Josh Hunt
CS 312
section 1
4/3/19

Project 5

1. Include your well-commented code.
2. Explain both the time and space complexity of your algorithm by showing and summing up the complexity of each subsection of your code. Keep in mind the following things:

   See code for complexity explanations.

3. Describe the data structures you use to represent the states.

I created a subProblem class that holds the functions and data members of my states. Each state had a cost (how much it is to travel from first city to last city in its path), a lower bound, representing the lower bound of that subProblem, and a path (a list of indexes representing the cities on the graph). The states also contained a matrix (used for lower bound calculation) and the very last city referred to before the new city is added.

The Sub-Problem class also contains a get lower bound function which takes a newly created matrix, and calculates the lower-bound of the matrix. The rest of the functions act as supporter methods to the state.

4. Describe the priority queue data structure you use and how it works.

The priority queue that I built utilized the heaq class for python. The three functions I use from heapq are heapq.heappop to get the new problem to expand, the heap.heappush which I used to add new subProblems to the heap, and heapq.heapify to initialize the heap. One functionality was necessary to make the heap work, and that was overriding the less than operator. In my SubProblem class, I overrode the class by telling it to return the problem with the smaller cost. I also used lower bound as a tiebreaker.

5. Describe your approach for the initial BSSF.

I used the default random tour function already implemented in the program. I retrieve its cost to initialize the BSSF cost. This will at least give a ball-park estimate of what the cost is supposed to be. This will save time for the algorithm, instead of using infinity to initialize the BSSF.

6. Include a table containing the following columns.

# Cities	**Seed**	**Time (sec)**	**Cost (BT)**	**Max size queue**	**# BSSF updates**	**# States Created**	**# States Pruned**
15	20	19.07	10,534	696	49	45485	38793
16	902	58.83	7,954	2784	55	188,566	162,931
10	588	0.1967	8,463	44	13	192,454	165,915
11	409	0.6644	7,745	187	18	195,281	168,143
12	320	0.5887	7,305	103	8	197,434	169,917
20	735	60	13,533	5,805	63	374,194	277,653
30	258	60	37,095	18,006	10	249,882	197,418
40	404	60	45,810	13,747	11	286,717	208,276
50	728	60	58,883	9,728	3	264,018	201,013

7. Discuss the results in the table and why you think the numbers are what they are, including how time complexity and pruned states vary with problem size.

   The values I receive will still be different for running them multiple times with the same number of cities and the same seed. This is because I utilized the randomized tour. This makes it difficult to see a correlation between the number of cities in a graph, and the number of states created and pruned.

   The cost is easier to identify a correlation. One a program states timing out, the cost of the graph keeps going up as the number of cities goes up. This can be true for max queue size and bssf updates. However, once the updates get very large, it's harder to complete expansions of sup problems, and the number starts to go down again.

   Again, data like this is hard to analyze with the randomness of each seed and randomness of the initial BSSF used. Using a greedy algorithm that consistently came up with the same solution would better help these statistics. Such a solution would also make the timing faster for these solutions.