For this project we used the simulated annealing algorithm to solve the order of wizards. We were able to solve the order of wizards up to 200 inputs before the algorithm reduced in accuracy and thus no longer returned an optimal solution. Our algorithm ran on the time scale of about 20 minutes for the larger number of inputs however, we were using a python solver library which allowed us to set either the time run or the time steps run to avoid just waiting for the algorithm to run and instead timing out after a certain amount of time or time steps. The algorithm ran very quickly for the student inputs that were released.

We used simulated annealing to find the optimal state of the system. Our goal was to find the optimal solution or to find a close-to-optimal solution (within 4 violated constraints) so that we would either already have the ordering or we could manually find the ordering from the smaller set of violated constraints. The general algorithm was to start with a random state, and then the first step would to randomly move or alter the state. In this case it would be to flip any two wizards in their order. We would then calculate the energy of the new state. The energy was basically a value that directly correlated with the number of unsatisfied constraints. Now we then compare the energy and temperature of the new state with the previous state and then determine whether to "accept" the new state or to "reject" it which would then bring us back to the beginning of the loop. The temperature is a value that decreases exponentially as the algorithm progresses and it is responsible for helping ensure that the algorithm does not converge on a local minima of energy. For a new state to be "accepted" it must meet one of the two requirements that show the algorithm that we are moving towards an optimal solution. 1) The move causes a decrease in state energy or 2) the move increases the state energy but the temperature stays within a set of predetermined bounds. As mentioned earlier, if the temperature drops too much after taking a state that increases the energy, then we are most likely converging on a local minima. The parameters we used for the simulated annealing algorithm changed based on the size of the inputs but mostly for the student inputs, we had a time boundary of 5 minutes and for the staff inputs that we were able to solve we had a time input of 20 minutes. In terms of the max temperature and the min temperature, the python library that we used had an auto_schedule function, which chose ideal values for that based on the inputs and so optimized that part of the algorithm.

The python solver library that we used is called Simanneal (https://github.com/perrygeo/simanneal) and it is just a python module for simulated annealing optimization. In order to run it, you simply have to run the following command in terminal.

```
pip install simanneal
```

If you need, you may have to run with root privileges. The code should then run without error with Python3.5. The library does not work with Python2.7. Within the code you have to set the time that you want the algorithm to run for within the auto_schedule = tsp.auto(minutes=x) line where in you set x to 20 if it already has not been set.