Homework 2

Due by **12:30pm, Friday, September 23, 2016**.

Make sure you follow all the homework policies (http://www-student.cse.buffalo.edu/~atri/cse331/fall16/policies/hw-policy.html).

All submissions should be done via Autolab (http://www-student.cse.buffalo.edu/~atri/cse331/fall16/autolab.html).

Sample Problem

The Problem

This problem is just to get you thinking about asymptotic analysis and input sizes.

An integer $n \geq 2$ is a prime, if the only divisors it has is 1 and n. Consider the following algorithm to check if the given number n is prime or not:

> For every integer $2 \leq i \leq \sqrt{n}$, check if i divides n. If so declare n to be *not* a prime. If no such i exists, declare n to be a prime.

What is the function $f(n)$ such that the algorithm above has running time $\Theta(f(n))$? Is this a polynomial running time-- justify your answer. (A tangential question: Why is the algorithm correct?)

Click here for the Solution

Submission

You will **NOT** submit this question. This is for you to get into thinking more about asymptotic analysis.

Question 1 (Programming Assignment) [40 points]

</> Note

This assignment can be solved in either Java, Python or C++ (you should pick the language you are most comfortable with). Please make sure to look at the supporting documentation and files for the language of your choosing.

The Problem

In this problem we will consider the extension of the stable matching problem that more closely resembles the resident matching program that NRMP (http://www.nrmp.org/) administers.

We are given m hospitals and n medical students. Each hospital has a ranking of *all* the students in order of preference, and each student has a ranking of *all* the hospitals in order of preference. Unlike the stable matching problem, a hospital *can* have more than one open slots (but a student can be assigned at most one hospital). We will assume that there are more students than there are slots available in all the m hospitals put together. The goal is to output a **stable** assignment of students to hospitals. Note that since there are more students than hospitals, some students may be not assigned to any hospital. Also no hospital is assigned more students than the number of openings it has.

An assignment of students to hospital is stable if neither of the following situation arises:

1. First type of instability: There are students s and s', and a hospital h, so that:
   - s is assigned to h, and
   - s' is assigned to no hospital, and
   - h prefers s' to s.
2. Second type of instability: There are students s and s', and hospitals h and h', so that:

- s is assigned to h, and
- s' is assigned to h', and
- h prefers s' to s, and
- s' prefers h to h'.

Input

The input is an instance of the national resident problem in a text file of the following format:

```
m                                    <- Number of hospitals
n                                    <- Number of students
s0^1  s1^1  s2^1  s3^1 ...  sn^1      <- Preference of the 1st hospital (most preferred first, s0^1 i
s0^2  s1^2  s2^2  s3^2 ...  sn^2      <- Preference of the 2nd hospital (most preferred first, s0^2 i
s0^3  s1^3  s2^3  s3^3 ...  sn^3      <- Preference of the 3rd hospital (most preferred first, s0^3 i
s0^4  s1^4  s2^4  s3^4 ...  sn^4      <- Preference of the 4th hospital (most preferred first, s0^4 i
.
.
.
s0^m  s1^m  s2^m  s3^m ...  sn^m      <- Preference of the mth hospital (most preferred first, s0^m i
h1^1  h2^1  h3^1 ...  hm^1        <- Preference of the 1st student (most preferred first)
h1^2  h2^2  h3^2 ...  hm^2        <- Preference of the 2nd student (most preferred first)
h1^3  h2^3  h3^3 ...  hm^3        <- Preference of the 3rd student (most preferred first)
h1^4  h2^4  h3^4 ...  hm^4        <- Preference of the 4th student (most preferred first)

.
.
.
h1^n  h2^n  h3^n ...  hm^n        <- Preference of the nth student (most preferred first)
```

For example

```
3                                <- Number of hospitals
5                                <- Number of students
1   2   3   5   1   4            <- Preference of the 1st hospital (most preferred first with first i
1   5   1   2   4   3            <- Preference of the 2nd hospital (most preferred first with first i
2   5   2   3   1   4            <- Preference of the 3rd hospital (most preferred first with first i
2   1   3                        <- Preference of the 1st student (most preferred first)
3   2   1                        <- Preference of the 1st student (most preferred first)
3   1   2                        <- Preference of the 1st student (most preferred first)
1   2   3                        <- Preference of the 1st student (most preferred first)
1   2   3                        <- Preference of the 1st student (most preferred first)
```

Note

A hospital can have more than one available slot which is stored in the first index of its preference list.

Output

The output is an instance of stable matchings for the input in a text file of the following format:

```
(h1, s1)            <- Pairing of the form (h,s)
(h2, s2)            <- Pairing of the form (h,s)
(h3, s3)            <- Pairing of the form (h,s)
.
.
.
(hn, sn)            <- Pairing of the form (h,s)
```

For example:

```
(1, 5)                           <- Pairing of the form (h,s)
(2, 1)
(3, 2)
(3, 3)
```

Note

We note that we will assume that a student that is not assigned to any hospital is not part of the output. **More importantly**, please note that there is more than one possible stable assignments possible for any given input. I.e. even if your algorithm is correct, its output might not match the given sample output. However, as long as your output is a stable assignment, you should be fine.

Hint

The best possible algorithm for this problems that we are aware of runs in time $O(m \cdot n)$.

! Note

Both the input and output parsers in each of the three languages are already written for you.
Note that you have to work with the input data structures provided (which will come pre-loaded with the data from an input file).

! Addition is the only change you should make

Irrespective of what language you use, you will have to submit just one file. That file will come pre-populated with some stuff in it. You should **not** change any of those things because if you do you might break what the grader expects and end up with a zero on the question. You should of course add stuff to it (including helper functions and data structures as you see fit).

Java | Python | C++

Download Java Skeleton Code (HW2Java.zip)

Directory Structure

```
├── Driver.java
├── HW2_Student_Solution.java
├── Match.java
├── testcases/
│   ├── input1.txt
│   ├── input2.txt
│   ├── input5.txt
│   ├── output1.txt
│   ├── output2.txt
│   └── output5.txt
```

You are given three coding files: `Driver.java`, `Match.java` and `HW2_Student_Solution.java`. `Driver.java` takes the input file, parses it and creates an instance of the class and prints result in output file. You only need to update the `HW2_Student_Solution.java` file. You may write your own helper methods and data structures in it.

The testcases folder has 3 input files and their corresponding output files for your reference. We will use these three input files (and seven others) in our autograding.

Method you need to write:

5.

```
/**
 * This method must be filled in by you. You may add other methods and subclasses
 * but they must remain within the HW2_Student_Solution class.
 * @return Your set of stable matches. Order does not matter.
 */
public ArrayList<Match>> getMatches() {

  return new ArrayList<Match>>();
}
```

The `HW2_Student_Solution` class has 4 instance variables.

- `_nHospital` which is of type int and stores number of hospitals.
- `_nStudent` which is of type int and stores number of students.
- `_hospitalList` which is of type `HashMap<Integer, ArrayList<Integer>>` and stores the preference lists of hospitals. Please note that the front of the `ArrayList<Integer>` (index 0) denotes the number of available slots.

- `_studentLIst` which is of type `HashMap<Integer, ArrayList<Integer>>` and stores the preference lists of students. Please note that the front of the `ArrayList<Integer>` (index 0) denotes the most preferred hospital.

The Other files

`Match.java` defines the `Match` class. This should be fairly intuitive. Below is the entire content of the class:

```
public class Match {
    public Integer _hospital;
    public Integer _student;
    Match(Integer hospital, Integer student) {
        _hospital = hospital;
        _student = student;
    }
    public String toString() {
        return "(" + _hospital + ", " + _student + ")";
    }

    public Integer getHospital(){
        return this._hospital;
    }

    public Integer getStudent(){
        return this._student;
    }

    /**
     * used to compare if two matches are the same
     * @param match that will be compared
     * @return true if they share the same hospital and student
     */
    public boolean equals(Match aMatch) {
        return ((this._hospital.equals(aMatch.getHospital())) && (this._student.equals(aMatch.getStudent()))) ;
    }
}
```

Compiling and executing from command line:

Assuming you're in the same directory level as `Driver.java`. Run `javac Driver.java` to compile.

To execute your code on `input1.txt`, run `java Driver testcases/input1.txt output.txt`. The output will be written to `output.txt`.

Submission

You only need to submit `HW2_Student_Solution.java` to Autolab.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading).

Question 2 (Home wrecker) [45 points]

The Problem

This problem is inspired by a question raised by Devashish in class. As Devashish's question pointed out, the stable marriage problem does not handles "divorces." This is because we assume everyone is interested in everyone else of the opposite gender and we assume that the preferences *do not change*.

In this problem, we will see the effect of changes in preferences in the outcome of the Gale-Shapley algorithm (for this problem you can assume the version of the Gale-Shapley algorithm that we did in class where the women do all the proposing).

Given an instance of the stable marriage problem (i.e. set of men M and the set of women W along with their preference lists: L_m and L_w for every $m \in M$ and $w \in W$ respectively), call a man $m \in M$ a **home-wrecker** if the following property holds. There exists an L'_m such that if m changes his preference list to L'_m (from L_m) then the Gale-Shapley algorithm matches everyone to someone else. In other words, let S_{orig} be the stable marriage output by the Gale-Shapley algorithm for the original input and S_{new} be the stable marriage output by the Gale-Shapley algorithm for the new instance of the problem where m's preference list is replaced by L'_m (but everyone else has the same preference list as before). Then $S_{\text{orig}} \cap S_{\text{new}} = \emptyset$.

For **every** integer $n \geq 2$ prove the following: There exists an instance of the stable marriage problem with n men and n women such that there is a man who is a home-wrecker.

Note

To get full credit you must present an example for every $n \geq 2$, that is, you have to present a "family" of examples (i.e. for each $n \geq 2$, you have to present the original $2n$ preference lists, the identity of the home-wrecker m and his changed preference list L'_m). Further, your proof argument should work for every value of $n \geq 2$.

You have to give both the idea and the details on your instance for each $n \geq 2$ but you only need to give a proof idea for the correctness of your instance.

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible. Also make sure that you preview your upload on Autolab to make sure it was uploaded correctly.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to this problem:

1. `Proof idea: 23` points for outlining your instances for every $n \geq 2$.
2. `Proof details: 22` points for a detailed description of your instance for each $n \geq 2$. (This is worth 17 points.) You also have to argue why your family of instance proves what is needed to be proven-- this can be at the level of proof idea: proof details for this part are not needed. (This part is worth 5 points.) However, note that if the grader cannot understand why your construction works immediately, then you might (and most probably will) lose points. So it is in your best interest to make sure that is enough intuition given on why your construction works.
3. `Note:` If your solution only consists of examples for *specific* values of $n \geq 2$, then you get at most 7 points for the entire problem.

! Note

If you do not have separated out proof idea and proof details, you will get a zero(0) irrespective of the technical correctness of your solution..

Questions 3 (Big G is in town) [15 points]

The Problem

The `Big G` company in the bay area decides it has not been doing enough to hire CSE grads from UB so it decides to do an exclusive recruitment drive for UB students. The `Big G` decides to fly over n CSE majors from UB to the bay area during December for on-site interview on a single day. The company sets up m slots in the day and arranges for n `Big G` engineers to interview the n UB CSE majors. (You can and should assume that $m > n$.) The fabulous scheduling algorithms at `Big G`'s offices draw up a schedule for each of the n majors so that the following conditions are satisfied:

- Each CSE major talks with every `Big G` engineer exactly once;
- No two CSE majors meet the same `Big G` engineer in the same time slot; and
- No two `Big G` engineers meet the same CSE major in the same time slot.

In between the schedule being fixed and the CSE majors being flown over, the `Big G` engineers were very impressed with the CVs of the CSE majors (including, ahem, their performance in CSE 331) and decide that `Big G` should hire all of the n UB CSE majors. They decide as a group that it would make sense to assign each CSE major C to a `Big G` engineer E in such a way that after C meets E during her/his scheduled slot, all of C's and E's subsequent meetings are canceled. Given that this is December, the `Big G` engineers figure that taking the CSE majors out to the nice farmer market at the ferry building in San Francisco during a sunny December day would be a good way to entice the CSE majors to the bay area.

In other words, the goal for each engineer E and the major C who gets assigned to her/him, is to **truncate** both of their schedules after their meeting and cancel all subsequent meeting, so that no major gets **stood-up**. A major C is stood-up if when C arrives to meet with E on her/his truncated schedule and E has already left for the day with some other major C'.

Design an *efficient* algorithm that always finds a valid truncation of the original schedules so that no CSE major gets stood-up.

To help you get a grasp of the problem, consider the following example for $n = 2$ and $m = 4$. Let the majors be C_1 and C_2 and the `Big G` engineers be E_1 and E_2. Suppose C_1's original schedule is

$$E_1\text{; free; } E_2\text{; free}$$

and C_2's original schedule is

$$\text{free; } E_1\text{; free; } E_2.$$

(In the above schedules "free" means that the student is not meeting any engineer.) In this case the (only) valid truncation is for C_1 to get assigned to E_2 in the third slot and for C_2 to get assigned to E_1 in the second slot. (And as a bonus all four get to have dinner!)

Hint

In real life, you will almost never come across a problem whose description will match exactly with one you will see in this course. More often, you will come across problems that you have seen earlier *but* are stated in a way that don't look like the version you have seen earlier. *One way* to solve this problem would be to to simulate that situation. In algorithms-speak, you can to *reduce* the problem here to one that you have seen already.

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible. Also make sure that you preview your upload on Autolab to make sure it was uploaded correctly.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to this problem:

1. `Algorithm idea: 4` points for explaining the idea behind your algorithm or a reduction to a familiar problem.
2. `Algorithm details: 3` points for providing the specific details of the algorithm or the reduction.
3. `Proof idea: 4` points for a proof idea that you can always achieve a valid truncation.
4. `Proof details: 4` points for providing the proof details.

! Note

If you do not have separated out proof/algorithm idea and proof/algorithm details, you will get a zero(0) irrespective of the technical correctness of your solution..