Taylor Cowley
CS 312
Traveling Salesman Branch and Bound Lab

1. Code. See attached at end
2. Time and space complexity.
   We are doing the standard branch and bound. So we clone the data table so we can row and column reduce it at every node. With aggressive pruning, it appears that we never have more than 2^(n-1) nodes at a time, but this is definitely worst-case space complexity. Because we have to calculate nodes that we then prune, the very worst-case time complexity appears to be 2^n, but this is very difficult to achieve in practice.
   Starting with the base node, we bound it, which takes O(n). Then we branch it, which happens T(n-1) times, each of them requiring to bound themselves. This turns out being 2^n if we have to do all of them…
3. Every state is a node in my linked-list priority queue. A node has pointers to the previous and next node, a data 2-dimensional array (used for row-and-column reducing in bounding calculations), a path array, a path_size integer, and a double for the bounding of that node. Each node has everything required to compute itself and all of its children, allowing a priority queue instead of a tree.
4. The priority queue is implemented with a custom linked-list. Whenever a new node is created, a simple search from the beginning of the list to find where it belongs in the ordering is performed, and the node is inserted between the two nodes where it belongs. Because nodes are only removed, and never change priorities, the list stays sorted without outside influence.
5. To get the initial BSSF, we wait until one is calculated, and use that. If we do not find a solution within the allotted time, we call defaultSolveProblem instead of returning infinity.
6. Table of solutions.

# cities	**Seed**	**Running time**	**Cost of best tour (*=optimal)**	**Max # of stored states at a given time**	**# of BSSF updates**	**Total # of states created**	**Total # of states pruned**
15	20	60	3445	55639	305	64377	74494
16	902	0.3629	*3223	2402	13	2889	2575
17	902	0.5193	*3226	4914	3	5431	4924
18	902	4.4887	*3265	13913	11	26020	23603
16	901	60	4709	60502	11	72142	63653
17	901	60	5100	62102	1	69842	62429
19	903	60	3802	49894	9	71024	64701

7. Discuss results of the table. One thing I thought was super interesting is that some problems with fewer cities are much much more difficult than other ones with more cities. I suppose that is mostly just luck though, because if the algorithm dives down into an eventually horrible solution without being able to prune it, then it will waste much time. Another thing that was interesting is that the max # of states is suspiciously high. I suspect that it grows very high because it takes a very long time to actually find any solution, and then when one is found many states can then be pruned off. Obviously the time complexity is such that there is a very sharp cutoff between problems that are solvable within the time limit and ones that really are not. Although the difference between 16 and 17 cities with seed 901 is a lot smaller than I'd expect.