Carson Sklare

2.

a. Use opposite of Kurskal's algorithm
   i. Order by weight descending
   ii. Process edges by decreasing weight order
   iii. If edge can be added without creating a cycle, do so
b. For finding the minimum feedback weighted edge set, use the original Kruskal algorithm
   i. Order by weight increasing
   ii. Process edges by increasing weight order
   iii. If edge can be added without creating a cycle, do so

3.

A.

Code:

```
def tour_length(P):
    total = 0
    for i in range(len(P)-1):
        x1 = P[i][0]
        y1 = P[i][1]
        x2 = P[i+1][0]
        y2 = P[i+1][1]
        total += ((x2-x1)**2+(y2-y1)**2)**0.5
    x1 = P[len(P)-1][0]
    y1 = P[len(P)-1][1]
    x2 = P[0][0]
    y2 = P[0][1]
    total += ((x2-x1)**2+(y2-y1)**2)**0.5
    return total
```

Result:

```
>>> P = [(265, 963), (343, 24), (408, 815), (57, 858), (10, 696), (403, 227), (3
90, 79), (990, 102)]
>>> tour_length(P)
4744.69248044646
```

B.

Code:

```
def TSP(P):
    bpathl = 999999999
    perm = permutations(P)
    for i in perm:
        pathl = tour_length(i)
        bpathl = min(bpathl,pathl)
        if bpathl == pathl:
            bpath = i
    return bpath
```

Result (Did not know why it did not show up as you example but I used tour_length to show that it did match the optimal 3012 in length)

```
>>> P = [(265, 963), (343, 24), (408, 815), (57, 858), (10, 696), (403, 227), (390, 79), (990, 102)]
>>> TSP(P)
((390, 79), (343, 24), (990, 102), (408, 815), (265, 963), (57, 858), (10, 696), (403, 227))
>>> P = ((390, 79), (343, 24), (990, 102), (408, 815), (265, 963), (57, 858), (10, 696), (403, 227))
>>> tour_length(P)
3012.3466598955056
```

c. I got to 8 cities before it took more than 5 seconds

4.

a

Code:

```
def approx_TSP(P):
    G = nx.Graph();
    for i in range(len(P)-1):
        x1 = P[i][0]
        y1 = P[i][1]
        x2 = P[i+1][0]
        y2 = P[i+1][1]
        G.add_edge((x1,y1),(x2,y2),weight = ((x2-x1)**2+(y2-y1)**2)**0.5)
        #G.add_edge("{},{}".format(x1,y1),"{},{}".format(x2,y2),weight = ((x2-x1)**2+(y2-y1)**2)**0.5)
    x1 = P[len(P)-1][0]
    y1 = P[len(P)-1][1]
    x2 = P[0][0]
    y2 = P[0][1]
    G.add_edge((x1,y1),(x2,y2),weight = ((x2-x1)**2+(y2-y1)**2)**0.5)
    #G.add_edge("{},{}".format(x1,y1),"{},{}".format(x2,y2),weight = ((x2-x1)**2+(y2-y1)**2)**0.5)
    mst = tree.minimum_spanning_edges(G,algorithm="kruskal",data = False)
    edgelist = list(mst)

    G2 = nx.Graph()
    for x in range(len(edgelist)-1):
        x1 = edgelist[x][0][0]
        y1 = edgelist[x][0][1]
        x2 = edgelist[x][1][0]
        y2 = edgelist[x][1][1]
        G2.add_edge((x1,y1),(x2,y2),weight = ((x2-x1)**2+(y2-y1)**2)**0.5)
    best = list(nx.dfs_preorder_nodes(G2))
    ttal = tour_length(best)
    return best,ttal
```

Result:

```
>>> P = [(265, 963), (343, 24), (408, 815), (57, 858), (10, 696), (403, 227), (390, 79), (990, 102)]
>>> approx_TSP(P)
([(403, 227), (390, 79), (990, 102), (10, 696), (57, 858), (408, 815), (343, 24)], 3422.627421788937)
```
```

B. Comparing it to the original TSP, it was close to making an optimal route but does not get to the best 3012

C. Since its running with Kruskal's algorithm it should be $O(E \log E) = O(E \log V)$. But because it has to make another graph I think it would be $2(O(E \log E) = O(E \log V))$a
```