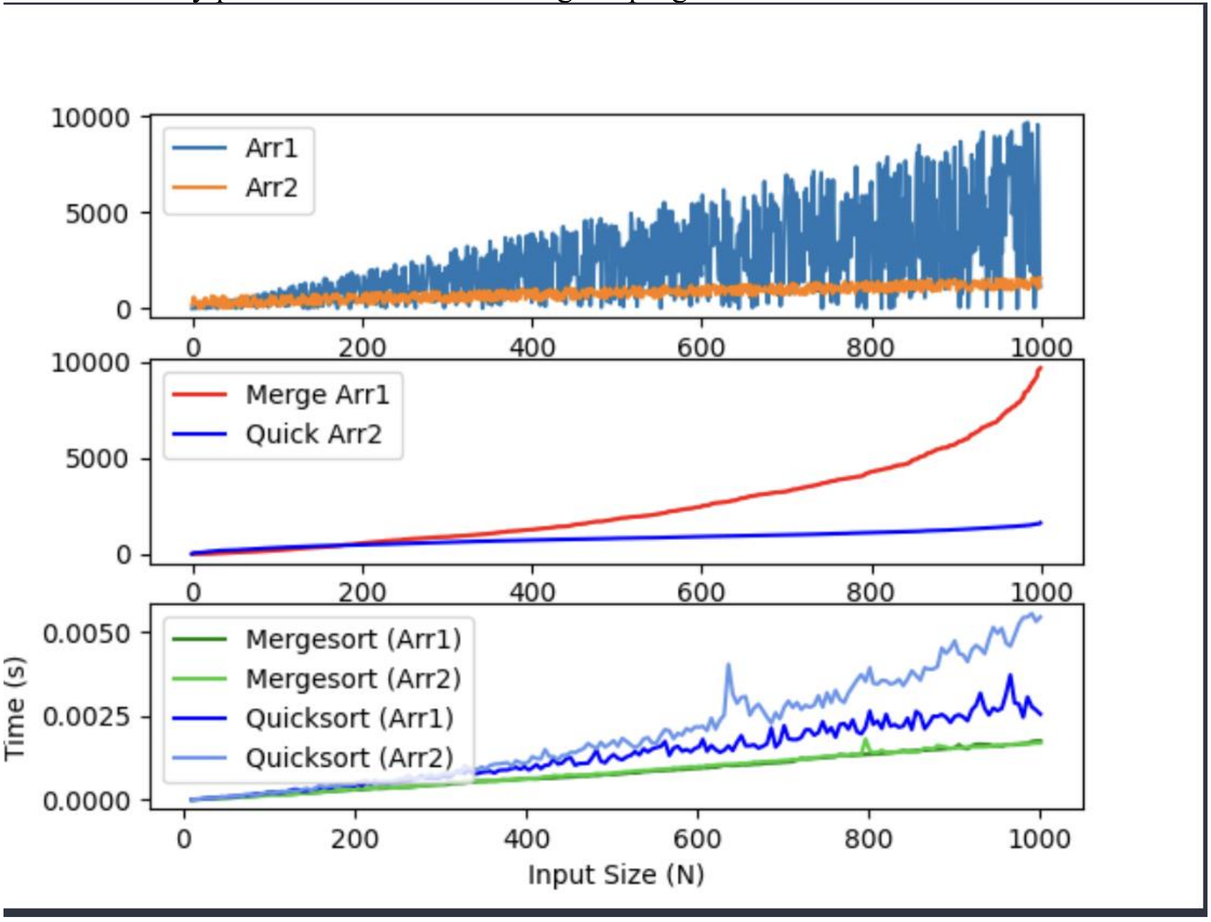
Programming Project - 3

Discussion (Part 2)

This is what my plot looks like after running the program.



Based on Big-O analysis:

- **Mergesort** has O(n log n) time complexity in all cases (best, average, and worst). So I expected it to show steady growth regardless of the input data.
- **Quicksort** has O(n log n) average case but $O(n^2)$ worst case. I expected it to be faster than Mergesort on average, but potentially much slower on certain types of data.

The results match what I expected from Big-O analysis:

Mergesort behaves consistently on both arrays because it always divides the array in half and merges, regardless of whether the data is sorted or random. This is why both green lines grow steadily.

Quicksort performance depends heavily on the data:

- On **Arr2** (nearly sorted data), Quicksort is very efficient because it can partition well

- On **Arr1** (random data with high variance), Quicksort struggles and shows spikes. This suggests it's hitting closer to its $O(n^2)$ worst case, especially when the pivot choices are poor

Goodreads Book Database (Part 3)

Student Name: Anton Weizmann

Student Number: s1124376

Date: 07.01.2026

Project Description

This is a book database manager that uses the Goodreads dataset, but any other data set with the same header would work. It lets you add books, search for them, get recommendations, find books by author, and see some statistics about all the books.

The program saves everything to a text file (goodreads.txt) so your changes are kept when you restart it.

How to use:

- Run: `python part3.py`
- Pick an option from the menu (1-6)
- Follow the instructions
- Choose option 6 to save and quit

Data Structures Used

Dictionary

Where: Main `database` variable
Functions: add_book(), find_book(), authors_books(), book_recommendation(), database_statistics()

Example:

```
database[title] = list(book_data)  # storing a book
if search in database:             # searching for a book
```

I used a dictionary because it's fast to look up books by their title. I understand that duplicates of the same book could pose a problem because of this, but this program and the corresponding database are designed for every book to only be included once.

Tuple

Where: get_new_book() function
Example:

```
book_data = (authors, rating, language, pages, ratings_count, pub_date)
```

I used a tuple to store the book info temporarily because tuples can't be changed accidentally.

NumPy Array

Where: database_statistics() function
Example:

```
ratings_array = np.array(ratings)
print(f"Average rating: {np.mean(ratings_array):.2f}")
```

I used numpy arrays to calculate statistics like average, median, and standard deviation easily.

Project Functionalities

Option 1: Add a new book

Asks you for book info and adds it to the database. Uses **dictionary** and **tuple**.

Option 2: Find a book by title

Type in a book title and it shows you all the info. Uses **dictionary**.

Option 3: Get recommendations

Enter a minimum rating and max pages, and it shows you matching books. Uses **dictionary**.

Option 4: Find books by author

Type an author name and see all their books. Uses **dictionary**.

Option 5: View statistics

Shows you stats like average rating, longest book, most popular book, etc. Uses **numpy array**.

Option 6: Save and exit

Saves everything and quits.

Authorship Declaration

I declare that the work submitted here is from my authorship only. I haven't 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.