Download the HW1 Skeleton before you begin.

Homework Overview

Vast amounts of digital data are generated each day, but raw data are often not immediately “usable”. Instead, we are interested in the information content of the data: what patterns are captured? This assignment covers a few useful tools for acquiring, cleaning, storing, and visualizing datasets.

Why specific versions of software are used in homework assignments? Using specific versions of software in homework assignments enables us to grade and provide immediate feedback to the large number of students in the course (1000+ OMS students, 250+ Atlanta students). Autograders are used to grade students' code submissions, and to ensure that these autograders can grade all submissions, we need to know the specific versions of software that students use. This is because different versions of software can have different features, and also to make sure that the autograders can detect potential errors that may occur in different libraries and provide students with appropriate feedback to resolve them. Continuously updating assignments to keep up with the latest versions of technology is a significant undertaking, so we carefully select which aspects of our autograders to update, to balance the workload for our course staff and provide a positive learning experience for students. As a result, you may see that certain assignment questions require the use of “older" versions of software or specific libraries.

Q1 [40 points] Collect data from TMDb to build a co-actor network

Goal	Collect data using an API for *The Movie Database* (TMDb). Construct a graph representation of this data that shows which actors have acted together in various movies. We use the word “graph” and “network” interchangeably.
Technology	• Python **3.7.x only** (question and autograder developed and tested for these versions). It is possible that more recent versions may also work, but we do not officially support them (it is possible that your code written with newer versions may break the autograder). • TMDb API version 3
Allowed Libraries	The Python Standard Library **only**. All other libraries (including and not limited to Pandas, Numpy, and Requests) are **NOT** allowed. Providing a consistent autograder experience for all students vastly outweighs the marginal utility of extending the scope of supported libraries. For example, urllib can be easily used instead of Requests in solving this question.
Max runtime	10 minutes. Submissions exceeding this will receive **zero** credit.
Deliverables	[Gradescope] • **Q1.py**: The completed Python file • **nodes.csv**: The csv file containing nodes • **edges.csv**: The csv file containing edges

For this question, you will use and submit a Python file**.** Complete all tasks according to the instructions

found in **`Q1.py`** to complete the Graph class, the TMDbAPIUtils class, and the one global function. The Graph class will serve as a re-usable way to represent and write out your collected graph data. The TMDbAPIUtils class will be used to work with the TMDB API for data retrieval.

Tasks and point breakdown

a) [10 pts] Implementation of the Graph class according to the instructions in **`Q1.py`**.
   - **The graph is undirected,** thus **{a, b} and {b, a}** refer to the **same undirected edge** in the graph; **keep only either {a, b} or {b, a}** in the Graph object. A node's degree is the number of (undirected) edges incident on it. In/ out-degrees are not defined for undirected graphs.

b) [10 pts] Implementation of the `TMDbAPIUtils` class according to instructions in **`Q1.py`**. Use version 3 of the TMDb API to download data about actors and their co-actors. To use the API:
   - Create a TMDb account and follow the instructions on this document to obtain an authentication token.
   - Refer to the TMDB API Documentation as you work on this question.

c) [20 pts] Producing correct **`nodes.csv`** and **`edges.csv`**.
   - As mentioned in the Python file, if an actor name has comma characters (","), remove those characters before writing that name into the csv files.

Q2 [35 points] SQLite

SQLite is a lightweight, serverless, embedded database that can easily handle multiple gigabytes of data. It is one of the world's most popular embedded database systems. It is convenient to share data stored in an SQLite database — just one cross-platform file which does not need to be parsed explicitly (unlike CSV files, which must be parsed).

You will modify the given **Q2.py** file by adding SQL statements to it. We suggest that you consider testing your SQL locally on your computer using interactive tools to speed up testing and debugging, such as DB Browser for SQLite.

Goal	Construct a TMDb database in SQLite. Partition and combine information within tables to answer questions.
Technology	• SQLite release 3.22. As some students have encountered challenges installing earlier versions of SQLite, we have furthered verified that this question can be completed with SQLite version 3.39.2 on our local machine. It is possible that other SQLite versions may also work. **Note:** while window functions may work in some versions of SQLite, they **DO NOT** work in v3.22. • Python **3.6.x only** (question developed and tested for these versions). It is possible that more recent versions may also work, but we do not officially support them.
Allowed Libraries	**Do not modify import statements**. Everything you need to complete this question has been imported for you. **Do not** use other libraries for this question.
Max runtime	10 minutes. Submissions exceeding this will receive **zero** credit.
Deliverables	[Gradescope] **`Q2.py`**: Modified file containing all the SQL statements you have used to answer parts a - h in the proper sequence.

Tasks and point breakdown

NOTE: A sample class has been provided to show example SQL statements; you can turn off this output by changing the global variable SHOW from True to False. This **must** be set to *False* before uploading to Gradescope.

NOTE: In this question, you must only use INNER JOIN when performing a join between two tables, except for part g. Other types of joins may result in incorrect results.

GTusername — update the method GTusername with your credentials

a. [9 points] *Create tables and import data*.
   i. [2 points] Create two tables (via two separate methods, part_ai_1 and part_ai_2, in **`Q2.py`**) named `movies` and `movie_cast` with columns having the indicated data types:
      1. movies
         1. id (integer)

```
        2. title (text)
        3. score (real)
    2. movie_cast
        1. movie_id (integer)
        2. cast_id (integer)
        3. cast_name (text)
        4. birthday (text)
        5. popularity (real)
```

ii. [2 points] Import the provided **movies.csv** file into the `movies` table and **movie_cast.csv** into the `movie_cast` table

1. Write Python code that imports the `.csv` files into the individual tables. This will include looping though the file and using the **'INSERT INTO'** SQL command. You **must** only use relative paths while importing files since absolute/local paths are specific locations that exist only on your computer and will cause the auto-grader to fail.

iii. [5 points] *Vertical Database Partitioning.* Database partitioning is an important technique that divides large tables into smaller tables, which may help speed up queries. Create a new table `cast_bio` from the `movie_cast` table (i.e., columns in `cast_bio` will be a subset of those in `movie_cast`). Do not edit the `movie_cast` table. Be sure that the values are unique when inserting into the new `cast_bio` table. Read this page for an example of vertical database partitioning.

```
    cast_bio
        1. cast_id (integer)
        2. cast_name (text)
        3. birthday (text)
        4. popularity (real)
```

b. [1 point] *Create indexes.* Create the following indexes. Indexes increase data retrieval speed; though the speed improvement may be negligible for this small database, it is significant for larger databases.

1. `movie_index` for the `id` column in `movies` table
2. `cast_index` for the `cast_id` column in `movie_cast` table
3. `cast_bio_index` for the `cast_id` column in `cast_bio` table

c. [3 points] *Calculate a proportion*. Find the proportion of actors who are born between 1965 and 1985 (both years included). Consider the actors with birthday as 'None' to be born before 1965 or after 1985. The proportion should be calculated as a percentage and should only be based on the total number of rows in the cast_bio table. Format all decimals to two places using `printf()`. Do **NOT** use the `ROUND()` function as in some rare cases it works differently on different platforms.

Output format and example value:

```
7.70
```

7

d. [4 points] *Find the most prolific actors*. List 5 cast members with the highest number of movie appearances that have a popularity > 10. Sort the results by the number of appearances in descending order, then by `cast_name` in alphabetical order.

Output format and example row values (`cast_name,appearance_count`):

```
Harrison Ford,2
```

e. [4 points] *Find the highest scoring movies with the smallest cast*. List the 5 highest-scoring movies that have the fewest cast members. Sort the intermediate result by score in descending order, then by number of cast members in ascending order, then by movie name in alphabetical order. Format all decimals to two places using `printf()`.

Output format and example values (`movie_title,movie_score,cast_count`):

```
Star Wars: Holiday Special,75.01,12
Games,58.49,33
```

f. [4 points] *Get high scoring actors.* Find the top ten cast members who have the highest average movie scores. Format all decimals to two decimal places using `printf()`.

- Sort the output by average score in descending order, then by `cast_name` in alphabetical order.
- **First** exclude movies with score <25 in the average score calculation.
- **Next** include only cast members who have appeared in three or more movies with score >= 25.

Output format and example value (`cast_id,cast_name,average_score`):

```
8822,Julia Roberts,53.00
```

g. [6 points] *Creating views.* Create a view (virtual table) called `good_collaboration` that lists pairs of actors who have had a good collaboration as defined here. Each row in the view describes one pair of actors who appeared in at least 3 movies together AND the average score of these movies is >= 40.

The view should have the format:

```
good_collaboration(
      cast_member_id1,
      cast_member_id2,
      movie_count,
      average_movie_score)
```

For symmetrical or mirror pairs, only keep the row in which `cast_member_id1` has a lower numeric value. For example, for ID pairs (1, 2) and (2, 1), keep the row with IDs (1, 2). There should not be any "self-pair" where the value of `cast_member_id1` is the same as that of `cast_member_id2`.

Remember that creating a view will not produce any output, so you should test your view with a

few simple select statements during development. One such test has already been added to the code as part of the auto-grading.

NOTE: Do not submit any code that creates a 'TEMP' or 'TEMPORARY' view that you may have used for testing.

Optional Reading: Why create views?

i. [4 points] *Find the best collaborators.* Get the 5 cast members with the highest average scores from the `good_collaboration` view, and call this score the `collaboration_score`. This score is the average of the `average_movie_score` corresponding to each cast member, including actors in `cast_member_id1` as well as `cast_member_id2`. Format all decimals to two places using `printf()`.

- Order your output by `collaboration_score` (before formatting) in descending order, then by `cast_name` alphabetically.

Output format and example values(`cast_id,cast_name,collaboration_score`):

```
2,Mark Hamil,99.32
1920,Winoa Ryder,88.32
```

h. [4 points] SQLite supports simple but powerful Full Text Search (FTS) for fast text-based querying (FTS documentation). Import movie overview data from the **movie_overview.csv** into a new FTS table called `movie_overview` with the schema:

```
movie_overview
    id (integer)
    overview (text)
```

NOTE: Create the table using **fts3** or **fts4** only. Also note that keywords like NEAR, AND, OR and NOT are case sensitive in FTS queries.

NOTE: If you have issues that fts is not enabled, try the following steps
1) Go to sqlite3 downloads page: https://www.sqlite.org/download.html
2) Download the dll file for your system
3) Navigate to your python packages folder, e.g.,
C:\Users\... ...\Anaconda3\pkgs\sqlite-3.29.0-he774522_0\Library\bin
4) Drop the downloaded .dll file in the bin.
5) In your IDE, import sqlite3 again, fts should be enabled."

i. [1 point] Count the number of movies whose `overview` field contains the word 'fight'. Matches are not case sensitive. Match full words, not word parts/sub-strings.
Example:

- *Allowed: 'FIGHT', 'Fight', 'fight', 'fight.'*
- *Disallowed: 'gunfight', 'fighting', etc.*

Output format and example value:

```
12
```

ii. [2 points] Count the number of movies that contain the terms 'space' and 'program' in the `overview` field with no more than 5 intervening terms in between. Matches are not case sensitive. As you did in h(i)(1), match full words, not word parts/sub-strings.

Example:

- *Allowed: 'In Space there was a program', 'In this space program'*
- *Disallowed: 'In space you are not subjected to the laws of gravity. A program.', etc.*

Output format and example value:

```
6
```

10

Q3 [15 points] D3 (v5) Warmup

Read chapters 4-8 of Scott Murray's Interactive Data Visualization for the Web, 2nd edition (sign in using your GT account, e.g., jdoe3@gatech.edu). Briefly review chapters 1-3 if you need additional background on web development. **This reading provides important foundation** you will need for Homework 2. This question and the autograder have been developed and tested for D3 version 5 (v5), while the book covers D3 v4. What you learn from the book (v4) is transferable to v5 because v5 introduced few breaking changes. In Homework 2, you will work with D3 extensively.

Goal	Visualize temporal trends in movie releases using D3 to showcase how interactive, rather than static plots, can make data more visually appealing, engaging and easier to parse.
Technology	D3 Version 5 (included in the lib folder) Chrome 97.0 (or newer): the browser for grading your code Python http server (for local testing)
Allowed Libraries	D3 library is provided to you in the **lib** folder. You must **NOT** use any D3 libraries (d3*.js) other than the ones provided. In Gradescope, these libraries will be provided for you in the auto-grading environment.
Deliverables	[Gradescope] **`Q3.html:`** Modified file containing all html, javascript, and any css code required to produce the bar plot. Do not include the D3 libraries or q3.csv dataset.

NOTE the following important points:

1. You will need to setup an HTTP server to run your D3 visualizations as discussed in the D3 lecture (OMS students: the video "Week 5 - Data Visualization for the Web (D3) - Prerequisites: JavaScript and SVG". Campus students: see lecture PDF.). The easiest way is to use http.server for Python 3.x. **Run your local HTTP server in the hw1-skeleton/Q3 folder.**

2. We have provided sections of code along with comments in the skeleton to help you complete the implementation. While you do not need to remove them, you may need to write additional code to make things work.

3. All d3*.js files are provided in the **lib** folder and referenced using **relative paths** in your html file. For example, since the file "Q3/Q3.html" uses d3, its header contains:
<script type="text/javascript" src="lib/d3/d3.min.js"></script>
It is incorrect to use an absolute path such as:
<script type="text/javascript" src="http://d3js.org/d3.v5.min.js"></script>

The 3 files that are referenced are:
- `lib/d3/d3.min.js`
- `lib/d3-dsv/d3-dsv.min.js`
- `lib/d3-fetch/d3-fetch.min.js`

11

4. In your html / js code**,** use a **relative path** to read the dataset file**.** For example, since Q3 requires reading data from the `q3.csv` file, the path must be `"q3.csv"` and **NOT** an absolute path such as "C:/Users/polo/HW1-skeleton/Q3/q3.csv". Absolute (local) paths are specific locations that exist only on your computer, which means your code will **NOT** run on our machines when we grade (and you will lose points). **As file paths are case-sensitive, ensure that you correctly provide the relative path**. Gradescope will provide a copy of the q3.csv dataset using the same directory structure provided in the HW skeleton.

5. Load the data from `q3.csv` using D3 fetch methods. We recommend `d3.dsv()`. Handle any data conversions that might be needed, *e.g.,* strings that need to be converted to integer. See https://github.com/d3/d3-fetch#dsv**. Note: use the correct reference path name to load the data; the filename is case-sensitive.**

6. IMPORTANT: use the Margin Convention guide for specifying chart dimensions and layout. The autograder assumes your code has followed this convention.

submission.html : when run in a browser, it should display a vertical bar plot with the following specifications:

a. [3.5 points] The bar plot must display one bar per row in the `q3.csv` dataset. Each bar corresponds to the running total of movies for a given year. The height of each bar represents the running total. The bars are ordered by ascending time with the earliest observation at the far left. i.e., 1880, 1890, ..., 2000

b. [1 point] The bars must have the same fixed width, and there must be some space between two bars, so that the bars do not overlap.

c. [3 points] The plot must have visible X and Y axes that scale according to the generated bars. That is, the axes are driven by the data that they are representing. Likewise, the ticks on these axes must adjust automatically based on the values within the datasets, i.e., they must not be hard-coded. The x-axis must be a `<g>` element having the `id: "x_axis"` and the y-axis must be a `<g>` element having the `id: "y_axis"`.

d. [2 points] Set x-axis label to 'Year' and y-axis label to 'Running Total'. The x-axis label must be a `<text>` element having the `id: "x_axis_label"` and the y-axis label must be a `<text>` element having the `id: "y_axis_label"`.

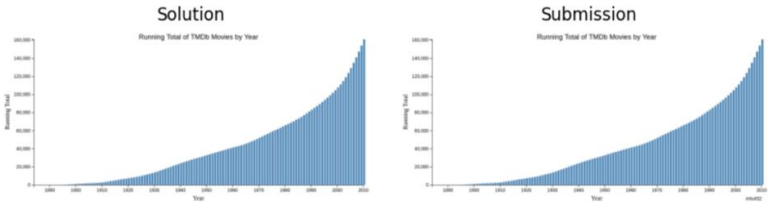
e. [1 point] Use a linear scale for the Y axis to represent the `running total` (recommended function: `d3.scaleLinear()`).

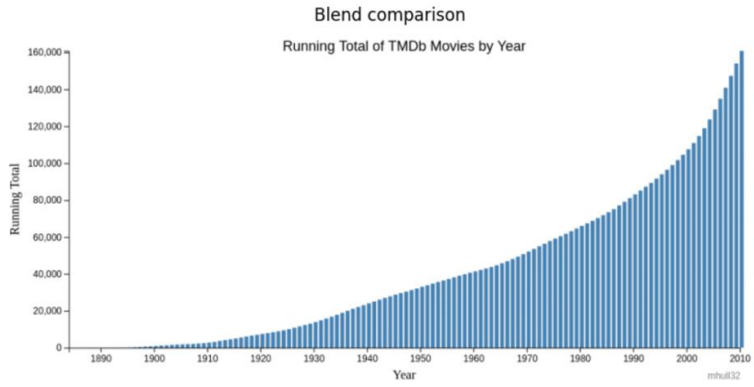
f. [3 points] Use a time scale for the x-axis to represent `year` (recommended function: `d3.scaleTime()`). It may be necessary to use time parsing / formatting when you load and display the `year` data. The axis would be overcrowded if you display every year value so set the x-axis ticks to display one tick for every 10 years.

g. [1 point] Set the HTML title tag and display a title for the plot. **Those two titles are independent of each other and need to be set separately.** Set the HTML title tag (i.e., `<title>` Running Total of TMDb Movies by Year `</title>`). Position the title "Running Total of TMDb Movies by Year" above the bar plot. The title must be a `<text>` element having the `id: "title"`

h. [0.5 points] Add your **GT username** (usually includes a mix of letters and numbers) to the area beneath the bottom-right of the plot (see example image). The GT username must be a `<text>` element having the `id: "credit"`

7. Gradescope will render your plot using Chrome and present you with a Dropbox link to view the screenshot of your plot with the solution plot in both a side-by-side and an overlay display.





The visual feedback helps you make adjustments and identify errors, *e.g.,* a blank plot likely indicates a serious error. It is not necessary that your design replicates the solution plot. However, the autograder **requires** the following DOM structure (including using **correct ids** for elements) and sizing attributes, so that it knows how your chart is built. We recommend using the Web Inspector to keep track of the DOM structure and debug. Based on our experience, most errors students encounter are due to incorrect DOM structures (including wrong ids). Make sure you have **strictly followed all instructions in this question**.

```
<svg id="svg1"> plot
      | width: 960
```

```
| height: 500
|
+-- <g id="container"> containing Q3.a plot elements
    |
    +-- <g id="bars"> containing bars
    |
    +-- <g id="x_axis"> x-axis
    |   |
    |   +-- (x-axis elements)
    |
    +-- <text id="x_axis_label"> x-axis label
    |
    +-- <g id="y_axis"> y-axis
    |   |
    |   +-- (y-axis elements)
    |
    +-- <text id="y_axis_label"> y-axis label
    |
    +-- <text id="credit"> GTUsername
    |
    +-- <text id="title"> chart title
```

Q4 [5 points] OpenRefine

Goal	Use OpenRefine to clean data from Mercari. Construct GREL queries to filter the entries in this dataset.
Technology	OpenRefine 3.6.2
Deliverables	[Gradescope] • **properties_clean.csv** : Export the final table as a csv file. • **changes.json** : Submit a list of changes made to file in json format. Go to 'Undo/Redo' Tab -> 'Extract' -> 'Export'. This downloads 'history.json' . Rename it to 'changes.json'. • **Q4Observations.txt** : A text file with answers to parts c.i, c.ii, c.iii, c.iv, c.v, c.vi. Provide each answer in a new line in the output format specified. Your file's final formatting should result in a .txt file that has each answer on a new line followed by one blank line.

OpenRefine is a Java application and requires Java JRE to run. However, OpenRefine v.3.6.2 comes with compatible Java version embedded with the installer. So, there is no need to install Java separately when working with this version.

Watch the videos on OpenRefine's homepage for an overview of its features. Then, download and install OpenRefine 3.6.2. The link to release 3.6.2 is https://github.com/OpenRefine/OpenRefine/releases/tag/3.6.2

a. Import Dataset
   - Run OpenRefine and point your browser at 127.0.0.1:3333.
   - We use a products dataset from Mercari, derived from a Kaggle competition (Mercari Price Suggestion Challenge). If you are interested in the details, visit the data description page. We have sampled a subset of the dataset provided as "properties.csv".
   - Choose "Create Project" → This Computer → `properties.csv`". Click "Next".
   - You will now see a preview of the data. Click "Create Project" at the upper right corner.

b. Clean/Refine the data
   NOTE: OpenRefine maintains a log of all changes. You can undo changes by the "Undo/Redo" button at the upper left corner. Follow the exact output format specified in every part below.

   i. [0.5 point] Select the `category_name` column and choose 'Facet by Blank' (Facet → Customized Facets → Facet by blank) to filter out the records that have blank values in this column. Provide the number of rows that return True in `Q4Observations.txt`. Exclude these rows.

   Output format and sample values:
   ```
   i.rows: 500
   ```

   ii. [1 point] Split the column `category_name` into multiple columns without removing the original column. For example, a row with "Kids/Toys/Dolls & Accessories" in the `category_name` column

would be split across the newly created columns as “Kids”, “Toys” and “Dolls & Accessories”. Use the existing functionality in OpenRefine that creates multiple columns from an existing column based on a separator (i.e., in this case ‘/’) and does not remove the original `category_name` column. Provide the number of new columns that are created by this operation, excluding the original `category_name` column.

Output format and sample values:
`ii.columns: 10`

NOTE: There are many possible ways to split the data. While we have provided one way to accomplish this in step ii, some methods could create columns that are completely empty. In this dataset, none of the new columns should be completely empty. Therefore, to validate your output, we recommend you verify that there are no columns that are completely empty by sorting and checking for null values.

iii. [0.5 points] Select the column `name` and apply the Text Facet (Facet → Text Facet). Cluster by using (Edit Cells → Cluster and Edit …) this opens a window where you can choose different “methods” and “keying functions” to use while clustering. Choose the keying function that produces the smallest number of clusters under the “Key Collision” method. Click ‘Select All’ and ‘Merge Selected & Close’. Provide the name of the keying function and the number of clusters produced.

Output format and sample values:
`iii.function: fingerprint, 200`

NOTE: Use the default Ngram size when testing Ngram-fingerprint.

iv. [1 point] Replace the null values in the `brand_name` column with the text “Unknown” (Edit Cells -> Transform). Provide the expression used.

Output format and sample values:
`iv.GREL_categoryname: endsWith("food", "ood")`

v. [0.5 point] Create a new column `high_priced` with the values 0 or 1 based on the “price” column with the following conditions: if the price is greater than 90, `high_priced` should be set as 1, else 0. Provide the GREL expression used to perform this.

Output format and sample values:
`v.GREL_highpriced: endsWith("food", "ood")`

vi. [1.5 points] Create a new column `has_offer` with the values 0 or 1 based on the `item_description` column with the following conditions: If it contains the text “discount” or “offer” or “sale”, then set the value in `has_offer` as 1, else 0. Provide the GREL expression used to perform this. Convert the text to lowercase in the GREL expression before you search for the terms.

Output format and sample values:

```
vi.GREL_hasoffer: endsWith("food", "ood")
```

Q5 [5 points] Introduction to Python Flask

Flask is a lightweight web application framework written in Python that provides you with tools, libraries and technologies to quickly build a web application and scale up as needed.

You will modify the given file: **wrangling_scripts/`Q5.py`**

Goal	Build a web application that displays a table of TMDb data on a single-page website using Flask.
Technology	Python **3.7.x only** (question developed and tested for these versions) Flask
Allowed Libraries	Python standard libraries Libraries already included in Q5.py Any other libraries (including but not limited to Pandas and NumPy) are **NOT** allowed in this assignment
Deliverables	[Gradescope] **`Q5.py:`** Completed Python file with your changes

Username() - Update the username() method inside Q5.py by including your GTUsername.

- Install Flask on your machine by running `pip install Flask`
    a. You can optionally create a virtual environment by following the steps here. Creating a virtual environment is purely optional and can be skipped.

- To run the code, navigate to the Q5 folder in your terminal/command prompt and execute the following command: `python run.py`. After running the command go to http://127.0.0.1:3001/ on your browser. This will open up index.html, showing a table in which the rows returned by `data_wrangling()` are displayed.

- You must solve the following 2 sub-questions:
    a. [2 points] Read and store the first 100 rows in a table using the data_wrangling() method.

    NOTE: The skeleton code by default reads all the rows from movies.csv. You must add the required code to ensure reading only the **first** 100 data rows. The skeleton code already handles reading the table header for you.

    b. [3 points]: Sort this table in *descending* order of the values i.e., with larger values at the top and smaller values at the bottom of the table in the last (3rd) column. Note that this column needs to be returned as a string for the autograder but sorting may require float casting.