Rajat Bhageria
HW 1 Writeup
CIS 350

1) I created two main classes. The first is of course is the Main class that has a main() method, a Reader() method that sets reads in the text file by taking in a .txt file, a Logger method that sets up the log file by taking in the name of the log file to print to, and a MainMenu() method that sets up the main menu. In the Main class I'm essentially reading in the text file and logger file, creating an instance of a Parse (a class where all the analysis and algorithms are happening), interacting with the user to take his/her response, and of course returning/printing the answers.

The second class I created is called is called Parse. In Parse, I initially take in the data and basically split each line by the comma into an array. Then I create an ArrayList of arrays (each line in the text file). I now have methods for each question, coursesByInstructor() that takes in the name of the instructor you're looking for, lowestDifficultyRatio() that returns a TreeMap mapping the course quality/difficulty ratio to the course name, and a coursesAboveQualityRating() method that returns a TreeMap mapping the course quality with the course name. The latter two methods are heavily dependent on a helper method I created called treeGenerator() that takes the ArrayList of arrays I created at the very beginning, and essentially generates a ratioTree for the ratio problem and a qualityTree for the quality problem. I used the TreeMap data structure because it automatically sorts the data alphanumerically.

2) The way I structured the code is that initially I take the data about a particular course and put it into an array so that the first element in the array is the course name, the second element is the instructor, etc. Then I put each array about a particular course into an ArrayList with data about all the courses. Thus, even if the data was in a different form initially (such as JSON instead of the .txt file separated by commas), as long as the programmer was able to convert the data into the ArrayList of arrays, all the methods would work appropriately.

I created a two classes--one that the user would interact with and one that holds all the algorithms. As a result, there's no ability to interact with the user in the Parse class. Plus, this allows the user to change languages and interact with the user in a different language, even while all the algorithms stay constant.

The Main class is the only class that reads in the input file. Thus, the algorithms in the Parse class are not dependent on the format of the input file. It would be completely fine if the input file was in a different form initially because the algorithms would stay the same.

3) I factored out the common treeGenerator() method from both the lowestDifficultyRatio() and the coursesAboveQualityRating() methods. In the latter two methods, all I do is just call treeGenerator() once and I get a tree that maps either course quality or course ratio to the course name. This saved a lot of lines of code.

Within the Parse class, I separated the code for each question into a different method. That means that adding a new question just means creating a method, and subtracting a previous question just means subtracting a previous method.

In the Main class I made different methods for each major function, rather than just adding everything to the main() method. For example, the MainMenu() method has all the printing for the main menu, the Reader() method creates the Scanner, and the Logger() method sets up the logger. As a result of this, the code is a lot more readable.

In terms of the treeGenerator(), I split the method into a treeGeneratorHelper() that creates a editedCourseInfo map mapping the course name to an arraylist conaining the number of students, the total course quality, and the total course difficulty. Then I use the treeGenerator() function that calls treeGeneratorHelper() to generate a map that maps the course difficulty/quality ratio to the name and a map that maps the course quality to the name. This is good design because each method has a different purpose and is easier to read. Plus, this designs allows the programmer to add different functionality very easily in the treeGenerator() method; for example, using this method, it would be very easy for the programmer to create a map that maps the course difficulty to the course name; this could be used to return all the courses less than a certain difficulty.

For the second question (ratio), I returned a tree mapping either the ratio to the course name to the Main class. Then in the front end, I went though and printed the top 5 courses. This is good style because to return the top 10 lowest ratio courses, all I have to do is change a for loop from 5 to 10 in the frontend; I don't need to change any algorithms.

4) For the first problem (professors), I went through and made sure that made sure that all substrings of a Professor's name returned that professor. Then, I manually went through to make sure that all the courses that a Professor taught were being displayed; for example, I searched "rajiv," "gandhi" and "iv gan" to make sure that all of his courses "RAJIV GANDHI, CIS-121-001, CIS-160-001, CIS-320-001."

For the second problem, I manually calculated the difficulty/quality ratio of multiple courses to make sure my algorithm returned the correct ratios. Then, I made sure that there weren't any anomalies in terms of courses that should be one of the lowest ratios but weren't.

For the third question, I inputted queries like 3.5 and made sure that all the courses with ratings above 3.5 were returned. Then I did the same thing for different ratings like 3.7 and 3.9.

**Known Bugs?**
The logger is printing a lot of time stamps. Although the answers are all correct, these are a bit distracting.