Task 1 – "Ordnung must sein” Report

Task:

B. The Olsen Gang has obtained a set of 20k credit card details. Unfortunately, when transferring through the dark web, due to a sloppy internet connection, the data set got split into two parts. Moreover, the card records in the second part got randomly shuffled, so, before they can sell the data for big money, they must match them up with card numbers from the first part. Luckily, this is possible, as it is known that the records in the first dataset go in the order of increasing expiration dates and the PIN. Choose a linear time algorithm for the job and help the gang cash out.
Empirically investigate at what data volume the linear solution gains supremacy over a log-linear algorithm.

Introduction:

This report presents an empirical comparison of linear and log-linear algorithms in the case of one data set only containing the first twelve digits of the credit card number sorted, and the other data set containing all the information but only with the last four digits of the credit card number unsorted.

Approach:

- Sort the file2 according to expiry date and PIN (ascending order).
- Merge file1 and sorted2.
- Run the solution in linear time (O(n)) and the sorting-based log-linear (O(n log n)) approach.

Algorithms:

Log-Linear:

- Sorting the part2 will take O(n) time.
- Merging part1 and part2 will take O(log n) time.
- Overall time complexity become O(n log n) time.

Linear:

- For the linear approach to work at O(n) time bot part1 and part2 must be sorted.
- Merging part1 and part2 will directly match part1 and part2 and since the rate of growth is constant the time complexity will be O(n).

Setup:

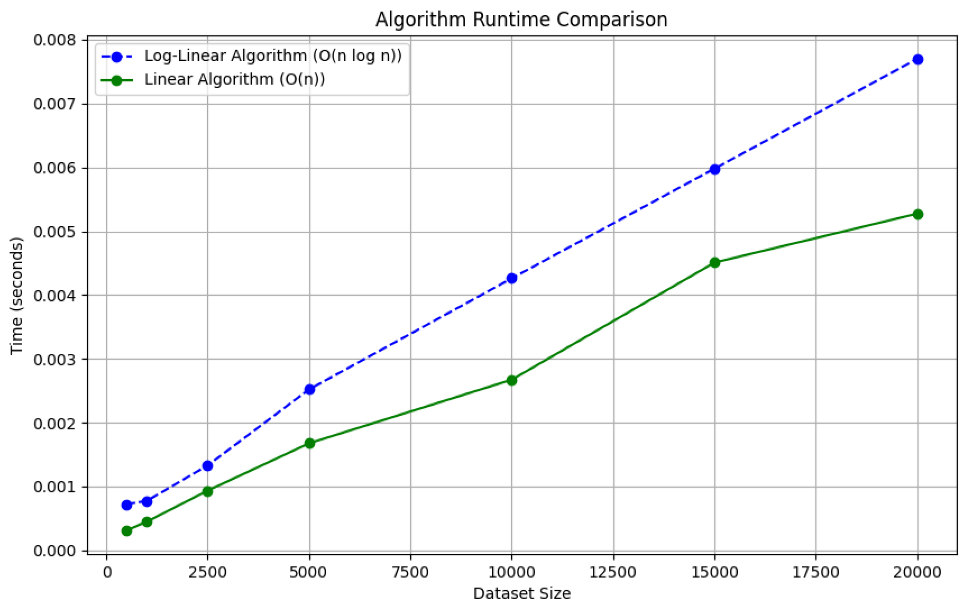
- Datasets of increasing size: 1000, 5000, 10000, 15000, 20000 (20000 data in total in each part) records.
- Time measurements taken for both algorithms.
- Comparison plotted on a graph.

Result:

Here is the table with input sizes and their times for each sorting algorithm:

Dataset Size	Log-Linear Time (O(n log n))	Linear Time (O(n))
500	0.0007250308990478516	0.0003130435943603516
1000	0.0007758140563964844	0.0004498958587646484
2500	0.0013308525085449219	0.0009357929229736328
5000	0.002521038055419922	0.0016770362854003906
10000	0.004261970520019531	0.0026738643646240234
15000	0.005979776382446289	0.004508018493652344
20000	0.0076999664306640625	0.005274772644042969

Here is the graph:



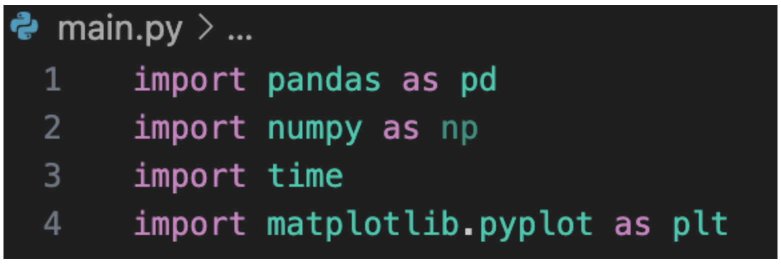
Findings:

- Up to 5000 datasets both log-linear and linear approach perform well and close in time.
- After 5000 linear algorithm starts to gain supremacy against the log-linear algorithm.
- At 20000 datasets linear algorithm it is seen that the linear algorithm almost halves the time of log-linear algorithm
- Small note: I do not completely understand the linear approach’s time at 15000 datasets.

Conclusion:

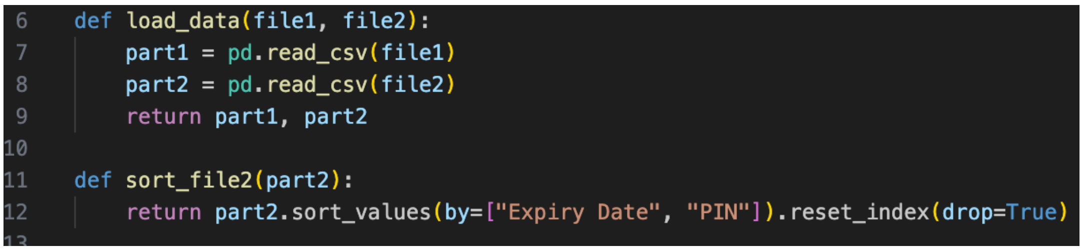
The linear approach is the optimal choice for large datasets, as it avoids repeated sorting.

Code Analysis and Explanation:
Main.py:



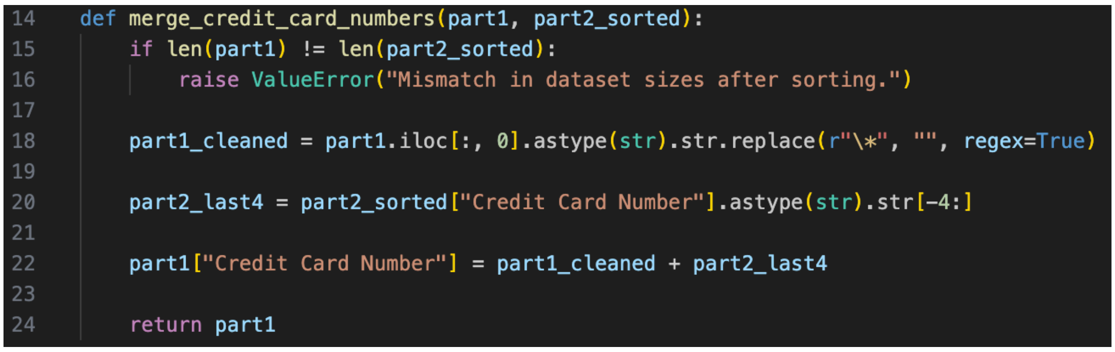
```python
import pandas as pd
import numpy as np
import time
import matplotlib.pyplot as plt
```

- Importing libraries to plot the graphs, tables, measure the time.



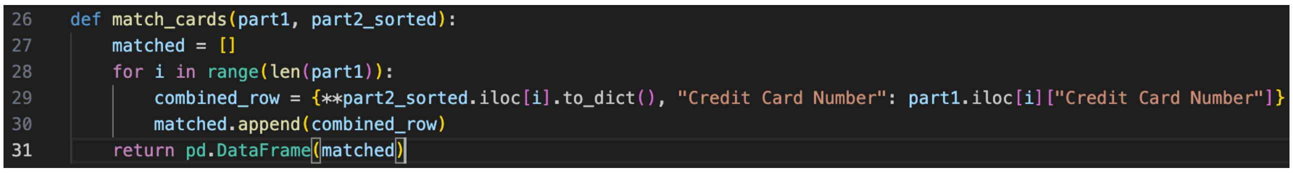
```python
def load_data(file1, file2):
    part1 = pd.read_csv(file1)
    part2 = pd.read_csv(file2)
    return part1, part2

def sort_file2(part2):
    return part2.sort_values(by=["Expiry Date", "PIN"]).reset_index(drop=True)
```

- Load the datas from the cvs files to my code and name them as part1 and part2.
- As only the second file is not sorted, sort the part2 by expiry date and PIN.



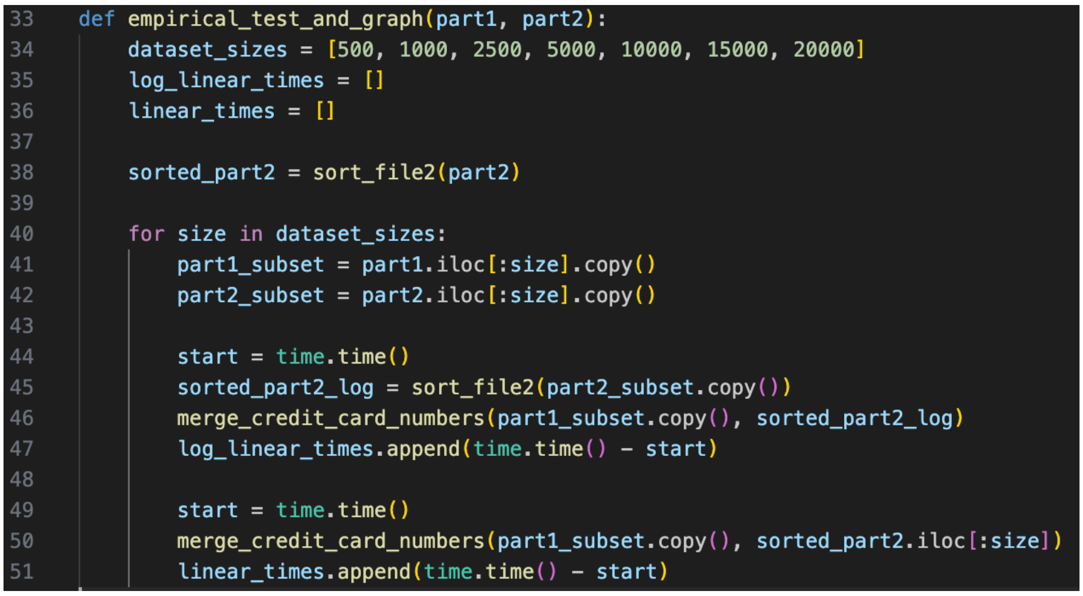
```python
def merge_credit_card_numbers(part1, part2_sorted):
    if len(part1) != len(part2_sorted):
        raise ValueError("Mismatch in dataset sizes after sorting.")

    part1_cleaned = part1.iloc[:, 0].astype(str).str.replace(r"\*", "", regex=True)

    part2_last4 = part2_sorted["Credit Card Number"].astype(str).str[-4:]

    part1["Credit Card Number"] = part1_cleaned + part2_last4

    return part1
```

- Checks if the length of part1 and part2 is the same before merging to see if there was a problem with loading data or sorting to prevent mistakes.
- Gets rid of the * symbol to make the merged credit card number correctly shown.
- Merges the cleaned part1 and the last 4 digits from part2 into part1 to create the cvs file with merged data.



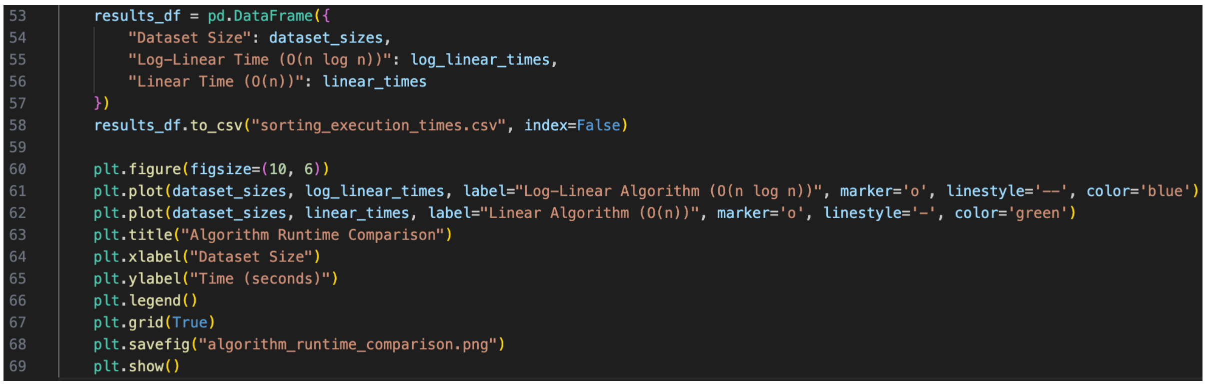
```python
def match_cards(part1, part2_sorted):
    matched = []
    for i in range(len(part1)):
        combined_row = {**part2_sorted.iloc[i].to_dict(), "Credit Card Number": part1.iloc[i]["Credit Card Number"]}
        matched.append(combined_row)
    return pd.DataFrame(matched)
```

- Combines the merged credit card number with all the other data from part2 row by row.



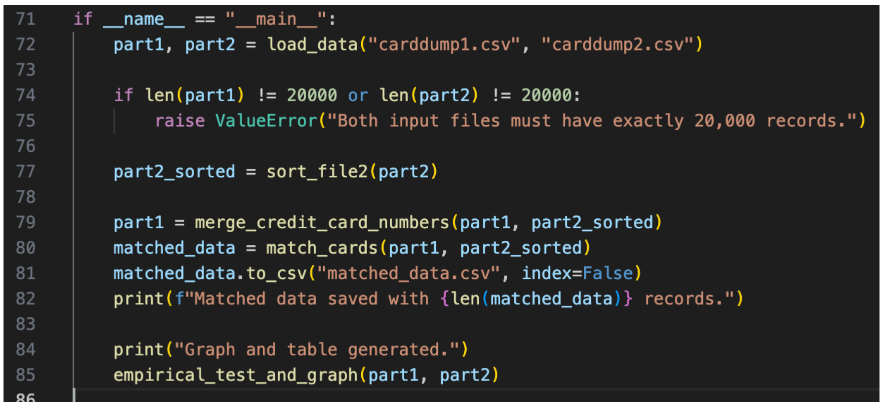
```python
def empirical_test_and_graph(part1, part2):
    dataset_sizes = [500, 1000, 2500, 5000, 10000, 15000, 20000]
    log_linear_times = []
    linear_times = []

    sorted_part2 = sort_file2(part2)

    for size in dataset_sizes:
        part1_subset = part1.iloc[:size].copy()
        part2_subset = part2.iloc[:size].copy()

        start = time.time()
        sorted_part2_log = sort_file2(part2_subset.copy())
        merge_credit_card_numbers(part1_subset.copy(), sorted_part2_log)
        log_linear_times.append(time.time() - start)

        start = time.time()
        merge_credit_card_numbers(part1_subset.copy(), sorted_part2.iloc[:size])
        linear_times.append(time.time() - start)
```

- Function defines a dataset of ascending sizes up to 20000.
- Stores execution times for log-linear and linear approach.
- Sorts the part2 again before the linear approach to justify it’s O(n) time as it requires that the arrays are sorted.
- Times the log-linear approach.
- Times the linear approach.



```python
    results_df = pd.DataFrame({
        "Dataset Size": dataset_sizes,
        "Log-Linear Time (O(n log n))": log_linear_times,
        "Linear Time (O(n))": linear_times
    })
    results_df.to_csv("sorting_execution_times.csv", index=False)

    plt.figure(figsize=(10, 6))
    plt.plot(dataset_sizes, log_linear_times, label="Log-Linear Algorithm (O(n log n))", marker='o', linestyle='--', color='blue')
    plt.plot(dataset_sizes, linear_times, label="Linear Algorithm (O(n))", marker='o', linestyle='-', color='green')
    plt.title("Algorithm Runtime Comparison")
    plt.xlabel("Dataset Size")
    plt.ylabel("Time (seconds)")
    plt.legend()
    plt.grid(True)
    plt.savefig("algorithm_runtime_comparison.png")
    plt.show()
```

- Saves the result in a cvs table form and png graph.



```python
if __name__ == "__main__":
    part1, part2 = load_data("carddump1.csv", "carddump2.csv")

    if len(part1) != 20000 or len(part2) != 20000:
        raise ValueError("Both input files must have exactly 20,000 records.")

    part2_sorted = sort_file2(part2)

    part1 = merge_credit_card_numbers(part1, part2_sorted)
    matched_data = match_cards(part1, part2_sorted)
    matched_data.to_csv("matched_data.csv", index=False)
    print(f"Matched data saved with {len(matched_data)} records.")

    print("Graph and table generated.")
    empirical_test_and_graph(part1, part2)
```

- Main function of the code checking and performing every stage previously described.