EE W360C ALGORITHMS
Programming assignment 1
Name: Yang Hu
UT EID: yh8473
Due: July 5th

Part 1: Write a report [30 points]

Write a short report that includes the following information:

a) *This algorithm is incredibly inefficient. Assume there are n advisors and n students. In the worst case, what would be the number of pairs, x, made? What would the preference lists of the advisors and students look like to end up with only x pairs?*

In the worst case, three pairs would be made. In the worst case in the first round, all advisors would decide to offer admission to one student who has an outstanding GPA. Then this student will accept the offer from the advisor on the top of his or her preference list. Thus, one pair of advisors and students forms. Then in the second round, advisors will go to students who have not formed pairs because they know that students can not reject after accepting an offer. Therefore, the worst case is that all advisors will again offer admission to a student and he or she will accept the offer from the advisor on the top of his or her preference list. Then in the third round, the worst case happens again and only one pair forms. Then after three round, three pairs are made.

To allow this worst case to happen, the top three student in the preference list of the advisors will be the same such that the advisors want to go the top of their preference list and end up choosing the same students in each round. The preference list of students can vary.

b) Provide a better algorithm in pseudocode that finds a stable assignment. (Hint: it should be very similar to the Gale-Shapley Algorithm)

Create list of all advisors and students and their preference list and let it represent advisors and students who are free
Create a mapping student_matching set ready for inserting pairs
Also build the preference list for each advisors (in another function)
While there exist advisor who do not have any students who accept their offer or who has not tried to offer admission to all students do:
    Choose an advisori and find his or her most preferred studenti who need have been offered admission; Since the advisor in the previous cycle will be paired with some student anyway and will be removed, so choosing the first advisor in the advisor-free list will be good.
        While advisori has not find it's matching
            If the studenti not yet have any offer

Advisori offers admission and pair up advisori and studenti put in to the student_matching

Else (this student n0 has and accept another offer from advisor m1)

If student s0 prefer current paired advisor a1

Advisor m0 remains unpaired

Else (this studenti has and accept another offer from advisorj, get advisor from matching set)

find the rank of advisori and advisorj in studenti preference

if(studenti prefers advisori)

pair studenti with advisori

advisorj becomes unpaired so put advisorj back to free advisor list and remove current advisori at index 0

Else: studenti prefer current paired advisorj

Advisor m0 remains unpaired

if studenti does not pair up with advisori, go to the next student according to the advisor preference list

Return marriage with mapping set M

In function that build the advisors' preference list:

Create GPA list and a ranking list. The ranking list corresponds the GPA list, i.e. with the same index, what's in the ranking list represent who has that GPA. If there is no ties, then the corresponding ranking list has only one student. If there are ties, multiple student would be in the same ArrayList in the ranking nested list. Then we will personalize the ranking for each advisor afterwards according to the locations.

c) Give a proof of your algorithm's correctness. Remember that you must prove both that your algorithm terminates and gives a correct (i.e. no-one is unmatched, matching is stable) result.

The while loop in this algorithms breaks under two scenarios, either if every advisor find a student or if the advisor has offered admission to every student.

Firstly, we need to prove that this algorithm terminates.

Proof: The advisors all will be able to offer students admission according to their preference list. And the while loop in this algorithms breaks if the advisor has offered admission to every student. There is finite many of advisors and students, so there is only finite many times that the advisor would offer admission to students because the advisors will not offer students admission again after they tried once. So the while loop will terminate.

Then we want to proof that all advisors and students will match and there is no one left alone after the while loop breaks.

Proof: The students will always have an advisor to pair once they are offered admission, because students can only change their partner after matching with advisors. The while loop only breaks either:

1. If every advisor find a student, then this is already perfect matching.
2. If the advisor has offered admission to every student, every student will have at least get one offer, and since students will accept the offer once they have it, so all students have a pair. Because we have n students and n advisors, all advisors will have a student to pair with. Therefore, after the while loop terminates, all advisors and students must have their pairs, which is a perfect matching.

Finally we want to proof that the matching will be stable.

Proof by contradiction: Assume there exists instability and there are two pairs (studenti, advisori) and (studentj, advisorj). The instability exists because studenti prefers advisorj to advisori and advisorj prefers studenti to studentj. Since advisors would ask students according to their preference list and the algorithm give the result that advisori and studenti are matched, the advisori would either asked studentj earlier but get rejected or not able to ask studentj yet.

1. If the advisori asked studentj earlier but get rejected, studentj would not prefer advisori over advisorj.
2. If the advisori has not yet asked studentj, then the ranking of studentj is lower than the ranking of studenti in the preference of advisori. Then the advisori would not prefer studentj over studenti.

Both situations contradict to our assumption. Thus, the matching generated by out algorithm will be stable.

Since we proved the algorithm will terminate and will generate perfect matching that is stable, the algorithm is correct.

d) Give the runtime complexity of your algorithm in Big-O notation and explain how you derived it from the pseudocode.

The runtime complexity in Big-O notation is O(N^2).

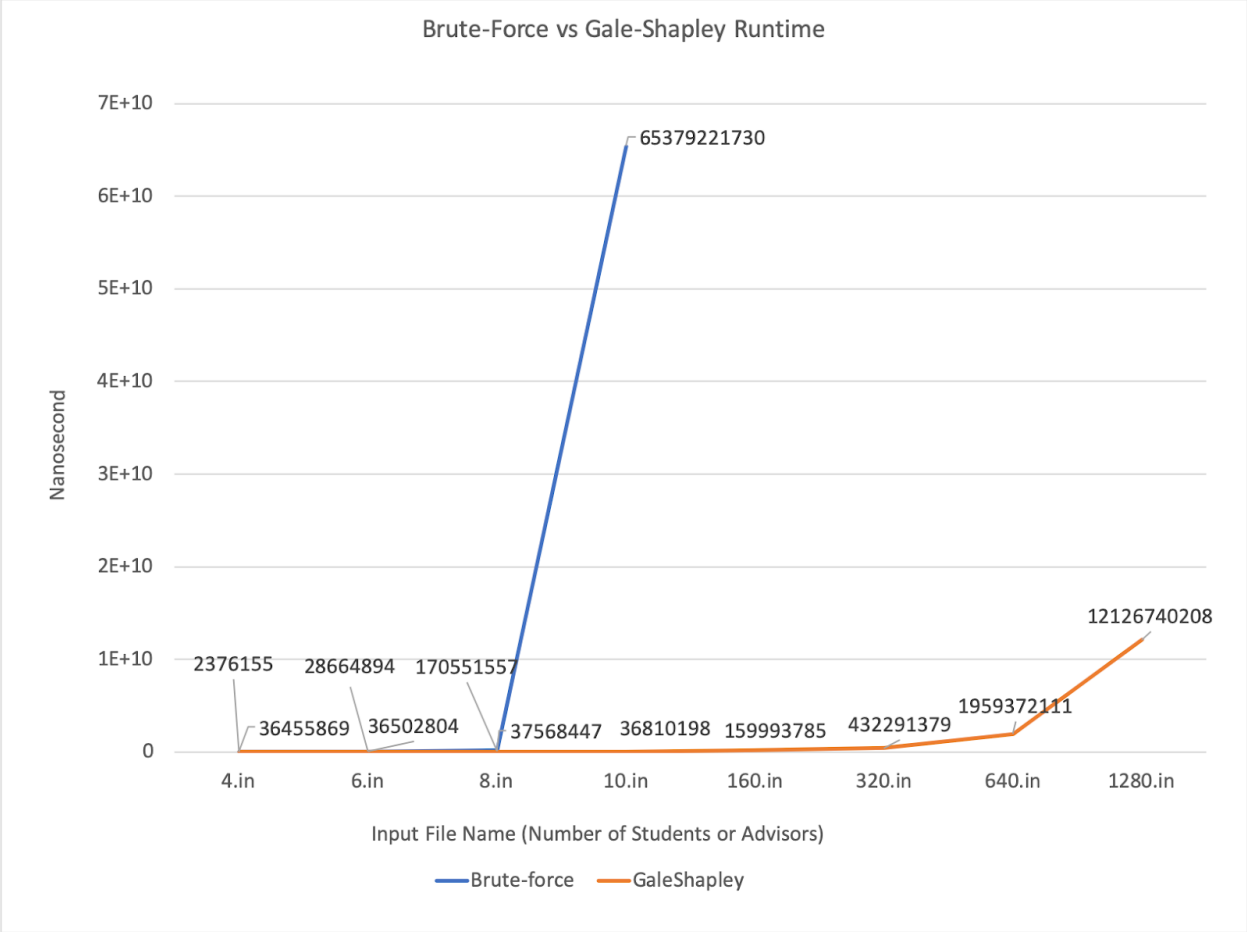
Since the while loop breaks under two scenarios, either if every advisor find a student or if the advisor has offered admission to every student. In the worst case, the advisor will ask every student, that is n students, and then find a student who would accept the offer. Checking if this student will accept needs constant time only because both checking if the student has an offer, and if the student prefer this advisor to another are only requiring constant amount of time. So each advisor at most need O(N) to pair with a student. Since there is n advisors, the runtime complexity is O(N) * N = O(N*N) = O(N^2).

Building the preference list for each advisor also has the same complexity O(N^2). In our algorithm, we first build a GPA list and a ranking list where we find all students who has same GPAs. This procedure take O(N) time because we went through each student's GPA once. Then we personalize the preference list for each advisor. Personalizing preference list takes O(N) because we go through GPA list and ranking list in O(N) time and the calculation and comparison of locations takes constant time O(1). For N advisors, the complexity of building the preference list is O(N*N) = O(N^2).

e) Consider a Brute Force Implementation of the algorithm where you find all combinations of possible matchings and verify whether they form a weakly stable marriage one by one. Give the runtime complexity of this brute force algorithm in Big O notation and explain why.

The runtime complexity is O(N!N^2). The Brute Force Implementation idea is to find all possible matching pairs and check if they are stable. The number of all possible matchings without duplications is n! pairs, if there are n elements. There are n! Possible pairs because the first element could possibly pair with any of the rest elements, that is (n-1) number of the elements. Then the second elements could possibly pair with (n-2) number of elements, and the third one with (n-3) number of elements, and so on. The total number of possible pairs will be (n-1)*(n-2)*(n-3)*...*3*2*1 = (n-1)!, and O((n-1)!) = O(n!). Then to verify if each pair forms a weakly stable marriage, we need to show there is no weakly instability. Then by definition of weak instability, when we have an advisor a and a student s, and their current pair are s0 and a0 respectively, either if a prefers s to s0, and s either prefers a to a0 or is indifferent between these two, or if s prefers a to a0, and a either prefers s to s0 or is indifferent between these two choices. Therefore, for each advisor, we need to compare their current student pair with all other students, and there are n advisor. Therefore, n^2 number of operations are needed to check there is no instability in the worst case. Checking n! Pairs with n^2 numbers of operations each time will lead to n^2 * n! Number of operations in the worst case. In big O-notation, the runtime complexity is O(N^2 * N!).

f) In the following two sections you will implement code for a brute force solution and an efficient solution. In your report, use the provided data files to plot the number of couples (x-axis) against the time in ms it takes for your code to run (y-axis). There are four small data files and four large data files included in the input provided. The large data files may be too large for the brute force algorithm to finish running on your machine. If that is the case, do not worry about plotting the brute force results for the large data files. Your plot should therefore contain 8 points from your efficient algorithm and 4-8 points from the brute force algorithm. Please make sure the points from different algorithms are distinct so that you can easily compare the runtimes from the brute force algorithm and your efficient algorithm. Scale the plot so that the comparisons are easy to make (we recommend a logarithmic scaling). Also take note of the trend in run time as the number of advisors increases.



The runtime for brute-force solution dramatically increases. Solving a 10 student 10 advisors problem takes about 400 times, not to mention the runtime of large input files. Whereas Gale-Shapley algorithm took about the same time to run all the small_inputs files, and increase slowly in polynomial runtime.

Part 2: Implement a Brute Force Solution [30 points]

A brute force solution to this problem involves generating all possible permutations of students and advisers, and checking whether each one is a stable matching, until a stable matching is found. For this part of the assignment, you are to implement a function that verifies whether or not a given matching is stable. We have provided most of the brute force solution already, including input, function skeletons, abstract classes, and a data structure. Your code will go inside a skeleton file called Program1.java. A file named Matching.java contains the data structure for a matching. See the instructions section for more information.
Inside Program1.java is a placeholder for a verify function called isStableMatching(). Implement this function to complete the Brute Force algorithm (the rest of the code for the brute force algorithm is already provided).
Of the files we have provided, please only modify Problem1.java, so that your solution remains compatible with ours. However, feel free to add any additional Java files (of your own authorship) as you see fit.

See *Program1.java.*

Part 3: Implement an Efficient Algorithm [40 points]

We can do better than the above using an algorithm similar to the Gale-Shapley algorithm. Implement the efficient algorithm you devised in your report. Again, you are provided several files to work with. Implement the function stableMarriageGaleShapley() inside of Program1.java.
Of the files we have provided, please only modify Problem1.java, so that your solution remains compatible with the grading program. However, feel free to add any additional Java files (of your own authorship) as you see fit.

See *Program1.java.*