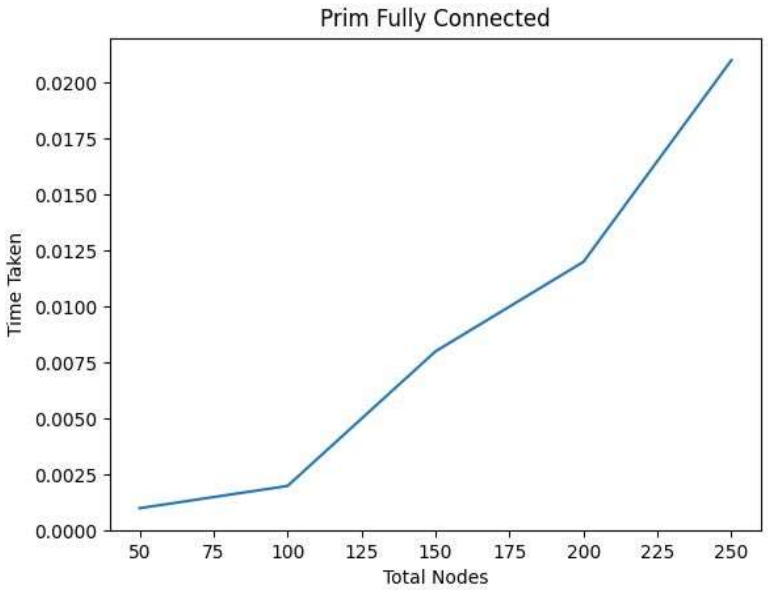
Product ID: 00342-20786-57972-AAOEM.

System type: 64-bit operating system, x64-based processor.

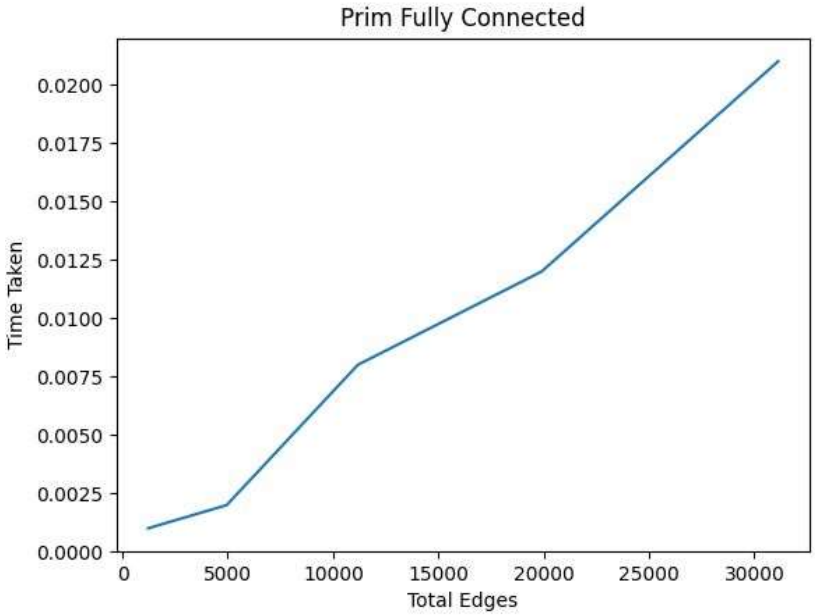
4. Results: 4.1 (Plot)

Prim's fully connected:

For Nodes:

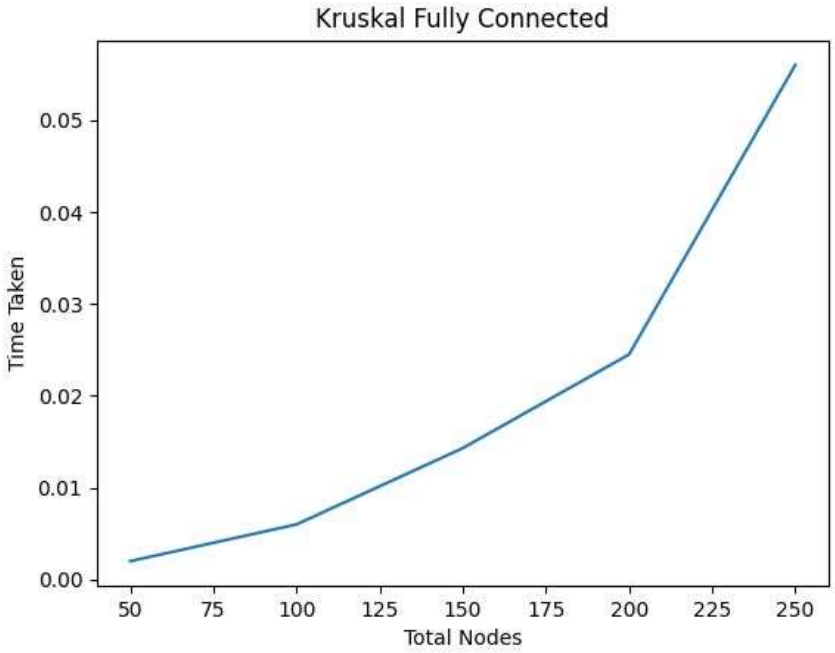


For Edges:

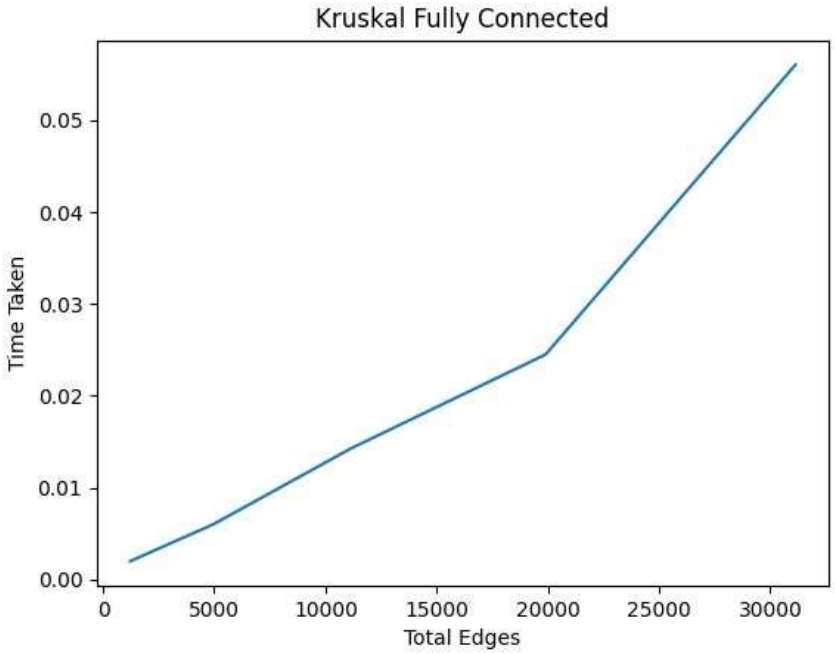


Kruskal's fully connected:

For Node:

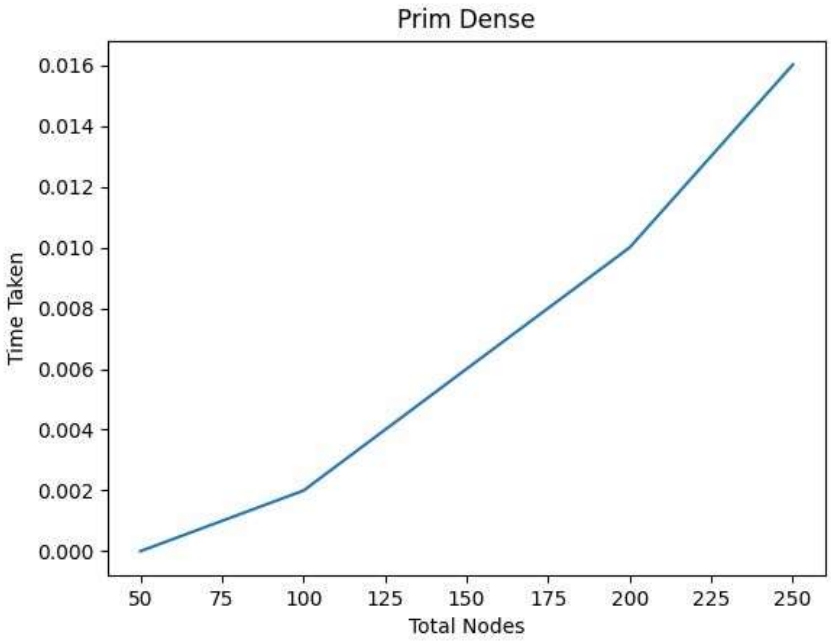


For Edges:

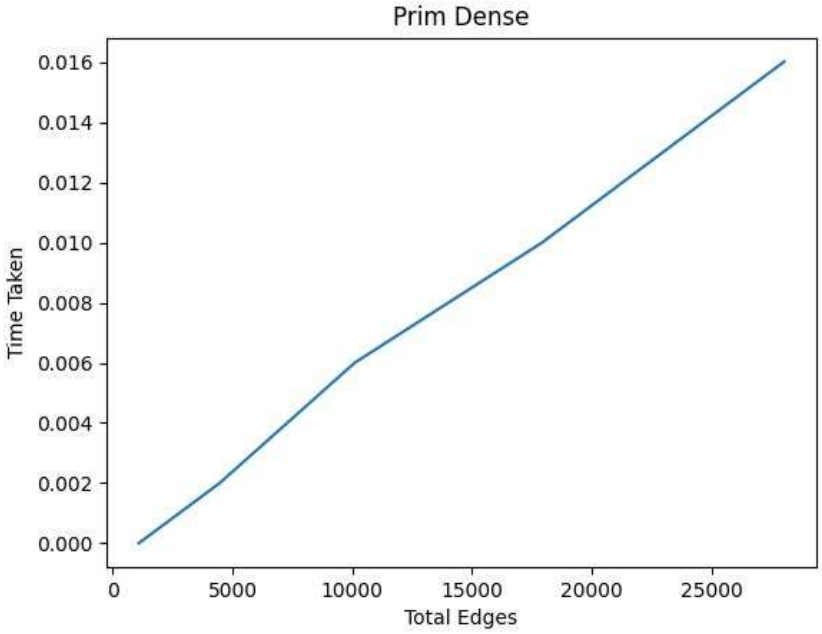


Prim's Dense:

For Nodes:

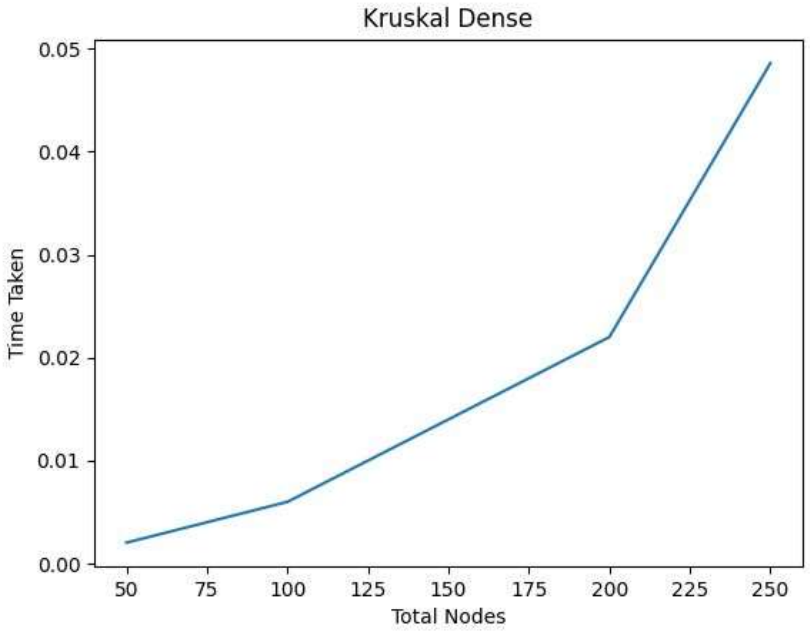


For Edges:

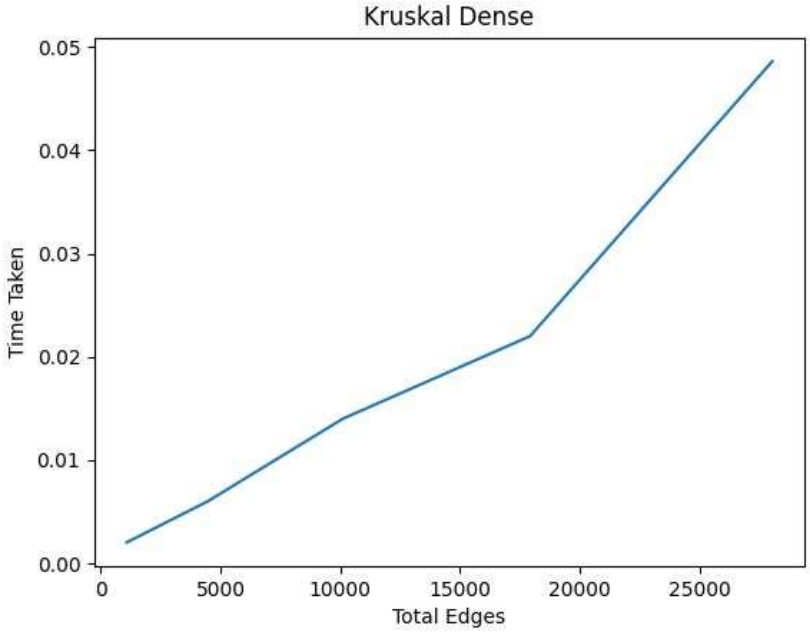


Kruskal's Dense:

For Nodes:

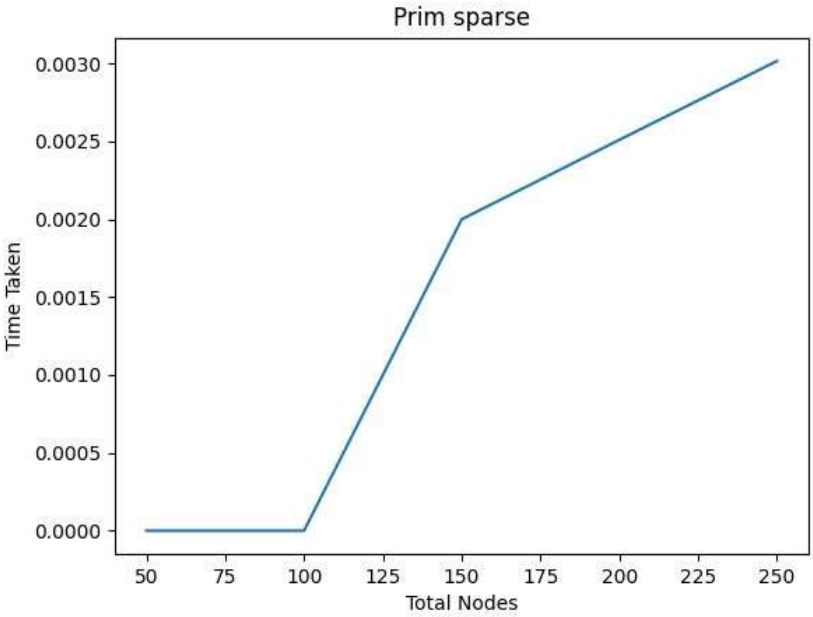


For Edges:

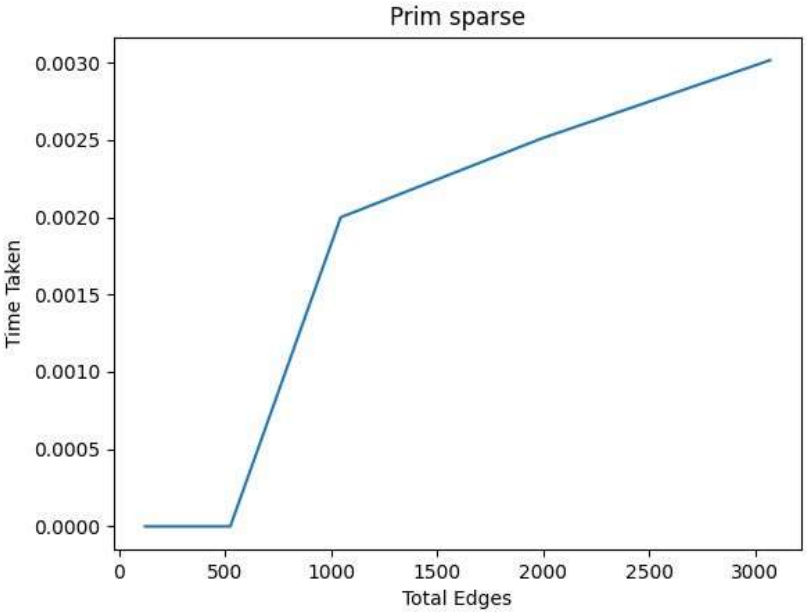


Prim's Sparse:

For Nodes:

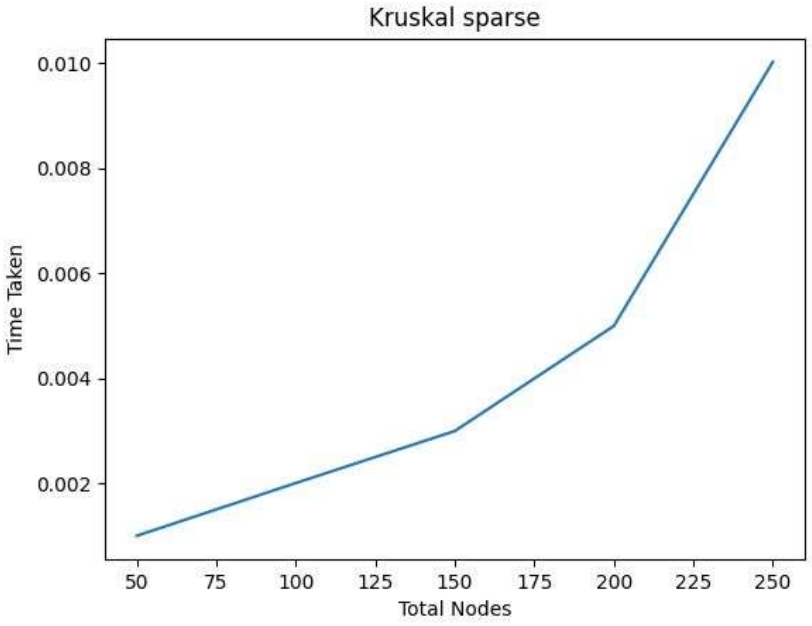


For Edges:

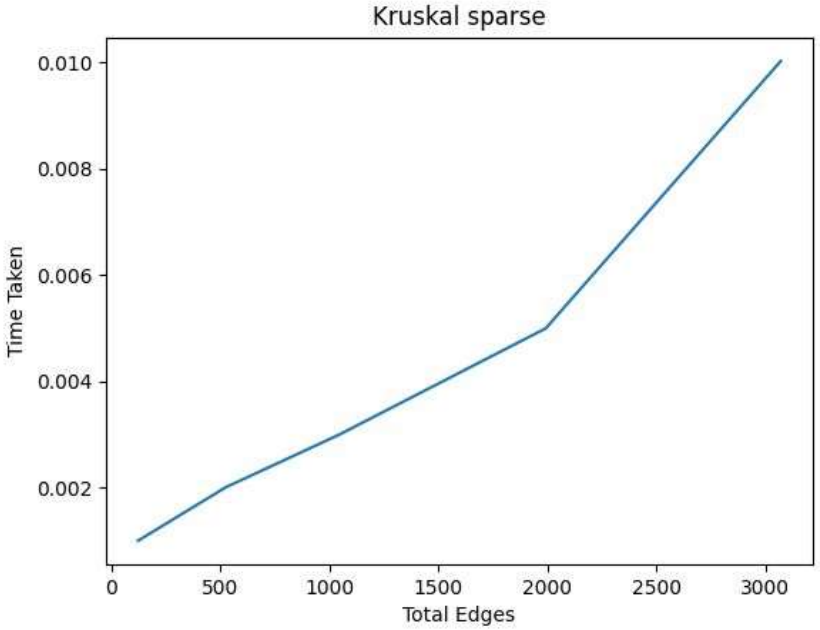


Kruskal's Sparse:

For Nodes:



For Edges:



4.2 Time:

Fully Connected

For n = 50, 100, 150, 200 and 250.

Probability = 1

```
Fully Connected
Time Taken Kruskal
dict_values([0.001999378204345703, 0.005000114440917969, 0.0120010375976562, 0.02155303955078125, 0.048004150390625])
Time Taken Prim
dict_values([0.0, 0.003000259399414062, 0.006000280380249023, 0.011001110076904297, 0.018508195877075195])
PS C:\Users\Afzal>
```

Dense

For n = 50, 100, 150, 200 and 250.

Probability = 0.5

```
Dense
Time Taken Kruskal
dict_values([0.000997781753540039, 0.003021001815795898, 0.00704455375671386, 0.011968851089477539, 0.018038988113403321])
Time Taken Prim
dict_values([0.0, 0.001999139785766601, 0.003106832504272461, 0.006000280380249023, 0.007972240447998047])
```

Sparse

For n = 50, 100, 150, 200 and 250.

Probability = 0.1

```
Sparse
Time Taken Kruskal
dict_values([0.001015663146972656, 0.001000881195068359, 0.001989364624023437, 0.002968549728393554, 0.004025697708129882])
Time Taken Prim
dict_values([0.0, 0.0, 0.001009464263916015, 0.001040458679199218, 0.002055644989013671])
```

The results consist of plot of time taken and values of time of our experiment for 5 different numbers of nodes and edges in fully connected, dense and sparse graph atmosphere to form a minimum spanning tree according to the nature of graph using Prim's and Kruskal's algorithm.

5. Discussion:

We tested both Kruskal's and Prim's algorithms for a range of number of vertices and edges of different graphs. We did this by generating minimum spanning trees from both Prim's and Kruskal's algorithms inside a loop which was increasing the number of vertices of the generated graphs by 50 for each iteration, giving us a range of nodes from 50 to 250. We did this for fully connected dense and sparse graphs. The number of edges were increased and decreased by changing the value of (p), that is the probability that two vertices have an edge between them, increasing the value of (p) increased the edge density of the generated graph and vice versa.

After recording the results and plotting the graphs for each of the above-mentioned graphs our results show that for our implementation Prim's algorithm is performing consistently better than Kruskal's algorithm on both dense and fully connected graphs while on a sparse graph Prim's algorithm is still performing better but not by much. The reason for this is that Prim's algorithm is implemented in networkx using heaps giving it a time complexity of O((V+E)logV) while Kruskal's algorithm has a time complexity O(ElogE).

In conclusion we recommend that Prim's algorithm should be used when creating a minimum spanning tree for large dense graphs and for smaller sparse graphs both Prim's algorithm and Kruskal's algorithm are viable options to use.