ECE241 PROJECT 1: Sorting and Searching
Due: October 25, 2018, 11PM on Moodle

Introduction:

In today's Internet-dominated world we take for granted the ability of computers to search through vast quantities of data to find information that interests us. The ability of search engines to find data quickly is the result of many programmers' and algorithm designers' efforts over a number of years. In this project, you will have the opportunity to join the ranks of this prestigious group by developing your algorithms and codes for a Million Song Dataset.
which will both sort and search through a database of information related to songs. This data, which is a subset of the, provides an example of the mass quantities of data that are evaluated by computers billions of times a day.
In this assignment, you will read in a subset of the Million Song Dataset from an Excel file. It contains information around 10,000 songs which contains several fields such as title, artist, song duration, track ID. Then You will then perform search and sort operations on the data to demonstrate the performance of different sorting and searching algorithms.

Task Overview

In this project, you will perform the following specific tasks.

1. Manage your data in Python classes and objects. For example, you should manage songs in the Song object and song library in SongLibrary.
    a. In the Song class, you should have variables for the different attributes of the songs (title, artist, song duration, track ID).
    b. In the SongLibrary class, you should have basic information about the library which usually contains an array for all the Song objects, an integer that indicates how many songs in the library and a Boolean variable that shows whether the library is sorted or not. In order to make it efficient to search the songs we need, we also need a binary search tree (BST) for the song library based on the song title.
2. Read in the database of songs using the ReadSongArray method in SongLibrary class. The method takes the input data file name and loads the songs in the file to the library. Make sure the order of the songs is the **same** as the one in the input file.
3. Implement a linear search algorithm to search the songs based on either song title or artist. Return the number of the songs found in the song array that satisfies your search. The

search function should have two parameters, one for your query and one for the attribute (string as ‘title’ or ‘artist’).

4. Implement a QuickSort algorithm to sort the song database based on the song title. The sorted songs should be stored in the same song array. Make sure you change the status of the Boolean variable that shows whether the library is sorted or not.
5. Implement a function to build a balanced BST of the songs based on the song title. Record the time spent on building the BST. Ideally the height of a balanced BST for 10,000 songs are 14. In this task, you need to build a tree with height **AT MOST 24.**
6. Implement a search function for song title on BST.

7. Perform a linear search in the sorted database for the 100 songs. You can a random number generator to arbitrarily select 100 song titles. Record the average time spent on the linear search.
8. Perform a search of the same 100 songs in the BST. Record the average time spent.
9. In order to perform a search on BST, we need to spend additional time to build the BST. If we have many songs to search, it is worthwhile for the additional time. Using the information calculated above (Task 5, 7, 8), determine how many linear searches $\boldsymbol{n}$ would be necessary before the overall run time of the linear searches would be more than the combined total of building a BST (only performed once) and $\boldsymbol{n}$ searches in the BST.
10. Plot a cumulative distribution function (CDF) curve based on the song durations of the 10,000 songs.

Hints and suggestions

Successfully completing the project and achieving a good grade requires completing the project as described above and clearly commenting the code. As always, it makes sense to start the project early. Unless you are an amazing programmer, you probably won’t be able to finish in one day. Build your project code step by step. For example, verify that you have successfully read in the database before attempting a linear search. Then, make sure the linear search works before writing and testing code for QuickSort, BST, etc. Additional hints are as follows:

1. Make sure that there are NO **print** statements in the code you are attempting to time. The use of these statements will negatively affect recorded time values and lead to incorrect results. Your submitted QuickSort, linear search, and search on BST methods should not include these statements.
2. For each line of the song record (“0,Qing Yi Shi,Leon Lai,203.38893,5237536”), you can split it based on the ‘,’.

3. Use the *random* module to identify the 100 random titles for searching. You can use the random numbers to locate specific song indices in the song database array. Note that you must save these song titles in an additional array so you can perform the same search.
4. For a balanced BST, you can implement the AVL tree. Another solution you can try is to randomize the song title to insert. Theoretically, you can get a better balanced tree, but you need to test the random seed to make sure the tree height is less than 24. (You can consider to implement an additional function to compute the tree height.)
5. The *QuickSort*, BST, and *linear search* methods can follow in a similar format from the lecture.

What to submit:

For Task 1-6, you should submit your code to Gradescope for auto-grading. Remember to comment your code properly.

For Task 5, 7-10, put your recorded running time of building BST, the average time for linear search and search on BST into a Document (.doc or .pdf). Explain how you compute the number of searches $\boldsymbol{n}$ for task 10 in the document. For task 11, paste your CDF curve of the song durations in the document too. Submit the document in Moodle.

Reminder: The course honesty policy requires you to write all code yourself, except for the code that we give to you. Your submitted code will be compared with all other submitted code for the course to identify similarities. Note that our checking program is not confused by changed variable or method names.

Grading:

- Code works on Gradescope (70%)
- Assignment results (search number $\boldsymbol{n}$ and CDF curve) (20%)
- Program structure and comments (10%)