COT 6405 - Intro to Theory of Algorithms

Spring 2023

Project 1: Sorting Algorithms

Due: March 8, 11:59 PM

❖ Project Description

Sorting is used time and again in many day-to-day life activities. While brute force way to sort is easy to build and simple to apply in any situation, the cost of implementing brute force increases exponentially as the size and complexity of the data increases. So, in this project, you will implement various algorithms to efficiently sort a real-world dataset. In particular, you will be implementing 6 sorting algorithms namely Insertion sort, Selection sort, Quick sort, Heap sort, Shell sort, and Merge sort. And in addition, you will get to see how the time to compute differs for each algorithm.

❖ Project Package

For this project, you will be given a few files that you need to work with. Those files include the two dataset files, two skeleton code files, and one python code file to run the test cases. The testing python script tells you how many test cases your code passed and failed.

• Dataset

For this project you will be using the ***IMDb dataset***. The required data is extracted and provided as CSV files.

(1) ***imdb_dataset.csv*** file

The dataset file includes the following attributes as columns.

tconst – a unique identifier for each movie
primaryTitle – the popular title that is used for promotions
originalTitle – original title in the original language
StartYear – release year of the title/ start year of a series
runtimeMinutes – primary run time of title in minutes
genres – includes up to three genres associated with the title
averageRating – weighted average of all individual user ratings
numVotes – number of votes the title has received
ordering – a number to uniquely identify the rows for a given title

category – the category of job that person was in
seasonNumber – season number the episode belongs to
episodeNumber – episode number of tconst in the TV series
primaryName – name by which the person is most often credited
birthYear – in YYYY format
deathYear – in YYYY format if applicable, else '\N'
PrimaryProfession – top 3 professions of the person

(2) ***test_cases_1_2.csv*** file
This file contains a relatively smaller number of data samples than the 'imdb_dataset.csv' file since it will be only used for test case 1 and 2.

• Skeleton codes

(1) *sorting_algos.py*
This python skeleton code file has the functions defined for all the **6 sorting algorithms** you will be implementing. Detailed comments are provided on what you need to code and where to write the code. 'Need to Code' comments are provided at all places where you will need to code.

(2) *mystery_sorting.py*
This python skeleton code file needs to be used to implement **the last 3 test cases (test cases #13, #14, and #15)**. All the test cases and their implementation details are provided below. Comments on what you need to do and "Need to Code" comments are provided. Read the comments carefully and complete the code.

• Testing scripts

A testing script named '**testcases.py**' is provided. You do not have to write any code in this file. This file has all the test case functions called. And you can run this file to check and test your code as running this file gives you the run time for your algorithm implementations and how many test cases your code passed or failed.

❖ Project Instructions

`testcases.py:`

- This Python file will execute your codes and check whether the provided test cases are passed or failed.

`sorting_algos.py`

1. Start implementing the `sorting_algos.py` file.
   This file will be used for the test cases from #1 to #12.
   This file contains a function `sorting_algorithms()` which calls other sorting algorithm functions. This function chooses which sorting algorithm to call, to calculate the time complexity, and return statements are included in the code given to you.
   #Need to do – comments are added at places where you need to put in your code.
   - ***Need to Do:***
     At the start of the sorting_algortihms function, you need to read the given `imdb_dataset.csv` and extract the necessary data/columns.

2. To extract the columns needed, you will need to understand the test cases implemented using the code. The test cases are described in the following section, and the column names required for each test case are also mentioned. You will need to extract the necessary columns as a ***list of lists*****.** For example, if you need columns `'years'` and `'names'`, then you need to extract a list where the first element is again a list of **'tconst'** values (i.e. a `unique identifier for each movie`), the second element is a list of columns `'years',` and the third element is a list containing `'names'`.

3. **`quicksort()`** function is defined for you. It takes two arguments `arr` and `columns`. The first argument `arr` is a list containing the list of columns to be sorted. For example, when you need to sort 'startYear' column, arr should look like - [[tconst1, tconst2, tconst3….], [startYear1, startYear2, startYear3,…..]]. The second argument `columns` is a list of column names that are being sorted. Example: ['startYear']
   - ***Need to Do:***
     You need to write the code to implement quicksort algorithm.

4. **`selection_sort()`** function is defined for you in the skeleton code. It takes the same two arguments arr and columns.
   - ***Need to Do:***
     You need to write the code to implement selectionsort algorithm.

5. **`heap_sort()`** function is defined for you in the skeleton code.

- ***Need to Do:***
  You need to write the code to implement heapsort algorithm. There are two additional functions: **`build_max_heap()`** and **`max_heapify()`**.

6. **`shell_sort()`** function is defined for you in the skeleton code.
   - ***Need to Do:***
     You need to write the code to implement shellsort algorithm.

7. **`merge_sort()`** function is defined for you in the skeleton code.
   - ***Need to Do:***
     You need to write the code to implement mergesort algorithm. You also need to implement **`merge()`** function.

8. **`insertion_sort()`** function is defined for you in the skeleton code.
   - ***Need to Do:***
     You need to write the code to implement insertionsort algorithm.

9. For all the sorting algorithm functions you implement, you need to return a list of sorted **'tconst'** values. (movie IDs)

10. You are required to code the function **`data_filtering`** to create filtered datasets for test cases 7 to 10.
    - if **num == 1** -> filter data based on years (**years in range 1941 to 1955**)
    - if **num == 2** -> filter data based on genres (**genres are either 'Adventure' or 'Drama**')
    - if **num == 3** -> filter data based on primaryProfession (if **primaryProfession** column contains substrings **{'assistant_director', 'casting_director', 'art_director', 'cinematographer'}**)
    - if **num == 4** -> filter data based on **primaryNames which start with vowel character**.

Mystery_Function.py

You will need to write code for test cases #13, #14, and #15 in the `mystery_function.py` file. Here, you have a memory limitation. Meaning that now you can only load 2,000 data samples at a time. Because of this limitation, you cannot sort the entire dataset at once.

- **`def data_chuncks(file_path, columns, memory_limitation)`**

11. This function loads 2,000 data samples at a time and sorts them using `merge_sort()` that you have already implemented. Then each sorted chunk (with 2,000 data samples) will be stored as an individual output file.
    - The total dataset contains 1,84,265 data samples. Every time, you must load and sort 2,000 samples. This will result in 93 output files.
    - Note that the last file (i.e., the 93rd file) will have less than 2,000 data samples.

- All output files must be stored under a directory named '`./Individual/`'.
- Each output file must use the following name format: '`sorted_x.csv`', where x indicates the number of files starting from 1 and ending with 93.
- As a result, you will have the following output files: '`./Individual/sorted_1.csv`', '`./Individual/sorted_2.csv`', '`./Individual/sorted_3.csv`', … '`./Individual/sorted_93.csv`'

- **`def Mystery_Function(file_path, memory_limitation, columns):`**

12. This function uses 93 individually sorted files (i.e., the output files of data_chuncks()) to sort entire data samples.
    - The function also has the same memory limitation. That is, you can load and store 2,000 data samples at most.
    - You may load 2,000 data samples from any of these files you like. For example, student_1 can take 700 data samples from sorted_1.csv, 1,000 data samples from sorted_6.csv, and the remaining 300 from sorted_50.csv. Student_2 can choose 500 samples from sorted_1.csv, sorted_2.csv, sorted_3.csv, and sorted_4.csv.
    - You may have temporary files to store intermediate sorting results. The temporary files must meet the memory limitation. (At most 2,000 samples).
    - Finally, all sorted output files must be stored under a directory named '`./Final/`'.
    - There must be 93 final output files. Any temporary files must not be stored.
    - As a result, you will have the following output files: '`./Final/sorted_1.csv`', '`./Final/sorted_2.csv`', '`./Final/sorted_3.csv`', … '`./Final/sorted_93.csv`'
    - Note that the data items should be in ascending order using the given columns. **That is, if we have 10,000 numbers from 1 to 10,000. The 'Final/sorted_1.csv' file should have 1-2000 in ascending order, 'Final/sorted_2.csv' file should have 2001 – 4000 in ascending order, and so on.**

- **`chuncks_2000 :`**
    - List data structure named "**`chuncks_2000`**" where you need to store 2000 records at a time from the dataset.
    - You should only use this data structure to load the 2000 records per time from "**imd_dataset.csv**" or the file "**Individual**" incase of implementing "**data_chuncks**" as well as "**Mystery_Function**" functions.

❖ Test Cases

Test cases #1-#6

Each test case will assess an individual sorting algorithm. There are three sub-test cases that require to use of different columns for sorting.

Test cases #7-#10

Each test case will assess the computational time to sort multiple columns. There are six sub-test cases that require the use of different sorting algorithms for given columns.

Test cases #11-#12

Each test case will assess the sorting of a given column combination. There are two sub-test cases that require the implementation of merge sort and quick sort on the given column combinations.

Test cases #13-#15

Each test case will assess the efficient sorting of a given column combination with a memory limitation of 2,000 items.

For test cases 7-10, you need to extract the required data (filter on conditions mentioned above) and rename it to appropriate name as mentioned in the test case descriptions. You need to write the code to perform this at the start of the skeleton file.

- Time will be measured for a few test cases and the code to measure the time is already included in the skeleton code.
- The code to implement test case 13, 14 and 15 is given in a different file named "Mystery_Function.py"

test case 1.1	Sort 'startYear' column based on insertion sort algorithm. Use test_case_1_2.csv dataset for this
test case 1.2	Sort 'averageRating' column based on insertion sort algorithm. Use test_case_1_2.csv dataset for this
test case 1.3	Sort 'primaryTitle' column based on insertion sort algorithm. Use test_case_1_2.csv dataset for this
test case 2.1	Sort 'startYear' column based on selection sort algorithm. Use test_case_1_2.csv dataset for this
test case 2.2	Sort 'averageRating' column based on selection sort algorithm. Use test_case_1_2.csv dataset for this
test case 2.3	Sort 'primaryTitle' column based on selection sort algorithm. Use test_case_1_2.csv dataset for this
test case 3.1	Sort 'startYear' column based on quick sort algorithm

test case 3.2	Sort ‘averageRating’ column based on quick sort algorithm
test case 3.3	Sort ‘primaryTitle’ column based on quick sort algorithm
test case 4.1	Sort ‘startYear’ column based on heap sort algorithm
test case 4.2	Sort ‘averageRating’ column based on heap sort algorithm
test case 4.3	Sort ‘primaryTitle’ column based on heap sort algorithm
test case 5.1	Sort ‘startYear’ column based on shell sort algorithm
test case 5.2	Sort ‘averageRating’ column based on shell sort algorithm
test case 5.3	Sort ‘primaryTitle’ column based on shell sort algorithm
test case 6.1	Sort ‘startYear’ column based on merge sort algorithm
test case 6.2	Sort ‘averageRating’ column based on merge sort algorithm
test case 6.3	Sort ‘primaryTitle’ column based on merge sort algorithm
test case 7.1	Sort the columns ‘startYear’ and ‘primaryTitle’ using insertion sort. For this filter the dataset on ‘startYear’ column (years in range 1941 to 1955, including 1941 and 1955) and name the dataset as “imdb_years_df.csv”. startYear column should be sorted first and then on primaryTitle. Meaning all the movies should be sorted based on startYear and then those movies that have same ‘startYear’ should be sorted based on the ‘primaryTitle’
test case 7.2	Sort the columns ‘startYear’ and ‘primaryTitle’ using selection sort. For this filter the dataset on ‘startYear’ column (years in range 1941 to 1955, including 1941 and 1955) and name the dataset as “imdb_years_df.csv”. startYear column should be sorted first and then on primaryTitle. Meaning all the movies should be sorted based on startYear and then those movies that have same ‘startYear’ should be sorted based on the ‘primaryTitle’
test case 7.3	Sort the columns ‘startYear’ and ‘primaryTitle’ using quick sort. For this filter the dataset on ‘startYear’ column (years in range 1941 to 1955, including 1941 and 1955) and name the dataset as “imdb_years_df.csv”. startYear column should be sorted first and then on primaryTitle. Meaning all the movies should be sorted based on startYear and then those movies that have same ‘startYear’ should be sorted based on the ‘primaryTitle’
test case 7.4	Sort the columns ‘startYear’ and ‘primaryTitle’ using heap sort. For this filter the dataset on ‘startYear’ column (years in range 1941 to 1955, including 1941 and 1955) and name the dataset as “imdb_years_df.csv”. startYear column should be sorted first and then on primaryTitle. Meaning all the movies should be sorted based on startYear and then those movies that have same ‘startYear’ should be sorted based on the ‘primaryTitle’
test case 7.5	Sort the columns ‘startYear’ and ‘primaryTitle’ using shell sort. For this filter the dataset on ‘startYear’ column (years in range 1941 to 1955, including 1941 and 1955) and name the dataset as “imdb_years_df.csv”. startYear column should be sorted first and then on primaryTitle. Meaning all the movies should be sorted based on startYear and then those movies that have same ‘startYear’ should be sorted based on the ‘primaryTitle’

test case 7.6	Sort the columns ‘startYear’ and ‘primaryTitle’ using merge sort. For this filter the dataset on ‘startYear’ column (years in range 1941 to 1955, including 1941 and 1955) and name the dataset as “imdb_years_df.csv”. startYear column should be sorted first and then on primaryTitle. Meaning all the movies should be sorted based on startYear and then those movies that have same ‘startYear’ should be sorted based on the ‘primaryTitle’
test case 8.1	Sort the columns ‘startYear’, ‘runningMinutes’ and ‘primaryTitle’ using insertion sort algorithm. For this filter the dataset where genres are either ‘Adventure’ or ‘Drama’ and name the resulting dataset as” imdb_genres_df.csv”. Sorting should be done first on ‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’, and those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’
test case 8.2	Sort the columns ‘startYear’, ‘runningMinutes’ and ‘primaryTitle’ using selection sort algorithm. For this filter the dataset where genres are either ‘Adventure’ or ‘Drama’ and name the resulting dataset as” imdb_genres_df.csv”. Sorting should be done first on ‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’, and those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’
test case 8.3	Sort the columns ‘startYear’, ‘runningMinutes’ and ‘primaryTitle’ using quick sort algorithm. For this filter the dataset where genres are either ‘Adventure’ or ‘Drama’ and name the resulting dataset as” imdb_genres_df.csv”. Sorting should be done first on ‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’, and those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’
test case 8.4	Sort the columns ‘startYear’, ‘runningMinutes’ and ‘primaryTitle’ using heap sort algorithm. For this filter the dataset where genres are either ‘Adventure’ or ‘Drama’ and name the resulting dataset as” imdb_genres_df.csv”. Sorting should be done first on ‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’, and those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’
test case 8.5	Sort the columns ‘startYear’, ‘runningMinutes’ and ‘primaryTitle’ using shell sort algorithm. For this filter the dataset where genres are either ‘Adventure’ or ‘Drama’ and name the resulting dataset as” imdb_genres_df.csv”. Sorting should be done first on

	‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’, and those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’
test case 8.6	Sort the columns ‘startYear’, ‘runningMinutes’ and ‘primaryTitle’ using merge sort algorithm. For this filter the dataset where genres are either ‘Adventure’ or ‘Drama’ and name the resulting dataset as” imdb_genres_df.csv”. Sorting should be done first on ‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’, and those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’
test case 9.1	Sort the columns ‘startYear’, ‘runtimeMinutes’ and ‘primaryTitle’ using insertion sort algorithm. For this filter the dataset where ‘primaryProfession’ column contains substrings {‘assistant_director’, ‘casting_director’, ‘art_director’, ‘cinematographer’} and name the resulting dataset as “imdb_professions_df.csv”. Sorting should be done first on ‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’ and then those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’
test case 9.2	Sort the columns ‘startYear’, ‘runtimeMinutes’ and ‘primaryTitle’ using selection sort algorithm. For this filter the dataset where ‘primaryProfession’ column contains substrings {‘assistant_director’, ‘casting_director’, ‘art_director’, ‘cinematographer’} and name the resulting dataset as “imdb_professions_df.csv”. Sorting should be done first on ‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’ and then those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’
test case 9.3	Sort the columns ‘startYear’, ‘runtimeMinutes’ and ‘primaryTitle’ using quick sort algorithm. For this filter the dataset where ‘primaryProfession’ column contains substrings {‘assistant_director’, ‘casting_director’, ‘art_director’, ‘cinematographer’} and name the resulting dataset as “imdb_professions_df.csv”. Sorting should be done first on ‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’ and then those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’

test case 9.4	Sort the columns ‘startYear’, ‘runtimeMinutes’ and ‘primaryTitle’ using heap sort algorithm. For this filter the dataset where ‘primaryProfession’ column contains substrings {‘assistant_director’, ‘casting_director’, ‘art_director’, ‘cinematographer’} and name the resulting dataset as “imdb_professions_df.csv”. Sorting should be done first on ‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’ and then those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’
test case 9.5	Sort the columns ‘startYear’, ‘runtimeMinutes’ and ‘primaryTitle’ using shell sort algorithm. For this filter the dataset where ‘primaryProfession’ column contains substrings {‘assistant_director’, ‘casting_director’, ‘art_director’, ‘cinematographer’} and name the resulting dataset as “imdb_professions_df.csv”. Sorting should be done first on ‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’ and then those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’
test case 9.6	Sort the columns ‘startYear’, ‘runtimeMinutes’ and ‘primaryTitle’ using merge sort algorithm. For this filter the dataset where ‘primaryProfession’ column contains substrings {‘assistant_director’, ‘casting_director’, ‘art_director’, ‘cinematographer’} and name the resulting dataset as “imdb_professions_df.csv”. Sorting should be done first on ‘startYear’, then on ‘runtimeMinutes’ and finally on ‘primaryTitle’. Meaning all the movies should be sorted based on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted based on ‘runtimeMinutes’ and then those having the same ‘runtimeMinutes’ should be sorted based on ‘primaryTitle’
test case 10.1	Sort the columns ‘startYear’, ‘averageRating’, ‘category’ and ‘primaryTitle’ using insertion sort. For this filter the dataset where primary Names start with vowels, and then name the resulting dataset as “imdb_vowel_names_df.csv”. Sorting should be done first on ‘startYear’, ‘averageRating’, ‘category’ and then on ‘primaryTitle’. Meaning all the movies should be sorted on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted on ‘averageRating’, and those having same ‘averageRating’ should be sorted on ‘category’, and those having same ‘category’ should be sorted on ‘primaryTitle’
test case 10.2	Sort the columns ‘startYear’, ‘averageRating’, ‘category’ and ‘primaryTitle’ using selection sort. For this filter the dataset where primary Names start with vowels, and then name the resulting dataset as “imdb_vowel_names_df.csv”. Sorting should be done first on ‘startYear’, ‘averageRating’, ‘category’ and then on

	‘primaryTitle’. Meaning all the movies should be sorted on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted on ‘averageRating’, and those having same ‘averageRating’ should be sorted on ‘category’, and those having same ‘category’ should be sorted on ‘primaryTitle’
test case 10.3	Sort the columns ‘startYear’, ‘averageRating’, ‘category’ and ‘primaryTitle’ using quick sort. For this filter the dataset where primary Names start with vowels, and then name the resulting dataset as “imdb_vowel_names_df.csv”. Sorting should be done first on ‘startYear’, ‘averageRating’, ‘category’ and then on ‘primaryTitle’. Meaning all the movies should be sorted on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted on ‘averageRating’, and those having same ‘averageRating’ should be sorted on ‘category’, and those having same ‘category’ should be sorted on ‘primaryTitle’
test case 10.4	Sort the columns ‘startYear’, ‘averageRating’, ‘category’ and ‘primaryTitle’ using heap sort. For this filter the dataset where primary Names start with vowels, and then name the resulting dataset as “imdb_vowel_names_df.csv”. Sorting should be done first on ‘startYear’, ‘averageRating’, ‘category’ and then on ‘primaryTitle’. Meaning all the movies should be sorted on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted on ‘averageRating’, and those having same ‘averageRating’ should be sorted on ‘category’, and those having same ‘category’ should be sorted on ‘primaryTitle’
test case 10.5	Sort the columns ‘startYear’, ‘averageRating’, ‘category’ and ‘primaryTitle’ using shell sort. For this filter the dataset where primary Names start with vowels, and then name the resulting dataset as “imdb_vowel_names_df.csv”. Sorting should be done first on ‘startYear’, ‘averageRating’, ‘category’ and then on ‘primaryTitle’. Meaning all the movies should be sorted on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted on ‘averageRating’, and those having same ‘averageRating’ should be sorted on ‘category’, and those having same ‘category’ should be sorted on ‘primaryTitle’
test case 10.6	Sort the columns ‘startYear’, ‘averageRating’, ‘category’ and ‘primaryTitle’ using merge sort. For this filter the dataset where primary Names start with vowels, and then name the resulting dataset as “imdb_vowel_names_df.csv”. Sorting should be done first on ‘startYear’, ‘averageRating’, ‘category’ and then on ‘primaryTitle’. Meaning all the movies should be sorted on ‘startYear’ and then those movies that have the same ‘startYear’ should be sorted on ‘averageRating’, and those having same ‘averageRating’ should be sorted on ‘category’, and those having same ‘category’ should be sorted on ‘primaryTitle’
test case 11.1	Sort the columns ‘startYear’ and ‘primaryTitle’ using merge sort on “imdb_dataset.csv”. Sorting should be applied first on ‘startYear’

	and then to 'primaryTitle'. Meaning after sorting 'startYear', then all the movies with same 'startYear' should be sorted based on 'primaryTitle'
test case 11.2	Sort the columns 'startYear' and 'primaryTitle' using quick sort on "imdb_dataset.csv". Sorting should be applied first on 'startYear' and then to 'primaryTitle'. Meaning after sorting 'startYear', then all the movies with same 'startYear' should be sorted based on 'primaryTitle'
test case 12.1:	Sort the columns 'startYear', 'averageRating', 'category' and 'primaryTitle' using merge sort algorithm. Sorting should be applied first on 'startYear' then on 'averageRating', then on 'category' and then on 'primaryTitle'. Meaning after sorting on 'startYear', movies having same 'startYear' should be sorted on 'averageRating', after that sort, movies having the same 'averageRating' should be sorted on 'category' and finally movies having the same 'category' should be sorted on 'primaryTitle'
test case 12.2	Sort the columns 'startYear', 'averageRating', 'category' and 'primaryTitle' using quick sort algorithm. Sorting should be applied first on 'startYear' then on 'averageRating', then on 'category' and then on 'primaryTitle'. Meaning after sorting on 'startYear', movies having same 'startYear' should be sorted on 'averageRating', after that sort, movies having the same 'averageRating' should be sorted on 'category' and finally movies having the same 'category' should be sorted on 'primaryTitle'
test case 13	Sort the 'startYear' column in "imdb_dataset.csv" using merge sort. But the constraint you have is memory limitation of 2000. Meaning you can only sort at most 2000 items at a time. Meaning if the dataset is 10,000 items long, only 2000 can be loaded at a time and be compared. And then the next 2000 can be loaded and compared that
test case 14	Sort the 'primaryTitle' column from "imdb_dataset.csv" using merge sort algorithm. But the constraint you have is a memory limitation of 2000. Meaning you can only sort at most of 2000 items at a time. Meaning if the dataset is 10,000 items long, only 2000 can be loaded at a time and be compared. And then the next 2000 can be loaded and compared
test case 15	Apply merge sort on 'imdb_dataset.csv'. You have a memory constraint where you can load and/or sort at most of 2000 items at a time. Sort on columns = {'startYear', 'runtimeMinutes', 'primaryTitle'}. To explain the constraint in detail, if the dataset is 10,000 items long, only 2000 can be loaded at a time and be compared. More details on what and how to implement in mystery_function.py file will be given below.

❖Submission

All the coding should be done only in ***python*** language. Once done, please compress your python codes as a **ZIP** file and submit it on Canvas. Do not include other files such as dataset CSV files.

Discussion about the project is totally encouraged. However, all the submissions will be checked for plagiarism and any instance of copying or reusing other students' code is prohibited and will have serious consequences. You will be reported as well as getting a 0 for the project, and getting a grade 'F' for the course.

❖ Grading

The submitted code will be tested on different test cases and points will be assigned based on the correctness and efficiency to sort, meaning time complexity to sort. A few of the test cases are provided for you to check the correctness and efficiency of your code. In addition, a few test cases which are not provided to you will also be used to grade your submission.

Late submissions for up to a week will be accepted with a penalty of **10%** reduction per late day (including weekends).