Student: John Rizzo
Course: CS590-A Algorithms
Instructor: Dr. William Hendrix
Due Date: April 18, 2025
Description: Homework 6 Algorithms

Problem 1

Line 2: Since we are initializing an $n \times n$ matrix it is $O(n^2)$
Line 7: $O(1)$ per operation
Line 6: Second Inner Loop $\sum deg(u)$ iterations
Line 5: $O(1)$ per operation
Line 4: 1st Inner Loop $\sum deg(v)$ iterations
Line 3: $O(n)$ times since it runs once per vertex
Lines 8-11: $O(1)$ time

Since each of the loops will run at most n times the deg component will be $O(n) = O(n \times n \times n) = O(n^3)$. The Sum of these times is $O(n^2) + O(n^3)$. As n grows $O(n^3)$ would dominate $O(n^2)$ so the answer becomes $O(n^3)$

Problem 2

```
from collections import deque

def is_tree(n, edges):
    if n == 0:
        return False
    if m != n - 1:
        return False

    # The adjacency list
    adj = [[] for _ in range(n)]
    for u, v in edges:
        adj[u].append(v)
        adj[v].append(u)

    visited = [False] * n
    parent = [-1] * n
    queue = deque()
    queue.append(0)
    visited[0] = True
```

```python
    while queue:
        u = queue.popleft()
        for v in adj[u]:
            if not visited[v]:
                visited[v] = True
                parent[v] = u
                queue.append(v)
            elif v != parent[u]:
                return False

    # Check if all vertices are visited (connected)
    return all(visited)

# Example
n = 3  # vertices
m = 2  # edges
edges = [(0, 1), (1, 2)]
print(is_tree(n, edges))
True
```

Problem 3

```python
from queue import PriorityQueue


class Point:
        def __init__(self, x, y):
            self.x = x
            self.y = y

        def __repr__(self):
            return f"Point({self.x}, {self.y})"

        def __eq__(self, other):
            return self.x == other.x and self.y == other.y

        def __hash__(self):
            return hash((self.x, self.y))


def shortestRookPath(points):
    start = points[0]
    end = points[-1]
```

```
# Build coordinate maps
x_map = {}
y_map = {}
for point in points:
    if point.x not in x_map:
        x_map[point.x] = []
    x_map[point.x].append(point)
    if point.y not in y_map:
        y_map[point.y] = []
    y_map[point.y].append(point)

# Dijkstra's setup
distances = {point: float('inf') for point in points}
distances[start] = 0
heap = PriorityQueue()
heap.put((0, start))

while not heap.empty():
    current_dist, current = heap.get()

    if current == end:
        return current_dist

    if current_dist > distances[current]:
        continue

    # Get all neighbors (same x or y)
    neighbors = []
    for point in x_map[current.x]:
        if point != current:
            neighbors.append(point)
    for point in y_map[current.y]:
        if point != current:
            neighbors.append(point)

    for neighbor in neighbors:
        neightbor_distance = abs(current.x - neighbor.x) + abs(current.y - neighbor.
        distance = current_dist + neightbor_distance
        if distance < distances[neighbor]:
            distances[neighbor] = distance
            heap.put((distance, neighbor))

return float('inf')  # if no path exists
```

```
if __name__ == '__main__':
    points = [Point(0, 0), Point(10, 0), Point(10, 1),
              Point(0, 2), Point(1, 2), Point(1, 1)]
    print(shortestRookPath(points))
```

```
CS590/HW6/problem3.py
4
```