Music Search

Vocabulary

- **Array** – An **array** is a series of memory locations, or 'boxes', each of which holds a single item of data. All data in an array must be of the same data type.
- **Procedure** – A **procedure** is a set of commands that can be executed in order. It is similar to a function except that it does not need to return anything.

Notice

- A music library has been exported as a CSV file.
- The file contains a list of artist names, number of albums, and number of tracks.
- The file is sorted in ascending order of artist name.
- The first ten rows of data from the file are shown below:

Artist Name	Number of Albums	Number of Tracks
2Pac	7	35
30 Seconds to Mars	1	1
50 Cent	9	46
Action Bronson	1	3
Aerosmith	4	4
Akon	4	4
Alanis Morissette	1	1
Alexis Jordan	1	1
Alter Bridge	1	11
Amerie	1	1

You need to implement a class called **`MusicLibrary`** based on the documents listed below and finish some tasks listed after the function signature.

Notice

- The sort should be **ascending order** (from smallest to largest).
- You don't need to specify the secondary comparing rule, i.e. take default Python behavior if multiple elements share the same value.
- **Do not change the class attribute name and method name**, or it will not pass the autograder.
- You may **only change** the name of the **helper** function, and you **are allowed** to define your own **helper functions** if needed.

- You need to sort the data before running binary search and shuffle the data before sorting. Or the time result or search result will not be correct. If you use **`timeFunc`** correctly, these things will already be taken care of.

Testing

- When testing search methods, please sort the data by the target index before calling the search method.
- When testing sorting methods, please shuffle the data before calling the search method.
- The methods are decorated using Python decorators. Please familiar yourself with **`timeFunc`** that defines the decorator.
- A sample test function is provided in main function. Feel free to comment out the ones you haven't finished, or modify the test function based on your needs.
- **DO NOT** modify the decorator loader before the declaration of the class methods (starting with @), unless you are aware of what you are doing.

Submit

Please submit **music_search.py** to Gradescope

Notice

Test your implementation in the **`main()`** function, not the other place. Refer to the starter file **music_searchStarter.py** for the code.

Reminding tasks

1. Read and understand the logic about timing in **`timeFunc`**. Here is a link that may be helpful. Write some meaningful comments that reflect your understanding of it: https://www.guru99.com/timeit-python-examples.html
2. Finish your writing in the **comment** method on your time analysis of your sort and search operations. Based on this project, can you say that a binary search is more efficient than a sequential search? Can you say that one sort is more efficient than another? Why or why not?

Extra credit (10 points)

Tasks

1. In a file named **quick_sort.py**, write a method called **`quickSort`**.

   This method should take an argument named **`array`**(data type **`List`**), do an in-place sort, and return the sorted **`array`**. The array only contains **`int`** element.
   - Notice: The test case will include many(more than 10k) same values.
   - For example: `[2* 10k, 1,3]`
2. Write a method called **comment** which takes no arguments. **Print** your thoughts about the time complexity and space complexity of the sorting algorithms(more than 50 words).

Submit

Submit the **quick_sort.py** to Gradescope.