Part 1 – MovieMate: project and database

This project, **MovieMate**, is designed to manage a movie dataset based on user input. The dataset is stored in a dictionary format, where each movie's name is a key, and its details, such as rating, duration, release year, genre, and gross income, are stored as values in a tuple. Users can interact with the system through a menu to add movies, calculate the average gross income, display rating statistics, and check the movies filtered by genre.

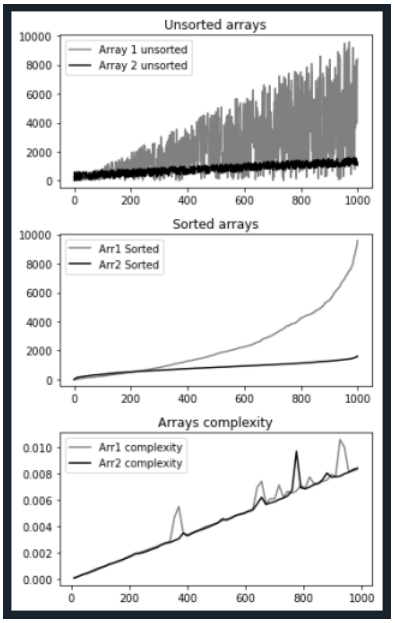
In addition, the dataset can be dynamically modified by the users. Functions that manipulate the dataset are stored in the "database.py". This ensures that the user can grow and customize the project, rather than being limited to pre-defined data. What is more, the project is integrated with NumPy and Matplotlib, thus rating analysis is more suitable for the user, compared to the text.

Project functionalities:

The user can choose among 5 options:

- Option 1 **(Dictionary):** Calls the function **add_movie**(name = str, rating = float, duration = int, year = int, genre = str, gross_income = int) that modifies the database file "movies.txt" by adding a new movie line which is saved as a dictionary.
- Option 2 **(Tuple):** Calls the function **average_gross_income**(data = dict) to compute and return the average gross income of all movies in the dataset.
- Option 3 **(Array):** Calls the function **rating_statistics**(data = dict) that adds movies rating to the list and then makes an array. It also divides the ratings from 0 to 10 and makes a bar graph, that shows the highest frequency of movies in specific rating and colour it on red. User can see the rating distribution.
- Option 4 **(Dictionary):** Calls the function **genre_films**(data = dict, genre = str) that prints a list of films in the given genre.
- Option 5: Exit the program

Part 2 – Discussion:



At the beginning of my Big-O analysis, I expected that my sorting function (Quick Sort) would show an average complexity according to this algorithm, such as O(n log n). The time should be increased nonlinearly but steadily, and with a bigger dataset and poorly chosen pivot, it should look similar to its worst-case complexity O(n^2).

The shown plots support my expectations. The complexity plot shows that the sorting time increases with data size. The plot of unsorted arrays depicts the irregular behaviour of the arrays. However, the sorted arrays plot displays a smooth increasing trend, which is correct. What is more, these results align with the theory and confirm that Quick Sort is efficient for average cases and can be less efficient in bigger problems. In addition, if another algorithm, such as Bubble Sort, were to be used, the results could differ in time. In conclusion, the observed results match my expectations of Quick Sort, and the plots clearly visualize the sorting process and its time complexity.

Acknowledgements:

I would like to thank my housemate Emilia Załęska with the help on discussing the conceptual topics of this project.

Authorship declaration:

I declare that the work submitted here is from my authorship only. I have not used any generative AI to help with any code/text included in my work. I have given credit for the help I had conceptualizing my project. My work respects the university and course code of conduct.