CS 5112 Fall 2023	HW1: Stable Matching, Greedy, Divide and Conquer
Name: Matthew Shen	Due: September 26, 11:59pm ET
NetID: mds377	
Collaborators: Linjing Rao(lr534)	

Problem 1. Your friend is wrapping up medical school and applying for residency programs, but is concerned and confused about how the matching system works. As the local expert on algorithms, your colleague wants your help understanding how matchings work.

(a) (2 points) Your friend tried to implement the textbook Gale-Shapley (G-S) algorithm in Python. We provide this code in the homework materials in problem_1/p1_a.py. Using this implementation, your friend thinks they found a way to propose a ranking that unfairly advantages them in getting the school of their choice.

There is a logical bug in the implementation. Provide a minimal test case demonstrating the bug, i.e., an input with the smallest possible n that, when run with the buggy implementation, outputs a non-stable matching. Write your test case input in problem_1/p1a_test.txt.

(b) (8 points) Now we turn to characterizing the performance of this implementation. Fix the bug from part (a) and conduct an of the implementation (see homework instructions in the first page). Include the fixed code in problem_1/p1_b.py.

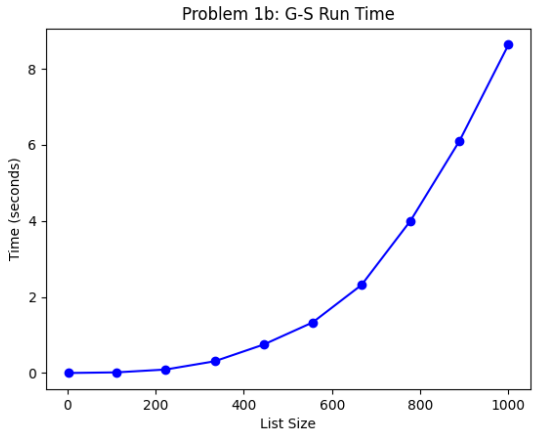
Provide a brief explanation of the performance you observe, and explain why the implementation does not achieve $\mathcal{O}(n^2)$ run time advertised for the G-S algorithm.

Answer: The environment used for the empirical performance analysis can be seen below:

- CPU: Intel(R) Core(TM) i7-8500Y CPU @ 1.50GHz (4 threads, 4.20GHz)
- OS & Version: Debian GNU/Linux 11 (bullseye)
- Memory: 15.53 GB

The graph created from the empirical performance analysis can be seen below:

The following code provided does not reach $\mathcal{O}(n^2)$. For the worst case scenario all hosptials have the same preference order for all doctors, and all doctors have the same preference order for all hospital. Additionally, the hospitals list is in the same order as the preferences that all the doctors have; therefore, a stable match is when the hosptial number and docter number matches (i.e. { 0:0, ... , n:n}).This means, in the first loop through the first hosptial gets matched and all other hospitals are not matched. In the second loop the program goes through all the hospitals again except for the first/matched hosptial. In the second loop on the second hosptial gets matched because all other hosptials want to get the second student, but the second student's top choice is the second hosptial. This means we get a loop has a time complexity of $\mathcal{O}(n)$ and is run $\mathcal{O}(n!)$ times. This means the algorithm is $\mathcal{O}(n!)$.



(c) (10 points) Correct and improve the run time of the G-S implementation and turn it in for auto-grading. It must be correct (always outputting a stable matching) and should run in the expected $\mathcal{O}(n^2)$ time. Provide a description of the optimizations you implemented. Additionally, provide an empirical performance analysis of your implementation in the same environment (same system and configuration) that you performed the earlier analysis. Include your new implementation in problem_1/p1_c.py.

Answer: In order to go from $\mathcal{O}(n!)$ to $\mathcal{O}(n^2)$ there are two changes that need to be made. The first is that in problem-1b the program loops through all the hospitals even if they are matched. This means that hosptials that are already matched are still checked even when they have already been matched to a doctor. This was fixed by setting a list of unmatched hosptials at the start. When a hospital gets matched they are removed from the unmatched hospital list, and it is this list that is continuously looped thorugh until all hospitals are matched. This means a hospital is only looped through if they dont have a match.

The second change is the removal of a doctor from the hospitals prefernce list if they are already matched to a hosptial with a higher ranking. This prevents the program from looping through doctors that we know are matched to their preferred hospital.

These two changes make it so the program only loops through all the doctors once for each hospital. Each of these operations is $\mathcal{O}(n)$ so the total time complexity is $\mathcal{O}(n^2)$.

For the empirical performance analysis the environment was:

- CPU: Intel(R) Core(TM) i7-8500Y CPU @ 1.50GHz (4 threads, 4.20GHz)
- OS & Version: Debian GNU/Linux 11 (bullseye)
- Memory: 15.53 GB

The graph from the emperical performance analysis can be seen below: